# Incorporating Graph Domain Structure into Matrix Representations and Optimal-Transport Kernels

by

**Keishi Sando**

**Dissertation**

submitted to the Statistical Science Program

in partial fulfillment of the requirements for the degree of

***Doctor of Philosophy***

SOKENDAI

Graduate Institute for Advanced Studies, SOKENDAI

March 2026

# Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Professor Hideitsu Hino at the Institute of Statistical Mathematics (ISM). Professor Hino has guided me since my enrollment at the University of Tsukuba as my class advisor. It was also his lecture in an omnibus course that first sparked my interest in machine learning. I feel that he has led me from the very entrance of academia to where I stand today. When I joined his laboratory and boldly expressed my desire to work on information geometry, his thoughtful advice enabled me to embark on this research path. Throughout my research, from initial brainstorming to manuscript preparation, I have repeatedly benefited from his extensive knowledge and experience, which have given me the confidence to submit this dissertation. While I cannot claim to have taken full advantage of all the guidance and opportunities Professor Hino has provided, I hope that this dissertation, and my future research endeavors, will serve as a testament to my gratitude for his unwavering support over these years.

I would also like to express my sincere gratitude to the members of my dissertation committee: Professor Shuhei Mano (ISM), Professor Satoshi Hara (UEC), and Associate Professor Tasuku Soma (ISM). Professor Mano carefully reviewed the entire dissertation and pointed out numerous areas for improvement, down to the details of the proofs. Professor Hara identified several ambiguous sections in the manuscript and helped elevate the overall quality of the work. Associate Professor Soma provided insightful advice on clarifying the positioning of this work relative to existing research, drawing on his deep knowledge and experience in his field.

I am deeply grateful to Assistant Professor Tam Le (ISM), who made significant contributions as a co-author. The idea for the second paper stemmed from his prior research, and he provided precise and valuable comments on both the organization of

the manuscript and the details of the proofs. I am confident that without his assistance, the paper would not have reached its current level of quality.

I would also like to thank the faculty members at ISM, especially Associate Professor Yano, Assistant Professor Okuno, Assistant Professor Yagishita, and Assistant Professor Okazaki, as well as Ms. Fujiwara, who serves as secretary in Professor Hino's laboratory, and the graduate office staff at ISM. The faculty members have provided invaluable advice on research matters and engaged in wide-ranging conversations about student life. Without Ms. Fujiwara's support in handling administrative matters since my master's program at the University of Tsukuba, I would not have been able to concentrate on my research. The graduate office staff have kindly accommodated my numerous requests related to my activities at ISM.

I am grateful to my peers at the ISM, the National Institute of Japanese Language and Linguistics, and the National Institute of Polar Research for their support not only in academic matters but also in daily life. Dr. Kawashima, Dr. Kimura, and Dr. Sagawa served as senior colleagues and role models, while Mr. Iwasaki provided guidance not only in research activities but also as a mentor in life. The vibrant community in the graduate student room made my graduate school experience truly enjoyable, and the international students' patience with my limited English helped me gain confidence in communicating across languages. I extend my heartfelt thanks to everyone who has been involved in my student life.

Finally, I would like to express my deepest gratitude to my family for their constant support.

# Abstract

In the modern era of increasingly large and complex data, graph-structured data, which describes the relationships between entities, has become a crucial data format across a wide range of fields. However, the methods for analyzing this data do not always meet the demands imposed by its modern scale and complexity, creating a need for new techniques that can efficiently capture its essential structure.

For instance, in communication graphs arising from social networks, enterprise messaging, or service logs on servers, uncovering the latent structures such as communities, roles, and service influence regions requires appropriate visualizations and metrics that reflect the underlying structure. A naive visualization that does not consider the structure may collapse important features of the graph, such as bipartiteness or hierarchical structures, resulting in uninterpretable hairballs.

Moreover, in graph data resembling the modern internet, where traffic is concentrated on services provided by a few IT companies, it is known that using standard centrality measures or walk-based analysis can lead to overestimating the importance of hub nodes and their neighborhoods. This issue calls for methods that remain robust under heavy-tailed degree distributions and hub-dominant structures.

Simultaneously, the advancement of the information society in recent years has led to the rapid growth of graph data, increasing the need for scalable methods in addition to those capable of capturing complex structures. This dissertation aims to discover how to effectively uncover information inherent in graph data, with a focus on two topics.

First, we discuss matrix representations for capturing the structure of directed graphs and their applications. In spectral graph theory, which explores graph structures through the analysis of matrix representations, research has primarily focused on

undirected graphs, while efforts on directed graphs have been limited due to the challenges posed by their asymmetry. However, in recent years, matrix representations that utilize complex numbers to express directional information have been proposed, and their usefulness has been confirmed in applications such as clustering problems. We point out that existing methods are vulnerable in sparse situations commonly found in real-world data, and propose a new matrix representation that retains robust cluster information in its eigenvectors even in such situations. We demonstrate that the proposed matrix representation possesses favorable properties analogous to the relationships between matrix representations in undirected graphs, and further explore the connection between the proposed representation and clustering through discussions on belief propagation.

Next, we address the acceleration of graph kernels for evaluating graph similarity. Since graph data does not exhibit homogeneous structures such as grids or sequences, graph kernels are useful frameworks for handling such data in machine learning. However, evaluating differences in the structures of graphs often requires capturing both local and global structures, which can lead to high computational costs. In particular, graph kernels based on optimal transport, which have been proposed in recent years, offer high discriminative power but are challenging to apply to large-scale graph datasets. In this dissertation, we focus on the categorical Wasserstein Weisfeiler-Lehman graph kernel, and show that by leveraging the structure of the features used in this kernel, we can achieve faster and more accurate computations than conventional algorithms.

The methods developed in this work enable (i) the robust recovery of community structures in sparse and hub-dominant directed graphs by introducing the complex non-backtracking matrix and associated spectral techniques, and (ii) the exact yet faster computation of the categorical Wasserstein Weisfeiler-Lehman graph kernel, one of the recently proposed graph kernels with high discriminative power, by leveraging the structure of its features. These achievements contribute to improving the stability and interpretability of visualizations in directed graphs, while also expanding the applicability of expressive optimal transport-based graph kernels to large-scale datasets.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

Advances in software and hardware, together with the maturation of machine learning theory, have led to data that are both large in scale and rich in relational structure. Examples include social networks formed by user interactions, product–user relations derived from browsing and purchase histories, and molecular graphs or protein–protein interaction networks in the natural sciences. Even when such relations are not explicitly given, many phenomena can naturally be modeled as graphs whose vertices represent entities and whose edges encode their relationships.

Analyzing graph structures plays a crucial role in inference and practical interventions. For instance, in communication graphs derived from social networks or workplace messaging, grouping nodes based on "who communicates with whom and how frequently" can reveal communities and the roles of individual nodes. Based on the characteristics of these groups, one can optimize notifications and interventions or control information flow. For example, when evaluating whether products or information favored in community A are also favored in community B, one can estimate the effect with minimal influence by notifying a few users located at the boundary

between A and B.

While the latent structure of targets provides important information for downstream tasks, the direct application of conventional machine learning methodologies is challenged by properties inherent to graph data. These include the lack of a uniform grid structure, as seen in vectors or images, a variable number of vertices, the absence of a canonical ordering over vertices, and asymmetric relationships. Moreover, it has been known that the presence of singular entities, such as hub nodes, can distort the results of standard methods. In this context, two methodologies that have provided an important mathematical foundation for graph data analysis are graph kernels based on similarity design and spectral methods for graphs.

## 1.1 Graph Kernels Based on Predefined Similarities

By designing a kernel function that measures similarity between graphs, one can apply the framework of kernel methods. Classically, the family of R-convolution kernels (Haussler 1999) defines similarities based on enumerating substructures of graphs and aggregating them. These methods are highly interpretable, yet suffer from computational costs that grow with the number of vertices. For example, the shortest-path kernel (Borgwardt and Kriegel 2005) compares pairwise shortest paths across two graphs. Computing all-pairs shortest paths with the Floyd-Warshall algorithm requires $O(n^3)$ time, where $n$ is the number of vertices in the graph. Another example is the $k$-node graphlet kernel (Pržulj 2007). This kernel measures graph similarity by counting the occurrences of small subgraphs called graphlets, typically with $k \in \{4, 5\}$ vertices, and comparing their frequencies between graphs. However, obtaining these counts requires enumerating all size-$k$ subgraphs contained in a graph with $n$ vertices, of which there are $O(n^k)$, or sampling from them.

To address this computational cost, the Weisfeiler-Lehman (WL) graph kernel (Shervashidze et al. 2011) generates label sequences through neighborhood aggregation and relabeling. It achieves a practical balance between representational power and computational efficiency with $O(Hm)$ time, where $H$ is the number of WL iterations and $m$ is the number of edges in the graph. However, it has been noted that simple aggregation can discard higher-order distributional information, including co-occurrence patterns and attribute dispersion (Togninalli et al. 2019).

To mitigate this loss, optimal transport has been incorporated into graph kernels. The Wasserstein Weisfeiler-Lehman (WWL) graph kernel (Togninalli et al. 2019) compares distributions over WL label sequences using optimal transport. The fused Gromov-Wasserstein method jointly aligns node attributes and structural information within a single objective (Titouan et al. 2019). Another line of work uses adjacency-based Markov kernels to progressively push label distributions into a hierarchical probability distribution, and then computes the Wasserstein distance by solving optimal transport problems on vertex pairs iteratively (S. Chen et al. 2022). Although these methods, which capture both local and global structure, exhibit high discriminative power, they lead to high computational costs. Balancing discriminative power and scalability therefore remains an important open problem.

## 1.2 Spectral Methods for Graphs

Spectral methods (Chung 1996) use eigenvalues and eigenvectors of matrix representations such as the adjacency matrix or the graph Laplacian to perform clustering, graph signal processing, and node centrality estimation. For undirected graphs, spectral clustering (Luxburg 2007; Ng, Jordan, and Weiss 2001) uses the bottom eigenvectors of the matrix representation as vertex embeddings. This view is supported by cut objectives and Cheeger inequalities and provides a geometric description of connectivity and community structure.

In many real-world networks, the direction of edges carries essential information about the flow of influence, dependence, or causality. To exploit directionality, a line of work has adapted spectral techniques to directed graphs. Examples include methods that use left and right singular vectors of the asymmetric adjacency matrix (Rohe, Qin, and Yu 2016). Another example uses a symmetrization of the asymmetric adjacency matrix of a directed graph, defined so that the $(i, j)$-entry represents the count of common out-neighbors and in-neighbors of vertices $i$ and $j$ (Satuluri and Parthasarathy 2011). These developments shift the undirected objective of finding densely connected communities toward the directed objective of grouping vertices with similar sources and destinations.

More recently, as part of efforts to extend graph spectral theory to directed graphs, Hermitian adjacency matrices have been defined by assigning complex weights to

directed edges. The properties of their eigenvalues and eigenvectors have been studied (Guo and Mohar 2017; Liu and X. Li 2015; Mohar 2020). Hermitian-based clustering methods have also been proposed and can outperform naive symmetrizations (Cucuringu et al. 2020; Martin, Rogers, and Zanetti 2024).

Despite these advances, sparse regimes remain challenging. In sparse undirected graphs, adjacency-based methods can fail to recover community structure, and hub nodes can be overemphasized in centrality estimation. One approach to mitigate these issues in undirected graphs is to use the non-backtracking matrix as a representation. However, it remains unclear how matrix representations should be extended to directed graphs and whether such extensions would be effective.

## 1.3 Contributions

Motivated by this background, this dissertation discusses two topics based on the published paper (Sando and Hino 2025) and a paper under review:

- We propose a matrix representation of directed graphs for clustering. We show that the proposed representation is a natural extension of existing representations for undirected graphs. Moreover, our clustering algorithm based on the eigenvectors of the matrix can robustly recover community structure even in sparse directed graphs. We also discuss the relationship between our algorithm and belief propagation.

- For the categorical Wasserstein Weisfeiler-Lehman graph kernel, we make explicit a latent tree structure in the domain information and develop an acceleration that exploits the structure. This allows us to compute the exact, unregularized Wasserstein distance in large datasets with many large graphs, where the previous algorithm was infeasible.

This dissertation is organized as follows. We first introduce the graph-theoretic preliminaries common to both topics in Chapter 2. In Chapter 3, we propose a matrix representation for directed graphs and develop a clustering algorithm as an application. Chapter 4 is devoted to the acceleration of the categorical Wasserstein Weisfeiler-Lehman graph kernel. We show that our insights lead to significant improvements in

computational efficiency over existing methods. Finally, we conclude this dissertation and discuss future directions in Chapter 5.

# 2 Preliminaries

This chapter provides the necessary background for this dissertation. As our work consistently focuses on graph-structured data, we begin by introducing definitions for representing graphs mathematically. These concepts are illustrated with concrete examples to facilitate understanding. Furthermore, in Section 2.7, we review the fundamentals of graph kernels, a prevalent approach for handling graph data within a machine learning framework.

## 2.1 Undirected Graphs

We start with the definition of undirected graphs. As the notation for the edge set differs in the directed case discussed later, we introduce the symbols while making this distinction explicit. Let $V$ be a finite set. We write $\binom{V}{2} \coloneqq \{\{u, v\} \mid u, v \in V,\ u \neq v\}$ for the set of unoriented pairs of distinct elements of $V$. For any distinct $u, v \in V$, we use the symbol $uv \coloneqq \{u, v\} = \{v, u\}$ to denote the unoriented pair. Let $E \subset \binom{V}{2}$ be a subset of unoriented pairs, and define the undirected graph $\mathcal{G} \coloneqq (V, E)$. We refer to elements

of $V$ as vertices and to elements of $E$ as undirected edges. This definition means that undirected graphs have no self-loops, i.e., no undirected edge connects a vertex to itself. The two vertices $u$ and $v$ are said to be *adjacent* if $uv \in E$. When each edge $e \in E$ is assigned a nonnegative weight, we denote it by $w : E \to \mathbb{R}_{\geq 0}$. For a vertex $u \in V$, we denote the set of vertices adjacent to $u$ by $N_u \coloneqq \{v \in V \mid uv \in E\}$. The *degree* of $u$ is the number of its adjacent vertices, denoted $d_u \coloneqq |N_u|$.

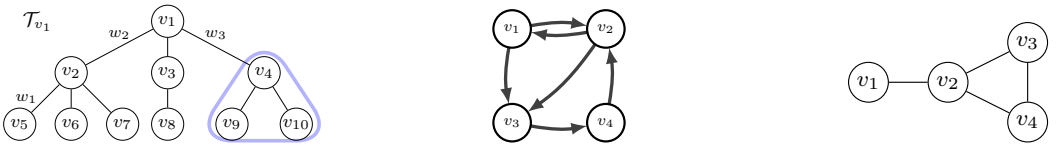
A sequence of vertices $W = (v_1, \ldots, v_k)$ in $\mathcal{G}$ is a walk when $v_i v_{i+1} \in E$ for every $i = 1, \ldots k-1$. The length of $W$, defined as the number of edges it contains, $k-1$, is denoted by $|W|$. An undirected graph is connected if, for every pair of vertices, there exists a walk between them. A walk is called a cycle if its first and last vertices coincide. A walk $(v_1, \ldots, v_k)$ is said to have backtracking if there exists $i \in \{2, \ldots, k-1\}$ such that $v_{i-1} = v_{i+1}$, and a walk without such $i$ is called a non-backtracking (NBT) walk. We can simply say that an NBT walk does not return to the immediately preceding vertex.

Since we handle datasets consisting of undirected graphs with node labels in Chapter 4, we introduce the necessary notation. Let $\Sigma_0$ be a finite categorical label set. We say that an undirected graph is labeled if each vertex of the graph is assigned a label from $\Sigma_0$. We denote a set of graphs by $\boldsymbol{\mathcal{G}} \coloneqq (\mathcal{G}_1 = (V_1, E_1), \ldots, \mathcal{G}_N = (V_N, E_N))$, where $N$ is the number of graphs in $\boldsymbol{\mathcal{G}}$. We also denote the number of vertices in $\mathcal{G}_i$ by $n_i$, the total number of vertices in $\boldsymbol{\mathcal{G}}$ by $n \coloneqq \sum_{i=1}^{N} n_i$, and the average number of vertices in $\boldsymbol{\mathcal{G}}$ per graph by $\bar{n} \coloneqq n/N$. Throughout this dissertation, we assume that all graphs in $\boldsymbol{\mathcal{G}}$ are labeled and share a common label set $\Sigma_0$. In addition, we use $\mathcal{V} \coloneqq \bigcup_{i=1}^{N} V_i$ for all vertices in $\boldsymbol{\mathcal{G}}$, and $l_0 : \mathcal{V} \to \Sigma_0$ for the function that assigns initial labels to vertices.

## 2.2 Trees

When an undirected graph is connected and contains no cycles, it is called a tree. To clearly indicate that a graph is a tree, we denote it by $\mathcal{T} \coloneqq (V(\mathcal{T}), E(\mathcal{T}))$ instead of $\mathcal{G}$, and represent its vertex and edge sets by $V(\mathcal{T})$ and $E(\mathcal{T})$, respectively. Note that, since a tree contains no cycles, there exists exactly one path between any pair of vertices in the tree. We write $\mathcal{P}(u, v)$ as a set of edges that form the unique path between $u, v \in V(\mathcal{T})$. In addition, the number of edges and vertices in a tree satisfy the relation $|E(\mathcal{T})| = |V(\mathcal{T})| - 1$.

Given a tree $\mathcal{T}$ with a weight function $w$, the path metric between $u, v \in V(\mathcal{T})$ is




(a) tree (b) directed graph (c) cycle with a tail

Figure 2.1: Illustrative examples

defined as $d_{(\mathcal{T},w)}(u,v) \coloneqq \sum_{e\in\mathcal{P}(u,v)} w_e$, where $w_e \coloneqq w(e)$, or simply $d_{\mathcal{T}}$ when there is no confusion about $w$. When a vertex of a tree is selected as a reference, that vertex is called the root. A tree rooted at $r \in V(\mathcal{T})$ is denoted by $\mathcal{T}_r$, and the vertex set of the subtree rooted at $u \in V(\mathcal{T})$ is denoted by $\Gamma(u) \coloneqq \{v \in V(\mathcal{T}_r) \mid u \in \mathcal{P}(v,r)\}$. For vertices that constitute an edge $e \in E(\mathcal{T}_r)$, we use $u_e \in V(\mathcal{T}_r)$ for the vertex closer to the root, and $v_e \in V(\mathcal{T}_r)$ for the vertex farther from the root.

Figure 2.1a shows an example of a tree. For instance, the set of edges in the path between vertices $v_4$ and $v_5$ is given by $\mathcal{P}(v_4,v_5) = \{v_2v_5,\ v_1v_2,\ v_1v_4\}$, and the path metric between these two vertices is $d_{\mathcal{T}}(v_4,v_5) = w_1 + w_2 + w_3$. Considering the subtree rooted at $v_4$ highlighted by the blue line, its vertex set is $\Gamma(v_4) = \{v_4,\ v_9,\ v_{10}\}$.

## 2.3 Directed Graphs

Many real-world relationships, such as social networks, economic transactions, or logistics, are inherently directional. To model such systems, we introduce directed graphs, or simply digraphs. We write $V^{(2)} \coloneqq \{(u,v) \mid u,v \in V, u \neq v\}$ for the set of oriented pairs of distinct elements of $V$. In $\binom{V}{2}$, the unoriented pairs $\{u,v\}$ and $\{v,u\}$ are identified, while in $V^{(2)}$ the oriented pairs $(u,v)$ and $(v,u)$ are considered distinct. We write $\overrightarrow{uv}\ (= \overleftarrow{vu}) \coloneqq (u,v)$ for an oriented pair from $u$ to $v$. When we denote an oriented pair $\overrightarrow{uv}$ by $e$, we write $i_e \coloneqq u$ and $t_e \coloneqq v$ to make explicit that $i_e$ and $t_e$ are, respectively, the initial and terminal vertex of $e$. We also use $e^{-1} \coloneqq \overrightarrow{t_e i_e}$ to denote its inverse pair. Let $\vec{E} \subset V^{(2)}$ be a subset of oriented pairs, and define the directed graph $G \coloneqq (V,\vec{E})$. We call elements of $\vec{E}$ *directed edges*. This means that an oriented pair is not necessarily a directed edge. For a directed graph $G = (V,\vec{E})$, we additionally define $\bar{E} \coloneqq \vec{E} \cup \vec{E}^{-1} \subset V^{(2)}$, where $\vec{E}^{-1} \coloneqq \left\{e^{-1} \mid e \in \vec{E}\right\}$. By construction, $\vec{E} \subset \bar{E}$ holds. In the context of a binary relation on $V$, $\bar{E}$ can be regarded as the symmetric closure of

$\vec{E}$. Note that $\left|\bar{E}\right|$ is even. This follows from the facts that $e \in \bar{E}$ implies $e^{-1} \in \bar{E}$ and that self-loops are not included. Let $\left|\bar{E}\right| =: 2m$, and assume that the elements of $\bar{E}$ are indexed so that $e_{m+i} = e_i^{-1}$ for $i = 1, \ldots, m$.

To indicate relationships between two vertices in a directed graph, we introduce the symbols $\to_G$ and $\leftrightarrow_G$. For distinct vertices $u, v \in V$, we write $u \to_G v (= v \leftarrow_G u)$ if $\overrightarrow{uv} \in \vec{E}$ and $\overleftarrow{uv} \notin \vec{E}$, and also write $u \leftrightarrow_G v$ if $\overrightarrow{uv}, \overleftarrow{uv} \in \vec{E}$. For a vertex $u \in V$, we define four types of neighborhoods as follows:

$$\overleftarrow{N_u} \coloneqq \{v \in V \mid u \leftarrow_G v\}, \qquad \overrightarrow{N_u} \coloneqq \{v \in V \mid u \to_G v\},$$
$$\overleftrightarrow{N_u} \coloneqq \{v \in V \mid u \leftrightarrow_G v\}, \qquad N_u \coloneqq \overleftrightarrow{N_u} \cup \overleftarrow{N_u} \cup \overrightarrow{N_u}.$$

We denote the in-degree, out-degree, bi-degree and degree of $u$ by $\overleftarrow{d_u} \coloneqq |\overleftarrow{N_u}|$, $\overrightarrow{d_u} \coloneqq |\overrightarrow{N_u}|$, $\overleftrightarrow{d_u} \coloneqq |\overleftrightarrow{N_u}|$ and $d_u \coloneqq \overleftarrow{d_u} + \overrightarrow{d_u} + \overleftrightarrow{d_u}$, respectively. We also use $D \coloneqq \mathrm{diag}(d_1, \ldots, d_n) \in \mathbb{R}^{n \times n}$.

The definition of a walk in undirected graphs can be applied to directed graphs. Moreover, for Chapter 3, we introduce a new concept for walks in directed graphs. A sequence of vertices $W = (v_1, \ldots, v_k)$ in a digraph $G = (V, \vec{E})$ is called a walk when $\overrightarrow{v_i v_{i+1}} \in \vec{E}$ for all $i = 1, \ldots, k-1$. In addition, we call a sequence a mixed walk if, for every $i = 1, \ldots, k-1$, either $\overrightarrow{v_i v_{i+1}} \in \vec{E}$ or $\overleftarrow{v_i v_{i+1}} \in \vec{E}$ holds. Intuitively, a mixed walk is allowed to traverse directed edges against their direction. The length of $W$, defined as the number of transitions it contains, $k-1$, is denoted by $|W|$. The concept of non-backtracking walks in undirected graphs is similarly defined for walks and mixed walks in directed graphs. The notion of cycles in undirected graphs also applies to directed graphs: a walk with $v_1 = v_k$ is called a cycle, denoted by $C$. Two cycles are considered equivalent if they differ only in the choice of the starting vertex, and the equivalence class of a cycle $C$ is denoted by $[C]$. For example, if there exists a cycle represented by the vertex sequence $(v_1, v_2, v_3, v_1)$, then the vertex sequences $(v_2, v_3, v_1, v_2)$ and $(v_3, v_1, v_2, v_3)$ belong to the same equivalence class. The $r$ times repetition of a cycle $C$ is denoted as $C^r$. If a cycle $C$ cannot be expressed as a repetition of a shorter cycle, the cycle is called primitive. We note that even if a cycle $C$ is NBT, $C^2$ is not necessarily NBT. A cycle $C$ is said to have a tail if $C$ is NBT but $C^2$ is not NBT.

We illustrate the terms defined above for directed graphs using the example in Figures 2.1b and 2.1c. Let the directed graph in Figure 2.1b be denoted by

$G = (V, \vec{E})$, where $V = \{v_1,\ v_2,\ v_3,\ v_4\}$ and $\vec{E} = \{\overrightarrow{v_1v_2},\ \overrightarrow{v_2v_1},\ \overrightarrow{v_1v_3},\ \overrightarrow{v_2v_3},\ \overrightarrow{v_3v_4},\ \overrightarrow{v_4v_2}\}$. With respect to vertices $v_1$ and $v_3$, since $\overrightarrow{v_1v_3} \in \vec{E}$ and $\overrightarrow{v_3v_1} \notin \vec{E}$, we write $v_1 \to_G v_3$. Also, for vertices $v_1$ and $v_2$, we write $v_1 \leftrightarrow_G v_2$. The symmetric closure $\bar{E}$ has $m = 5$, and although the indexing of its elements is arbitrary, we can set $\bar{E} = \{\overrightarrow{v_1v_2},\ \overrightarrow{v_1v_3},\ \overrightarrow{v_2v_3},\ \overrightarrow{v_3v_4},\ \overrightarrow{v_4v_2},\ \overleftarrow{v_1v_2},\ \overleftarrow{v_1v_3},\ \overleftarrow{v_2v_3},\ \overleftarrow{v_3v_4},\ \overleftarrow{v_4v_2}\}$. Considering the neighborhoods of vertex $v_2$ as an example, we have $\overleftarrow{N}_{v_2} = \{v_4\}$, $\overrightarrow{N}_{v_2} = \{v_3\}$, $\overleftrightarrow{N}_{v_2} = \{v_1\}$, $N_{v_2} = \{v_1,\ v_3,\ v_4\}$. This leads to the degrees $\overleftarrow{d}_{v_2} = 1$, $\overrightarrow{d}_{v_2} = 1$, $\overleftrightarrow{d}_{v_2} = 1$, $d_{v_2} = 3$. Moreover, while a sequence of vertices $S_1 = (v_1,\ v_2,\ v_3,\ v_4)$ is a walk in $G$, a sequence $S_2 = (v_1,\ v_2,\ v_4)$ is not considered a walk; however, $S_2$ is a mixed walk. Figure 2.1c illustrates a cycle with a tail. Since this property holds for both walks and mixed walks, the directions of the edges are omitted. Figure 2.1c shows that the sequence $C = (v_1, v_2, v_3, v_4, v_2, v_1)$ is an NBT cycle; however, $C^2 = (v_1, v_2, v_3, v_4, v_2, v_1, v_2, v_3, v_4, v_2, v_1)$ exhibits backtracking at the junction between the end of the first round and the beginning of the second round.

## 2.4 Matrix Representations of Graphs

To handle graph data mathematically, it is common to use matrix representations. The most fundamental among them is the adjacency matrix, denoted by $A$, defined based on the adjacency relationships between pairs of vertices. Let $n$ be the number of vertices in a graph, the entry $A_{uv}$ of two vertices $u, v \in V$ for undirected and directed graphs is defined as follows:

$$A_{uv} \coloneqq \begin{cases} 1 & uv \in E, \\ 0 & \text{otherwise}, \end{cases} \qquad A_{uv} \coloneqq \begin{cases} 1 & \overrightarrow{uv} \in \vec{E}, \\ 0 & \text{otherwise}. \end{cases}$$

In particular, for undirected graphs, representative matrices derived from the adjacency matrix include the *graph Laplacian* $L \coloneqq D - A$ and the *normalized graph Laplacian* $\mathcal{L} \coloneqq D^{-1/2}LD^{-1/2}$.

While the adjacency matrix is defined based on relationships between pairs of vertices, it is also possible to define a matrix based on relationships between pairs of edges. A representative example is the non-backtracking (NBT) matrix (Hashimoto 1989). To consider the NBT matrix of an undirected graph $\mathcal{G} = (V, E)$, we first construct an induced directed graph $G = (V, \vec{E})$ by replacing each undirected edge with two

directed edges in opposite directions, where $\vec{E} = \left\{\overrightarrow{uv}, \overleftarrow{uv} \mid uv \in E\right\}$. We note that, for the induced set of directed edges, $\bar{E} = \vec{E}$ holds. Let $m \coloneqq |E|$ and $e, f \in \vec{E}$, the NBT matrix $B$ of the undirected graph is a $2m \times 2m$ matrix, with each index corresponding to a directed edge. The entry $B_{ef}$ of the NBT matrix $B$ is defined as follows:

$$\begin{aligned} B_{ef} &\coloneqq \begin{cases} 1 & t_e = i_f \text{ and } i_e \neq t_f, \\ 0 & \text{otherwise.} \end{cases} \\ &= \delta_{t_e i_f}(1 - \delta_{i_e t_f}), \end{aligned} \tag{2.1}$$

where $\delta$ is the Kronecker delta. This means that the entry is 1 when the two directed edges constitute a non-backtracking walk. The NBT matrix can also be defined for directed graphs, but in that case, we use $\bar{E}$ instead of $\vec{E}$ and apply the same definition.

## 2.5 Graph Cut and Spectral Clustering

In an undirected graph, the optimization problem of partitioning the vertex set into several subsets while minimizing the number of edges that span different subsets is known as the *graph cut problem*, and is also recognized as a method for graph clustering. Let $\mathcal{G} = (V, E)$ be an undirected graph, and $S, U \subset V$ be subsets of the vertex set. The number of edges that span these two subsets is defined as follows:

$$\operatorname{cut}(S, U) \coloneqq |\{uv \in E \mid u \in S, v \in U\}| .$$

For example, when considering partitioning the vertex set into two clusters, simply minimizing $\operatorname{cut}(S, \bar{S})$ can lead to trivial solutions, such as $S$ consisting of only one vertex. Therefore, some form of regularization is applied. Here, we focus on one such method called *Normalized Cut (NCut)*. The NCut for a partition of the vertex set $V$ into $S_1, \ldots, S_K$ is defined as follows:

$$\operatorname{NCut}(S_1, \ldots, S_K) \coloneqq \sum_{k=1}^{K} \frac{\operatorname{cut}(S_k, \bar{S_k})}{\operatorname{vol}(S_k)}, \tag{2.2}$$

where $\bar{S_k}$ is the complement of $S_k$ in $V$, and $\mathrm{vol}(S) \coloneqq \sum_{u \in S} d_u$. Intuitively, the regularization in NCut aims to prevent the total degree within the two subsets from being imbalanced. The goal of the graph cut problem is to find a partition of the vertex set that minimizes NCut.

$$\min_{S_1,\ldots,S_K \subset V} \mathrm{NCut}(S_1,\ldots,S_K) \qquad \text{where} \quad \begin{cases} S_i \cap S_j = \phi & i \neq j, \\ \bigcup_{k=1}^{K} S_k = V. \end{cases} \tag{2.3}$$

The problem (2.3) can be equivalently transformed into the following form.

**Lemma 2.1.** *The optimization problem* (2.3) *is equivalent to the following optimization problem:*

$$\min_{S_1,\ldots,S_K \subset V} \mathrm{Tr}\left(H^\top L H\right) \qquad \textit{where} \begin{cases} H \in \mathbb{R}^{|V| \times K}, \\ H_{ik} = \begin{cases} \frac{1}{\sqrt{\mathrm{vol}(S_k)}} & v_i \in S_k, \\ 0 & \textit{otherwise.} \end{cases} \\ H^\top D H = I, \\ S_i \cap S_j = \phi \quad i \neq j, \\ \bigcup_{k=1}^{K} S_k = V. \end{cases} \tag{2.4}$$

In the problem (2.4), the matrix $H$ is an indicator matrix that shows which subset each vertex belongs to, taking $K+1$ discrete values. By relaxing $H$ to allow continuous values, we obtain *spectral clustering*, a method for obtaining cluster partitions.

### 2.5.1 Spectral Clustering

Spectral clustering (Luxburg 2007; Ng, Jordan, and Weiss 2001) is an algorithm that performs vertex clustering by solving the eigenvalue problem of the graph Laplacian, and deals with the optimization problem obtained by relaxing the constraint that $H$ takes discrete values.

$$\min_{H \in \mathbb{R}^{|V| \times K},\ H^\top D H = I} \mathrm{Tr}\left(H^\top L H\right).$$

**Algorithm 1:** Spectral Clustering

**Input:** $\mathcal{G} = (V, E)$: undirected graph; $K \in \mathbb{Z}$: number of clusters
**Output:** $z \in \{1, \ldots, K\}^{|V|}$: cluster assignment for each vertex

1 Compute the normalized graph Laplacian $\mathcal{L}$
2 Perform eigenvalue decomposition of $\mathcal{L}$ and obtain the eigenvectors corresponding to the $K$ smallest eigenvalues to form $X \in \mathbb{R}^{|V| \times K}$
3 Normalize each row of $X$ to have a norm of 1
4 Apply the K-means algorithm to the row vectors of $X$
5 Let the cluster assignment for vertex $v_i$ be $z_i$, corresponding to the cluster assignment of the $i$-th row of $X$
6 **return** $z = (z_1, \ldots, z_{|V|})$

For a derivation via continuous relaxation, see, for example, Oyelade et al. (2019). This corresponds to finding the eigenvectors associated with the $K$ smallest eigenvalues of the normalized graph Laplacian. As described in Algorithm 1, spectral clustering treats the obtained eigenvectors as embeddings of each vertex into a $K$-dimensional Euclidean space and performs clustering using the K-means algorithm.

### 2.5.2 Eigenvalue and Cheeger Inequality

The most ideal situation in the graph cut problem is when an undirected graph consists of $K$ connected components. In the graph Laplacian, the following relationship holds between the number of connected components and the eigenvalues.

**Theorem 2.2** (Chung (1996))**.** *For an undirected graph $\mathcal{G}$, the following two statements are equivalent:*

- *The graph has $K$ connected components.*
- *The multiplicity of the eigenvalue 0 in the normalized graph Laplacian is $K$.*

In practice, it is rare for an undirected graph to consist of $K$ connected components, and it is reasonable to expect that some edges exist between clusters that are intended to be separated. In particular, for the graph cut problem in two partitions, the *Cheeger Inequality* is a classical result that relates the degree of separation between the two clusters and the eigenvalue of the normalized graph Laplacian.

The Cheeger Inequality focuses on the *conductance*, defined as follows, as a measure of the quality of the partition:

$$\phi(S) \coloneqq \frac{\mathrm{cut}(S, \bar{S})}{\min\left\{\mathrm{vol}(S), \mathrm{vol}(\bar{S})\right\}}, \qquad \phi(\mathcal{G}) \coloneqq \min_{S \subset V} \phi(S).$$

Similar to NCut, conductance includes regularization to prevent the total degree of the subsets from being imbalanced. However, unlike NCut, it focuses on the larger value among the two partitions, and $\phi(\mathcal{G})$ represents the minimum conductance among all possible partitions. The following relationship holds between conductance and the second eigenvalue of the normalized graph Laplacian.

**Theorem 2.3** (Cheeger Inequality (Chung 1996))**.** *Let $\lambda_2$ be the second smallest eigenvalue of the normalized graph Laplacian of an undirected graph $\mathcal{G}$. Then, the following inequalities hold:*

$$\frac{\lambda_2}{2} \leq \phi(\mathcal{G}) \leq \sqrt{2\lambda_2}.$$

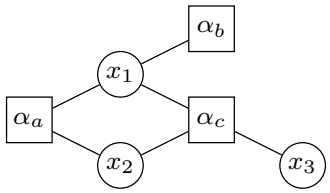
Although solving the actual two-partition graph cut problem is NP-hard (Delling et al. 2015), the Cheeger Inequality allows us to evaluate how good the best partition is by focusing on the second eigenvalue of the normalized graph Laplacian.

## 2.6 Belief Propagation

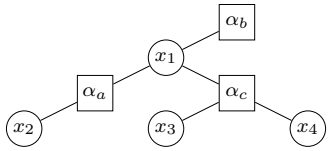
Belief propagation is an algorithm for computing marginal distributions in probabilistic models where the joint distribution can be expressed in Equation (2.5), commonly used in graphical models.

$$P(\{x_1, \ldots, x_n\}) \propto \prod_{\lambda=1}^{m} \psi_\lambda(\boldsymbol{x}_{\partial\lambda}), \tag{2.5}$$

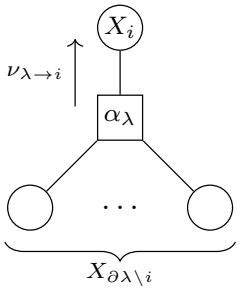
where $\alpha_1, \ldots, \alpha_m$ represents factors that indicate the relationship among random variables, and we use $i, j$ as index symbols for random variables and $\lambda$ for factors. The notation $\partial\lambda$ denotes the index set of random variables associated with factor $\alpha_\lambda$, and the notation $\boldsymbol{x}_{\partial\lambda}$ denotes the set of random variables corresponding to the index set



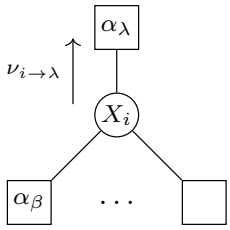

(a) Example of a factor graph




(b) Example of a factor tree




(c) from factor to variable




(d) from variable to factor

Figure 2.2: Conceptual diagrams of messages

$\partial\lambda$. Similarly, we denote the index set of factors associated with random variable $x_i$ by $\partial i$. The function $\psi_\lambda(\boldsymbol{x}_{\partial\lambda})$ is a factor function that describes the relationship of $\alpha_\lambda$. One way to visualize the relationships between factors and random variables is through factor graphs, which are bipartite graphs. Figure 2.2a shows an example of a factor graph. Factors $\alpha_a$ and $\alpha_c$ represent relationships involving random variables $x_1, x_2$ and $x_1, x_2, x_3$, respectively. Additionally, a factor $\alpha_b$ involving only $x_1$ can also be represented, which is often used to express prior distributions of individual variables.

In this section, we first explain the situation where marginal distributions can be computed exactly and efficiently, and then discuss the approximate computation of marginal distributions in more general situations. In this dissertation, we assume that the random variables handled by belief propagation are discrete random variables.

### 2.6.1 Exact Inference on Trees

When the factor graph is a tree, belief propagation can compute exact marginal distributions efficiently. Specifically, marginal distributions are computed by propagating values called *messages* along the edges. Since the factor graph in Figure 2.2a is not a tree, we use the example in Figure 2.2b here.

We explain the belief propagation based on two types of messages: the message

**Algorithm 2:** Belief Propagation on Factor Trees

**for** *compute messages starting from those that can be computed* **do**

$$\nu_{\lambda\to i}(x_i) \propto \sum_{\boldsymbol{x}_{\partial\lambda\setminus i}} \psi_\lambda(x_i, \boldsymbol{x}_{\partial\lambda\setminus i}) \prod_{j\in\partial\lambda\setminus i} \nu_{j\to\lambda}(x_j), \text{ where } \prod \nu_{j\to\lambda}(x_j) = 1 \text{ when } \partial\lambda \setminus i = \emptyset$$

$$\nu_{i\to\lambda}(x_i) \propto \prod_{\beta\in\partial i\setminus\lambda} \nu_{\beta\to i}(x_i), \text{ where } \prod \nu_{\beta\to i}(x_i) = 1 \text{ when } \partial i \setminus \lambda = \emptyset$$

**end**

**for** $i \in \{1, \ldots, n\}$ **do**

$$\hat{P}(x_i) \propto \prod_{\lambda\in\partial i} \nu_{\lambda\to i}(x_i)$$

**end**

$\nu_{\lambda\to i}$ from factor $\alpha_\lambda$ to random variable $x_i$ (Figure 2.2c), and the message $\nu_{i\to\lambda}$ from random variable $x_i$ to factor $\alpha_\lambda$ (Figure 2.2d). By using these messages, the belief propagation computes marginal distributions according to Algorithm 2. Algorithm 2 propagates messages starting from the leaves. Let us consider the message from $\alpha_b$ to $X_1$ in Figure 2.2b, $\partial b \setminus 1 = \emptyset$ holds. This means that $\nu_{b\to 1}(x_1) \propto \psi_{\alpha_b}(x_1)$ can be computed. Similarly, the messages from $X_2$ to $\alpha_a$, from $X_3$ to $\alpha_c$, and from $X_4$ to $\alpha_c$ can be computed, since $\partial 2 \setminus a = \emptyset$, $\partial 3 \setminus c = \emptyset$, $\partial 4 \setminus c = \emptyset$. By computing messages in this manner, starting from those that can be computed, all messages can be calculated.

When the factor graph is a tree, it can be shown by induction that $p(x_i)$ computed by Algorithm 2 is equal to the marginal distribution.

**Proposition 2.1.** *For a random variable $X_i$ in a graphical model whose factor graph is a tree, the marginal distribution of $X_i$ can be expressed using the messages computed by Algorithm 2 as follows:*

$$P(x_i) \propto \prod_{\lambda\in\partial i} \nu_{\lambda\to i}(x_i).$$

The proof can be done by treating the entire factor graph as a tree rooted at $X_i$ and using induction. Similarly, it can be proven for other random variables contained in the factor graph.

**Algorithm 3:** Belief Propagation on General Graphs

**Input:** $N$: the maximum number of iterations

// Initialize messages

**for** $\alpha \in F,\ i \in \partial\alpha$ **do**

  $\nu_{\alpha\to i}(x_i) \propto 1$

  $\nu_{i\to\alpha}(x_i) \propto 1$

**end**

**while** $N$ *times* **do**

  $$\nu_{\alpha\to i}(x_i) \propto \sum_{\boldsymbol{x}_{\partial\alpha\setminus i}\in X_{\partial\alpha\setminus i}} \psi_\alpha(x_i, \boldsymbol{x}_{\partial\alpha\setminus i}) \prod_{j\in\partial\alpha\setminus i} \nu_{j\to\alpha}(x_j)$$

  $$\nu_{i\to\alpha}(x_i) \propto \prod_{\lambda\in\partial i\setminus\alpha} \nu_{\alpha_\lambda\to i}(x_i)$$

**end**

**for** $i \in \{1,\dots,n\}$ **do**

  $$\hat{P}(x_i) \propto \prod_{\alpha\in\partial i} \nu_{\alpha\to i}(x_i)$$

**end**

### 2.6.2 Approximate Inference on General Graphs

In factor graphs with tree structures, messages can be computed deterministically in order from the leaves. However, in general graphs that contain cycles, the message computation formulas involve circular references. Therefore, in general factor graphs, belief propagation performs approximate computation of marginal distributions by iteratively updating messages until convergence. It is known that this algorithm may not converge in general, but it is effective in many applications. In Chapter 3, discussions based on this algorithm are presented. Belief propagation in general factor graphs is represented by Algorithm 3.

## 2.7 Graph Kernels

Many machine learning algorithms, such as linear regression, support vector machines, and principal component analysis, can be formulated in a way that their computations rely solely on the inner products between data points, rather than directly handling individual features of the data. The kernel method (Aronszajn 1950; Berlinet and

Thomas-Agnan 2011) is a methodology that treats nonlinear relationships in data as linear learning in an implicit feature space, by using a positive definite kernel function that provides the inner product in the feature space, and by expressing algorithms purely in terms of inner products between data points.

In the kernel method, a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{C}$ is a function that provides the inner product between points mapped to an implicit high-dimensional feature space, and can be interpreted as a measure of similarity between data points. Not just any function can serve as a kernel function. For a function $k$ to be a valid kernel, there must exist a corresponding feature map to a high-dimensional space such that $k$ represents the inner product in the space. The property that a function $k$ must satisfy for this to hold is positive definiteness.

**Definition 2.4** (Positive Definiteness (Berg, Christensen, and Ressel 1984))**.** *Let $\mathcal{X}$ be a non-empty set. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{C}$ is called a positive definite kernel if and only if*

$$\sum_{i,j=1}^{n} c_i \overline{c_j} k(x_i, x_j) \geq 0$$

*for all $n \in \mathbb{N}$, $\{x_1, \ldots, x_n\} \subset \mathcal{X}$, and $\{c_1, \ldots, c_n\} \subset \mathbb{C}$.*

A graph kernel refers to the application of kernel methods to graph data, or more specifically, the kernel function itself, which measures the similarity between two graphs. In general, graph data may vary in the number of vertices and edges, and the ordering of vertices does not have intrinsic meaning. Representing such data as fixed-length vectors may lead to the loss of essential structural information. To address this, a classical approach is based on the framework of R-convolution kernels, which defines the similarity between entire graphs by measuring the similarity of their constituent substructures. In the following, we introduce a shortest-path kernel as a representative example of graph kernels based on R-convolution.

### 2.7.1 Shortest-Path Kernel

One of the representative graph kernels based on R-convolution is the shortest-path kernel (Borgwardt and Kriegel 2005). Let $\mathcal{G} = (V, E)$ be an undirected graph in which each vertex $v$ is assigned a label $l_0(v)$. From $\mathcal{G}$, we construct a shortest-path graph

$S = (V, E_s)$ with weights $w\colon E_s \to \mathbb{R}$. The vertex set is identical to that of $\mathcal{G}$, while the edge set $E_s$ consists of all pairs of vertices that are mutually reachable in $\mathcal{G}$. Each edge is weighted by the corresponding shortest path length in $\mathcal{G}$. This weight is well-defined even when multiple shortest paths exist, since all such paths have the same length.

Given two undirected graphs $\mathcal{G}_1 = (V_1, E_1)$ and $\mathcal{G}_2 = (V_2, E_2)$, let their respective shortest-path graphs be $S_1 = (V_1, E_{s_1})$ and $S_2 = (V_2, E_{s_2})$. The shortest-path kernel $k_{\text{sp}}$ is defined as follows:

$$k_{\text{sp}}(\mathcal{G}_1, \mathcal{G}_2) \coloneqq \sum_{\substack{e_1=(u,v)\in E_{s_1} \\ e_2=(x,y)\in E_{s_2}}} k_{\text{walk}}\left(e_1, e_2\right)$$

$$\begin{aligned} \text{where} \quad k_{\text{walk}}\left(e_1, e_2\right) \coloneqq\ & k_{\text{node}}\left(l_0(u), l_0(x)\right) \times k_{\text{node}}\left(l_0(v), l_0(y)\right) \times k_{\text{edge}}\left(w_{e_1}, w_{e_2}\right) \\ & + k_{\text{node}}\left(l_0(u), l_0(y)\right) \times k_{\text{node}}\left(l_0(v), l_0(x)\right) \times k_{\text{edge}}\left(w_{e_1}, w_{e_2}\right), \end{aligned}$$

where $k_{\text{node}}$ and $k_{\text{edge}}$ are kernel functions that measure the similarity of vertex labels and edges, respectively. Typically, Dirac kernels are often used, which return 1 when the two arguments are equal.

## 2.8 Optimal Transport and Sinkhorn Algorithm

Optimal transport theory (Villani 2009) was originally formulated as a mathematical framework for finding the most efficient plan to transport resources from one location to another. In machine learning, it has been widely applied as a method for measuring the distance between two probability distributions. Although there are various formulations of optimal transport problems, we focus here on the specific setting used in this dissertation.

Suppose that $\mathcal{X}$ is a non-empty set, there are resources with masses $\boldsymbol{a} \coloneqq (a_1, \ldots, a_m)$ at $m$ locations $(x_1, \ldots, x_m) \subset \mathcal{X}$, and a demand of masses $\boldsymbol{b} \coloneqq (b_1, \ldots, b_n)$ at $n$ target locations $(y_1, \ldots, y_n) \subset \mathcal{X}$. The cost per unit mass for transporting from location $x_i$ to $y_j$ is given by $C(x_i, y_j) \geq 0$. We aim to solve the optimization problem of finding a transportation plan that minimizes the total transportation cost. In this plan, $P_{ij}$ denotes the mass transported from $x_i$ to $y_j$. We assume that the total mass of resources is equal to the total demand, and is normalized to one. This optimization problem is

formulated as the following linear programming problem:

$$\min_{P \in \mathbb{R}^{m \times n}} \sum_{i=1}^{m} \sum_{j=1}^{n} C(x_i, y_j) P_{ij} \tag{2.6}$$

$$\text{s.t.} \begin{cases} P_{ij} \geq 0, \quad i = 1, \ldots, m, \ j = 1, \ldots, n. \\ \sum_{j=1}^{n} P_{ij} = a_i, \quad i = 1, \ldots, m. \\ \sum_{i=1}^{m} P_{ij} = b_j, \quad j = 1, \ldots, n. \end{cases}$$

This discrete formulation can be extended to a continuous setting using probability measures. The mass distribution at the source locations, corresponding to the vector $\boldsymbol{a}$, is represented by a probability measure $\mu$ on $\mathcal{X}$. Likewise, the demand distribution at the target locations, corresponding to the vector $\boldsymbol{b}$, is denoted by another probability measure $\nu$ on $\mathcal{X}$. The transportation plan $P$ is then replaced by a joint probability measure $\pi$ on the product space $\mathcal{X} \times \mathcal{X}$, whose marginals are $\mu$ and $\nu$, called a coupling. Consequently, the optimization problem in Equation (2.6) can be rewritten as follows:

$$\inf_{\pi \in \Pi(\mu, \nu)} \int C(x, y) d\pi(x, y),$$

where $\Pi(\mu, \nu)$ denotes the set of all couplings of $\mu$ and $\nu$, and $C : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ denotes a cost function.

In the problem described above, when the transportation cost between nearby locations is low, the total transportation cost decreases as the set of supply locations and demand locations are more similar, and as the supply and demand amounts at each location are closer. In other words, the total cost is small when the distributions $\mu$ and $\nu$ are similar. This cost seems to represent a metric between $\mu$ and $\nu$ intuitively. However, whether this cost indeed satisfies the axioms of a metric depends on the design of the cost function $C$. The Wasserstein distance (Peyré and Cuturi 2020; Villani 2009) is a metric specifically designed so that the optimal transport cost becomes a valid distance metric.

**Definition 2.5** (Wasserstein Distance). *Let $(\Omega, d)$ be a metric space and $\mu, \nu$ be probability*

*measures supported on* $\Omega$. *The* $L^p$*-Wasserstein distance for* $p \geq 1$ *is defined as follows:*

$$W^p(\mu, \nu) \coloneqq \left( \inf_{\pi \in \Pi(\mu,\nu)} \int d(x,y)^p d\pi(x,y) \right)^{1/p}.$$

For the proof that this satisfies the axioms of a metric, we refer the reader to Villani (2009, Section 6). In this dissertation, we focus on the $L^1$-Wasserstein distance, which we denote by $W_d$, i.e., optimal transport with ground cost $d$.

### 2.8.1 Sinkhorn Algorithm

In practice, when computing the Wasserstein distance, we deal with the discrete formulation in Equation (2.6) derived from a finite number of data. However, solving this as a linear programming problem generally requires at least cubic time in the number of vertices in the worst case. Although recent work has proposed an algorithm with almost-linear time in the number of edges from a theoretical perspective (L. Chen et al. 2023), applying it to large-scale problems remains challenging due to the complexity of its implementation. As an algorithm better suited for modern computing environments, an approximate optimal transport problem with entropic regularization has been introduced (Cuturi 2013), which can be solved efficiently using the Sinkhorn algorithm.

In Cuturi (2013), the entropic regularized optimal transport problem can be written as follows:

$$\begin{aligned}
&\min_{P \in \mathbb{R}^{m \times n}} \sum_{i=1}^{m} \sum_{j=1}^{n} C(x_i, y_j) P_{ij} + \epsilon \mathrm{KL}(P || \boldsymbol{a}\boldsymbol{b}^\top) \\
&\text{s.t.} \begin{cases} P_{ij} \geq 0, \quad i = 1, \ldots, m, \ j = 1, \ldots, n. \\ \sum_{j=1}^{n} P_{ij} = a_i, \quad i = 1, \ldots, m. \\ \sum_{i=1}^{m} P_{ij} = b_j, \quad j = 1, \ldots, n. \end{cases} \\
&\text{where} \quad \mathrm{KL}(P || \boldsymbol{a}\boldsymbol{b}^\top) \coloneqq \sum_{i=1}^{m} \sum_{j=1}^{n} P_{ij} \log \frac{P_{ij}}{a_i b_j},
\end{aligned} \tag{2.7}$$

**Algorithm 4:** Sinkhorn algorithm

**Input:** $\boldsymbol{a} \in \mathbb{R}^m_{\geq 0}$, $\boldsymbol{b} \in \mathbb{R}^n_{\geq 0}$ such that $\|\boldsymbol{a}\| = \|\boldsymbol{b}\| = 1$; $C \in \mathbb{R}^{m \times n}_{\geq 0}$; $\epsilon > 0$
**Output:** optimal transport plan $P^* \in \mathbb{R}^{m \times n}_{\geq 0}$

$K \leftarrow \exp(-C/\epsilon)$
$\boldsymbol{u} \leftarrow \mathbf{1}_m$
$\boldsymbol{v} \leftarrow \frac{\mathbf{1}_n}{\|K\|_1}$
**while** $\boldsymbol{u}$ *or* $\boldsymbol{v}$ *change* **do**
  $\boldsymbol{u} \leftarrow \frac{\boldsymbol{a}}{K\boldsymbol{v}}$
  $\boldsymbol{v} \leftarrow \frac{\boldsymbol{b}}{K^\top \boldsymbol{u}}$
**end**
**return** $\operatorname{diag}(\boldsymbol{u})\ K\ \operatorname{diag}(\boldsymbol{v})$

where $\mathrm{KL}(P||\boldsymbol{a}\boldsymbol{b}^\top)$ is known as the Kullback-Leibler divergence, which measures the discrepancy between two probability distributions. Since the transportation plan $P$ can be viewed as a joint distribution with marginals $\boldsymbol{a}$ and $\boldsymbol{b}$, we note that $\boldsymbol{a}\boldsymbol{b}^\top$ is one feasible solution satisfying this condition. Therefore, the optimization problem (2.7) can be interpreted as a regularized optimal transport problem, where the solution is encouraged to be closer to $\boldsymbol{a}\boldsymbol{b}^\top$ as the regularization parameter $\epsilon$ increases.

The objective function of the problem (2.7) is strongly convex, and the feasible set is also convex. Therefore, it is guaranteed that a unique optimal solution exists. The iterative algorithm introduced by Cuturi (2013) for machine learning is the Sinkhorn algorithm, as shown in algorithm 4.

## 2.9 Proof of Proposition 2.1

To simplify the proof, we consider a tree structure rooted at $X_i$. Figure 2.3 illustrates an example of such a tree structure.

Since no two random variables are adjacent, and no two factors are adjacent, the structure alternates between layers of random variables and layers of factors at each depth. We also assume that each random variable has a factor corresponding to its prior distribution, which is associated only with itself. If there are random variables without such factors, we can add a factor with a factor function $\psi_\lambda(x) \propto 1$ without affecting the model, thus satisfying the assumption. Furthermore, for each random

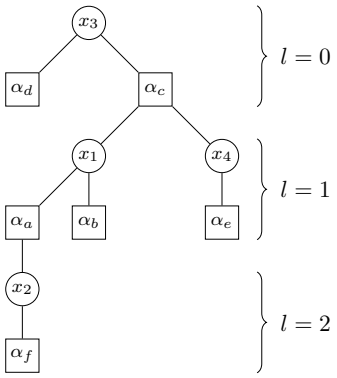



Figure 2.3: Example of treating $X_3$ as the root in Figure 2.2b. Factors $\alpha_d, \alpha_e, \alpha_f$ represent additional dummy factors that have been added.

variable, if there are multiple factors associated only with itself, we can merge them into a single factor without affecting the model. Therefore, we assume that each random variable has exactly one factor associated only with itself.

By rearranging the nodes in this way, we can describe a tree structure as shown in Figure 2.3. We treat the random variables at a certain depth and their adjacent child factors as a *layer*, such as $l = 0, 1, 2$ in Figure 2.3. Let $L$ be the number of layers in the factor tree, and if we denote the depth of the root of the factor tree as 0, the maximum depth of the tree can be expressed as $2L - 1$. We will prove the proposition by induction on $L$. To prepare for this, we denote the set of random variables located at layer $l$ by $\mathcal{X}_l$, and its cardinality by $n_l$. We re-index the random variables located at layer $l$, denoting the index of the $i$-th random variable as $v(l, i)$. Thus, we have $\mathcal{X}_l = \{X_{v(l,1)}, \ldots, X_{v(l,n_l)}\}$. Similarly, we denote the set of factors located at layer $l$ by $\mathcal{Y}_l$, and its cardinality by $m_l$. We re-index the factors located at layer $l$, denoting the index of the $\lambda$-th factor as $f(l, \lambda)$. We denote the set of indices of child factors associated with the random variable $X_{v(l,i)}$ by $Cv(l, i)$, and the set of indices of child random variables associated with the factor $\alpha_{f(l,\lambda)}$ by $Cf(l, \lambda)$. Additionally, we denote the set of indices of child factors associated with a random variable $X$ by $Cv(X)$, and the set of indices of child random variables associated with a factor $\beta$ by $Cf(\beta)$.

**Proof of Proposition 2.1.** We prove the proposition by induction on $L$, based on the preparations described above.

**Proof for $L = 1$**

Since there is only the layer related to the root, the only random variable in the model

is $X_{v(1,1)}$. Thus, from Equation (2.5), we have $P(x_{v(1,1)}) \propto \psi_{f(1,1)}(x_{v(1,1)})$.

On the other hand, regarding the message, since $\partial f(1,1) \setminus v(1,1) = \emptyset$, we have $\nu_{f(1,1)\to v(1,1)}(x_{v(1,1)}) \propto \psi_{f(1,1)}(x_{v(1,1)})$. Moreover, since $\partial v(1,1) = \{f(1,1)\}$, we have $\hat{P}(x_{v(1,1)}) \propto \nu_{f(1,1)\to v(1,1)}(x_{v(1,1)})$. Therefore, the proposition holds for $L = 1$.

**Proof that if the proposition holds for $L = L'$, then it also holds for $L = L' + 1$**
From the joint distribution in Equation (2.5), the marginal distribution of $x_{v(1,1)}$ can be expressed as follows:

$$\begin{aligned} P(x_{v(1,1)}) &= \sum_{x_{v(2,1)}} \cdots \sum_{x_{v(L'+1,n_{L'+1})}} P(\{x_{v(1,1)}, x_{v(2,1)}, \ldots, x_{v(L'+1,n_{L'+1})}\}) \\ &= \sum_{x_{v(2,1)}} \cdots \sum_{x_{v(L'+1,n_{L'+1})}} \prod_{\lambda=1}^{m} \psi_\lambda(\boldsymbol{x}_{\partial\lambda}). \end{aligned}$$

By separating the term $\prod_{\lambda=1}^{m} \psi_\lambda(\boldsymbol{x}_{\partial\lambda})$ into contributions from up to layer $L'-1$, layer $L'$, and layer $L'+1$, we have

$$P(x_{v(1,1)}) = \sum_{x_{v(2,1)}} \cdots \sum_{x_{v(L'+1,n_{L'+1})}} \left[ \begin{aligned} &\left\{ \prod_{l=1}^{L'-1} \prod_{i=1}^{n_l} \prod_{\lambda_1 \in Cv(l,i)} \psi_{\lambda_1}(\boldsymbol{x}_{\partial\lambda_1}) \right\} \times \left\{ \prod_{j=1}^{n_{L'}} \prod_{\lambda_2 \in Cv(L',j)} \psi_{\lambda_2}(\boldsymbol{x}_{\partial\lambda_2}) \right\} \\ &\times \left\{ \prod_{k=1}^{n_{L'+1}} \prod_{\lambda_3 \in Cv(L'+1,k)} \psi_{\lambda_3}(\boldsymbol{x}_{\partial\lambda_3}) \right\} \end{aligned} \right].$$

Considering the third term inside the square brackets, since the maximum number of layers is $L'+1$ and there is only one factor associated with a single random variable, we have $Cv(L'+1,k) = \{f(L'+1,k)\}$ and $\partial\lambda_3 = \{v(L'+1,k)\}$. This is because if $Cv(L'+1,k)$ contained factor indices other than $f(L'+1,k)$, then that factor would be associated with other random variables, which would contradict the assumption by creating loops or exceeding the number of layers $L'+1$. Similarly, if $\partial\lambda_3$ contained random variable indices other than $v(L'+1,k)$, it would also contradict the assumption by creating loops or exceeding the number of layers $L'+1$. Therefore, $P(x_{v(1,1)})$ can be

rewritten as follows:

$$P(\boldsymbol{x}_{v(1,1)}) = \sum_{\boldsymbol{x}_{v(2,1)}} \cdots \sum_{\boldsymbol{x}_{v(L'+1,n_{L'+1})}} \left[ \begin{array}{l} \left\{ \prod_{l=1}^{L'-1} \prod_{i=1}^{n_l} \prod_{\lambda_1 \in Cv(l,i)} \psi_{\lambda_1}(\boldsymbol{x}_{\partial\lambda_1}) \right\} \times \left\{ \prod_{j=1}^{n_{L'}} \prod_{\lambda_2 \in Cv(L',j)} \psi_{\lambda_2}(\boldsymbol{x}_{\partial\lambda_2}) \right\} \\ \times \left\{ \prod_{k=1}^{n_{L'+1}} \psi_{f(L'+1,k)}(\boldsymbol{x}_{v(L'+1,k)}) \right\} \end{array} \right].$$

Considering the term $\prod_{k=1}^{n_{L'+1}} \psi_{f(L'+1,k)}(\boldsymbol{x}_{v(L'+1,k)})$, since the set of child random variables of the factors in layer $L'$ coincides with the set of random variables in layer $L'+1$, we have $|Cf(L',1)| + \cdots + |Cf(L',m_{L'})| = n_{L'+1}$ and $|Cv(L',1)| + \cdots + |Cv(L',n_{L'})| = m_{L'}$. Thus, we can rewrite it as follows:

$$\prod_{k=1}^{n_{L'+1}} \psi_{f(L'+1,k)}(\boldsymbol{x}_{v(L'+1,k)}) = \prod_{j=1}^{n_{L'}} \prod_{\lambda_2 \in Cv(L',j)} \prod_{v(L'+1,k) \in Cf(\lambda_2)} \psi_{f(L'+1,k)}(\boldsymbol{x}_{v(L'+1,k)}).$$

$$P(\boldsymbol{x}_{v(1,1)}) = \sum_{\boldsymbol{x}_{v(2,1)}} \cdots \sum_{\boldsymbol{x}_{v(L'+1,n_{L'+1})}} \left[ \begin{array}{l} \left\{ \prod_{l=1}^{L'-1} \prod_{i=1}^{n_l} \prod_{\lambda_1 \in Cv(l,i)} \psi_{\lambda_1}(\boldsymbol{x}_{\partial\lambda_1}) \right\} \\ \times \left\{ \prod_{j=1}^{n_{L'}} \prod_{\lambda_2 \in Cv(L',j)} \psi_{\lambda_2}(\boldsymbol{x}_{\partial\lambda_2}) \left( \prod_{v(L'+1,k) \in Cf(\lambda_2)} \psi_{f(L'+1,k)}(\boldsymbol{x}_{v(L'+1,k)}) \right) \right\} \end{array} \right]$$

$$= \sum_{\boldsymbol{x}_{v(2,1)}} \cdots \sum_{\boldsymbol{x}_{v(L',n_{L'})}} \left\{ \prod_{l=1}^{L'-1} \prod_{i=1}^{n_l} \prod_{\lambda_1 \in Cv(l,i)} \psi_{\lambda_1}(\boldsymbol{x}_{\partial\lambda_1}) \right\} \times$$
$$\left[ \sum_{\boldsymbol{x}_{v(L'+1,1)}} \cdots \sum_{\boldsymbol{x}_{v(L'+1,n_{L'+1})}} \prod_{j=1}^{n_{L'}} \prod_{\lambda_2 \in Cv(L',j)} \psi_{\lambda_2}(\boldsymbol{x}_{\partial\lambda_2}) \left( \prod_{v(L'+1,k) \in Cf(\lambda_2)} \psi_{f(L'+1,k)}(\boldsymbol{x}_{v(L'+1,k)}) \right) \right]. \tag{2.8}$$

Here, we consider changing the order of summation and multiplication. Although this is not generally valid, it holds when the terms being summed are independent of

each other. Specifically, we group the random variables in layer $L'+1$ based on their association with the factors in layer $L'$. When we take an arbitrary factor $\lambda_2$ in layer $L'$, the underlined term in Equation (2.8) depends only on the random variables in $\partial\lambda_2$ ($\supset Cf(\lambda_2)$).

$$P(x_{v(1,1)}) = \sum_{x_{v(2,1)}} \cdots \sum_{x_{v(L',n_{L'})}} \left\{ \prod_{l=1}^{L'-1} \prod_{i=1}^{n_l} \prod_{\lambda_1 \in Cv(l,i)} \psi_{\lambda_1}(\boldsymbol{x}_{\partial\lambda_1}) \right\} \times \left[ \prod_{j=1}^{n_{L'}} \prod_{\lambda_2 \in Cv(L',j)} \underline{\sum_{\boldsymbol{x}_{Cf(\lambda_2)}} \psi_{\lambda_2}(\boldsymbol{x}_{\partial\lambda_2}) \left( \prod_{v(L'+1,k) \in Cf(\lambda_2)} \psi_{f(L'+1,k)}(x_{v(L'+1,k)}) \right)} \right]. \quad (2.9)$$

In Equation (2.9), we introduce the following definitions:

$$\phi_{\lambda_1}(\boldsymbol{x}_{\partial\lambda_1}) \coloneqq \psi_{\lambda_1}(\boldsymbol{x}_{\partial\lambda_1}),$$
$$\phi_{\lambda_2}(x_{\partial\lambda_2}) \coloneqq \sum_{\boldsymbol{x}_{Cf(\lambda_2)}} \psi_{\lambda_2}(\boldsymbol{x}_{\partial\lambda_2}) \left( \prod_{v(L'+1,k) \in Cf(\lambda_2)} \psi_{f(L'+1,k)}(x_{v(L'+1,k)}) \right).$$

This leads

$$P(x_{v(1,1)}) = \sum_{x_{v(2,1)}} \cdots \sum_{x_{v(L',n_{L'})}} \left\{ \prod_{l=1}^{L'} \prod_{i=1}^{n_l} \prod_{\lambda_1 \in Cv(l,i)} \phi_{\lambda_1}(\boldsymbol{x}_{\partial\lambda_1}) \right\}.$$

Now, we consider a new factor model where $\phi_\lambda$ represents the factors, and the maximum number of layers in the factor tree is $L'$. By the induction hypothesis, the proposition holds, so we have

$$P(x_{v(1,1)}) \propto \prod_{\lambda \in \partial v(1,1)} v'_{\lambda \to v(1,1)}(x_{v(1,1)})$$

Therefore, finally, we show that $v$ and $v'$ are equivalent. First, for $v'$, let $X_i$ be an arbitrary random variable in layer $L'$, and let $\alpha_\lambda$ be any of its child factors. Then,

$\nu'_{\lambda\to i}(x_i)$ can be expressed as follows:

$$\nu'_{\lambda\to i}(x_i) \propto \sum_{\boldsymbol{x}_{\partial\lambda\setminus i}} \phi_\lambda(x_i, \boldsymbol{x}_{\partial\lambda\setminus i}) \prod_{j\in\partial\lambda\setminus i} \nu'_{j\to\lambda}(x_j)$$

since this factor model has a factor tree with $L'$ layers, $Cf(\lambda)\setminus i = \emptyset$

$$\propto \phi_\lambda(x_i) = \sum_{\boldsymbol{x}_{Cf(\lambda)}} \psi_\lambda(\boldsymbol{x}_{\partial\lambda}) \left( \prod_{v(L'+1,k)\in Cf(\lambda)} \psi_{f(L'+1,k)}(x_{v(L'+1,k)}) \right). \tag{2.10}$$

On the other hand, for $\nu$, we have

$$\begin{aligned}
\nu_{\lambda\to i}(x_i) &\propto \sum_{\boldsymbol{x}_{Cf(\lambda)}} \psi_\lambda(\boldsymbol{x}_{\partial\lambda}) \prod_{j\in Cf(\lambda)} \nu_{j\to\lambda}(x_j) \\
&\propto \sum_{\boldsymbol{x}_{Cf(\lambda)}} \psi_\lambda(\boldsymbol{x}_{\partial\lambda}) \prod_{j\in Cf(\lambda)} \prod_{\beta\in\partial j\setminus\lambda} \nu_{\beta\to j}(x_j) \\
&= \sum_{\boldsymbol{x}_{Cf(\lambda)}} \psi_\lambda(\boldsymbol{x}_{\partial\lambda}) \prod_{v(L'+1,k)\in Cf(\lambda)} \prod_{\beta\in Cv(L'+1,k)} \nu_{\beta\to v(L'+1,k)}(v_{v(L'+1,k)}).
\end{aligned}$$

Because in the original factor tree, each random variable in layer $L'+1$ has only one child factor,

$$\begin{aligned}
\nu_{\lambda\to i}(x_i) &\propto \sum_{\boldsymbol{x}_{Cf(\lambda)}} \psi_\lambda(\boldsymbol{x}_{\partial\lambda}) \prod_{v(L'+1,k)\in Cf(\lambda)} \nu_{f(L'+1,k)\to v(L'+1,k)}(v_{v(L'+1,k)}) \\
&\propto \sum_{\boldsymbol{x}_{Cf(\lambda)}} \psi_\lambda(\boldsymbol{x}_{\partial\lambda}) \prod_{v(L'+1,k)\in Cf(\lambda)} \sum_{\boldsymbol{x}_{Cf(L'+1,k)}} \psi_{f(L'+1,k)}(\boldsymbol{x}_{\partial f(L'+1,k)}) \prod_{j\in Cf(L'+1,k)} \nu_{j\to f(L'+1,k)}(x_j).
\end{aligned}$$

Since $Cf(L'+1,k) = \emptyset$, we have

$$\nu_{\lambda\to i}(x_i) \propto \sum_{\boldsymbol{x}_{Cf(\lambda)}} \psi_\lambda(\boldsymbol{x}_{\partial\lambda}) \prod_{v(L'+1,k)\in Cf(\lambda)} \psi_{f(L'+1,k)}(x_{v(L'+1,k)}). \tag{2.11}$$

Therefore, from Equations (2.10) and (2.11), we have $\nu = \nu'$ for messages from factors in layer $L'$ to random variables in layer $L'$. Since messages from random variables in layer $L'$ to layer $L'-1$ are computed based on the messages in layer $L'$ according to the definition of messages, we have $\nu = \nu'$ for all messages up to layer $L'$. Thus, we have

$$P(x_{v(1,1)}) \propto \prod_{\lambda \in \partial v(1,1)} \nu_{\lambda \to v(1,1)}(x_{v(1,1)}).$$

From the proof for $L = 1$ and the proof for $L = L' + 1$ under the assumption of $L = L'$, we have shown by induction that the proposition holds for any $L$. □

# 3

# Complex non-backtracking Matrix and its Application

## 3.1 Background

To investigate graph structures, research on the spectra of undirected graphs has been conducted for a long time. Several characteristics of undirected graphs, including the number of connected components, bipartite structure, and diameter, have well-established relationships with eigenvalues of matrices (Chung 1996). In contrast, the study of eigenvalues in directed graphs remains relatively underexplored. One of the reasons is that the adjacency matrix of directed graphs, being asymmetric, typically yields complex eigenvalues, complicating the analysis. To address this challenge while preserving the intrinsic information of directed graphs, the Hermitian adjacency matrices have been introduced in Guo and Mohar (2017) and Liu and X. Li (2015). Compared with the spectral analysis of undirected graphs, the theory based on Hermitian adjacency matrices is still under development, yet it is regarded as a

promising approach for investigating properties of directed graphs.

Clustering is one of the areas in which the usefulness of Hermitian adjacency matrices has attracted attention. In node clustering for undirected graphs, the classical objective is to identify clusters that are densely connected internally and sparsely connected between clusters. In contrast, for directed graphs, it may be desirable to cluster nodes based on similar inflow and outflow patterns. With respect to these qualitative characteristics of cluster structures, the normalized cut defined in Equation (2.2) has been widely used as a quantitative measure in undirected graphs. For directed graphs, while a representative quantitative measure has yet to be established, several metrics have been employed in existing studies to evaluate the validity of numerical experimental results. For example, Cucuringu et al. (2020) introduced the *Cut Imbalance ratio (CI)* to quantify the degree of imbalance between two clusters $X$ and $Y$:

$$\mathrm{CI}(X,Y) \coloneqq \left| \frac{\overrightarrow{\mathrm{cut}}(X,Y)}{\overrightarrow{\mathrm{cut}}(X,Y) + \overrightarrow{\mathrm{cut}}(Y,X)} - \frac{1}{2} \right|, \quad \text{where } \overrightarrow{\mathrm{cut}}(X,Y) \coloneqq \left| \left\{ \overrightarrow{uv} \in \vec{E} \mid u \in X, v \in Y \right\} \right|.$$

Laenen and Sun (2020) proposed a directed extension of the normalized cut and formulated the objective of finding a $K$-partition $\{S_1, \ldots, S_K\}$ that maximizes the *flow ratio* defined below.

$$\sum_{j=1}^{K-1} \frac{\overrightarrow{\mathrm{cut}}(S_j, S_{j+1})}{\mathrm{vol}(S_j) + \mathrm{vol}(S_{j+1})}$$

They also discussed the relationship between the eigenvalues of the considered matrix representation and the resulting flow ratio.

A classical approach to node clustering in directed graphs is to first construct a symmetrized matrix representation and then apply clustering methods for undirected graphs. The simplest symmetrization is to use the sum of the adjacency matrix and its transpose $A + A^\top$ instead of the adjacency matrix $A$ itself. However, this approach discards the directional information of the graph. Another approach is to work with the matrix products of the adjacency matrix and its transpose (Satuluri and Parthasarathy 2011) such as $AA^\top$ and $A^\top A$. As discussed in Cucuringu et al. (2020) and Rohe, Qin,

and Yu (2016), the $(u, v)$ entry of $AA^\top$ represents the number of common destination vertices among the outgoing neighbors of $u$ and $v$, whereas the $(u, v)$ entry of $A^\top A$ represents the number of common source vertices among the incoming neighbors of $u$ and $v$. One can estimate clusters that reflect directionality more faithfully than naive symmetrization. Rohe, Qin, and Yu (2016) proposed a method that performs a singular value decomposition of the asymmetric adjacency matrix, and uses the left and right singular vectors for clustering. It has further been reported that methods based on the Hermitian adjacency matrix can identify more desirable clusters than these approaches (Cucuringu et al. 2020; Laenen and Sun 2020; Martin, Rogers, and Zanetti 2024).

However, these approaches do not always achieve satisfactory performance. As illustrated in Figure 3.1, in undirected graphs, when the target graph is sparse, spectral clustering methods based on adjacency-derived matrices can fail to recover the correct clusters. A similar issue can also arise in directed graphs. Since sparse graphs are common in real-world networks, developing methods that can capture more realistic graph structures remains an important challenge for the field.

### 3.1.1 Contributions

As a step toward developing a method that can be robustly applied across various situations, we introduce a complex non-backtracking (CNBT) matrix for directed graphs that incorporates characteristics of both the Hermitian adjacency matrix and the non-backtracking (NBT) matrix. The contributions of this work are summarized as follows:

1. We show that the relationship between the Hermitian adjacency matrix and the CNBT matrix for directed graphs exhibits favorable properties, analogous to the relationship between the adjacency matrix and the NBT matrix in undirected graphs.

2. We demonstrate through experiments that in the node clustering problem, spectral clustering based on the CNBT matrix performs effectively in scenarios where methods based on the Hermitian adjacency matrix lead to misclassification, especially when the underlying graph is sparse.

3. We show that spectral clustering based on the CNBT matrix has a connection to the belief propagation method.

Regarding our third contribution, viewing the cluster structure of directed graphs within the framework of the stochastic block model, we show that models in which inter-cluster connectivity exhibits a cyclic pattern are consistent with our proposed method. In other words, in this study, two vertices are said to belong to the same cluster when they share the same edge generation probability parameters. Under this model, we demonstrate that our model aligns with the cyclic structure defined in Equation (3.6).

In addition, we summarize the characteristics of existing methods below. Our study focuses on the sparse scenario in node clustering for directed graphs, which has not yet been sufficiently examined, and proposes a method that works robustly even in such situations.

**Clustering based on Graph Laplacian (Ng, Jordan, and Weiss 2001)** A standard clustering method using eigenvectors of the normalized graph Laplacian $\mathcal{L}$ defined in Section 2.4.
**Direction:** undirected, **Matrix Representation:** $\mathcal{L}$, **Sparsity:** not robust

**Clustering based on NBT Matrix (Krzakala et al. 2013; M. E. J. Newman 2013)** A clustering method that has been confirmed to perform well in sparse undirected graphs using eigenvectors of the NBT matrix defined in Section 3.2.1.
**Direction:** undirected, **Matrix Representation:** $B$, **Sparsity:** robust

**DI-SIM (Rohe, Qin, and Yu 2016)** A clustering method that first normalizes the asymmetric adjacency matrix by in-degrees and out-degrees, then performs singular value decomposition, and uses the left and right singular vectors.
**Direction:** directed, **Matrix Representation:** $A$, **Sparsity:** not verified

**BI-SYM and DD-SYM (Satuluri and Parthasarathy 2011)** Clustering methods that include BI-SYM, which uses the product of the adjacency matrix and its transpose for symmetrization, and DD-SYM, a degree-normalized version of this symmetrization.
**Direction:** directed, **Matrix Representation:** $AA^\top + A^\top A$, **Sparsity:** not verified

**Herm (Cucuringu et al. 2020)** A clustering method that uses the Hermitian adjacency matrix defined in Section 3.2.2 with $\alpha = i$.
**Direction:** directed, **Matrix Representation:** $A_i$, **Sparsity:** not verified

**SimpleHerm (Laenen and Sun 2020)** A clustering method that uses the graph Laplacian based on the Hermitian adjacency matrix defined in Section 3.2.2.
**Direction:** directed, **Matrix Representation:** $A_\alpha$, **Sparsity:** not verified

## 3.2 Related Work

### 3.2.1 Non-backtracking Matrix

In the node clustering problem for undirected graphs, spectral clustering (Luxburg 2007; Ng, Jordan, and Weiss 2001) as described in Section 2.5 is known to perform effectively when the graph is dense. In contrast, when the graph is sparse, i.e., the number of edges is on the order of a constant multiple of the number of vertices, spectral clustering based on the eigenvalues and eigenvectors of the adjacency matrix or the graph Laplacian has been shown to suffer a pronounced degradation in performance (Krzakala et al. 2013). In such situations, alternative approaches such as belief propagation (Mezard and Montanari 2009) have been theoretically and experimentally confirmed to achieve performance approaching the detectability limit for community recovery (Decelle et al. 2011; Krzakala et al. 2013). Spectral clustering using the non-backtracking (NBT) matrix is also one of the methods that has been confirmed to perform well in sparse undirected graphs (Krzakala et al. 2013; M. E. J. Newman 2013).

In this section, we describe spectral clustering using the NBT matrix for the node clustering problem in undirected graphs, based on Krzakala et al. (2013).

In an undirected graph $\mathcal{G} = (V, E)$, we write $B \in \{0, 1\}^{2m \times 2m}$ as the NBT matrix defined in Equation (2.1) (Hashimoto 1989), where $m$ is the number of undirected edges. Since the NBT matrix $B$ indexes directed edges obtained by orienting undirected edges, we treat each directed edge $e \in \vec{E}$ itself as an index.

In spectral clustering described in Algorithm 1, the eigenvectors of the graph Laplacian have length equal to the number of vertices, and the component indexed by each vertex contains the cluster information for the vertex. In contrast, since

**Algorithm 5:** Spectral Clustering based on NBT Matrix (Krzakala et al. 2013)

**Input:** $\mathcal{G} = (V, E)$: undirected graph; $K \in \mathbb{Z}$: number of clusters
**Output:** $z \in \{1, \ldots, K\}^{|V|}$: cluster assignment for each vertex

1 Compute the NBT matrix $B$
2 Obtain the eigenvectors of $B$ corresponding to the top $K$ eigenvalues with the largest real parts
3 Construct a matrix $X \in \mathbb{R}^{2m \times K}$ by arranging the eigenvectors as columns
4 For each column of $X$, compute its in-vector or out-vector to form a matrix $Y \in \mathbb{R}^{|V| \times K}$
5 Normalize each row of $Y$ to have a norm of 1
6 Apply the K-means algorithm to the row vectors of $Y$
7 Let the cluster assignment for vertex $v_i$ be $z_i$, corresponding to the cluster assignment of the $i$-th row of $Y$
8 **return** $z = (z_1, \ldots, z_{|V|})$

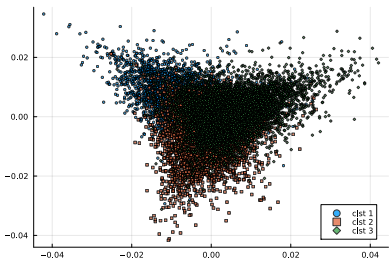
the eigenvectors of the NBT matrix have length $2m$, one cannot directly extract vertex-related information. Krzakala et al. (2013) addresses this issue by aggregating the components associated with directed edges related to each vertex to construct vertex-wise cluster information.

**Definition 3.1** (in/out-vectors (Krzakala et al. 2013))**.** *Let $\mathcal{G} = (V, E)$ be an undirected graph, $n$ and $m$ be the number of vertices and undirected edges, respectively, and $\boldsymbol{g} \in \mathbb{C}^{2m}$ be $2m$-dimensional vector. The in- and out-vectors of $\boldsymbol{g}$ are denoted by $\boldsymbol{g}^{in}, \boldsymbol{g}^{out} \in \mathbb{C}^n$, respectively, and their components at vertex $u \in V$ are defined as follows:*

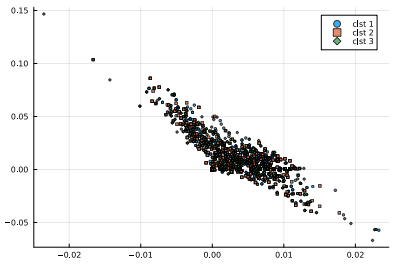
$$\left(\boldsymbol{g}^{in}\right)_u \coloneqq \sum_{v \in N_u} (\boldsymbol{g})_{\overrightarrow{vu}}, \qquad \left(\boldsymbol{g}^{out}\right)_u \coloneqq \sum_{v \in N_u} (\boldsymbol{g})_{\overrightarrow{uv}}.$$

This transformation allows us to obtain vertex-indexed vectors from an eigenvector of the NBT matrix, which is indexed by directed edges. Algorithm 5 describes an algorithm for node clustering in undirected graphs. Specifically, when estimating $K$ clusters, we extract the eigenvectors corresponding to the top $K$ eigenvalues with the largest real parts, compute their in-vectors or out-vectors, and then apply the K-means algorithm.

Figure 3.1 shows scatter plots of the eigenvector values obtained by applying spectral clustering based on the NBT matrix and the normalized graph Laplacian to a




(a) Scatter plot of the 2nd and 3rd eigenvectors in spectral clustering using the NBT matrix. ARI ≒ 0.145.




(b) Scatter plot of the 1st and 2nd eigenvectors in spectral clustering using the normalized graph Laplacian. ARI ≒ 0.0.

Figure 3.1: An illustration of node clustering in a sparse undirected graph with three clusters. A scatter plot of each vertex using two of the three eigenvectors of interest.

graph generated under the experimental settings in Krzakala et al. (2013). Specifically, we generate an undirected graph with three clusters using the stochastic block model (M. Newman 2010). The graph has $n = 30000$ vertices in total, with an equal number of vertices in each cluster. An edge between two vertices is present with probability $c_{\text{in}}/n$ if the vertices belong to the same cluster, and with probability $c_{\text{out}}/n$ if they belong to different clusters.

For a graph generated under this setting, spectral clustering using the normalized graph Laplacian fails to reveal any cluster structure, and the Adjusted Rand Index (ARI) (Hubert and Arabie 1985), a performance metric for cluster estimation, is nearly zero. In contrast, spectral clustering using the NBT matrix exhibits a substantial overlap among clusters, yet still a nontrivial cluster structure and yields an improved ARI.

### 3.2.2 Hermitian Adjacency Matrix

In spectral graph theory, graph structures are investigated through eigenvalues and eigenvectors of matrices such as adjacency matrices and graph Laplacians. In undirected graphs, extensive research has been conducted for a long time, establishing relationships between various graph properties, such as the number of connected components, bipartite structures, and diameters, and their eigenvalues. On the other hand, in directed graphs, the asymmetry of the adjacency matrix often leads to complex eigenvalues, and the exploration of graph structures through graph spectra has

not been fully developed. Hermitian adjacency matrices were introduced as matrix representations that maintain symmetry while preserving the directional information of graphs by incorporating complex numbers into the matrix representation.

In this chapter, let $\alpha \in \mathbb{C}$ be a complex number with an absolute value of 1, and $G = (V, \vec{E})$ be a directed graph with $n$ vertices. Although several definitions of Hermitian adjacency matrices have been proposed (Guo and Mohar 2017; Liu and X. Li 2015; Martin, Rogers, and Zanetti 2024; Mohar 2020), we focus on the Hermitian adjacency matrix $A_\alpha \in \mathbb{C}^{n\times n}$ of a directed graph $G$ defined as follows (Guo and Mohar 2017; Liu and X. Li 2015):

$$(A_\alpha)_{uv} \coloneqq \begin{cases} 1 & u \leftrightarrow_G v, \\ \alpha & u \rightarrow_G v, \\ \bar{\alpha} & u \leftarrow_G v, \\ 0 & \text{otherwise} \end{cases} \qquad u, v \in V, \tag{3.1}$$

where $\bar{\alpha}$ is a conjugate complex number of $\alpha$. We note that $A \in \{0, 1\}^{n\times n}$ denotes the standard adjacency matrix, where $(A)_{uv} = 1$ if there is a directed edge from $u$ to $v$, and 0 otherwise.

## 3.3 Complex non-backtracking Matrix

### 3.3.1 Rotation in a mixed walk

We first introduce the concept of "rotation" in a mixed walk. In graph theory, it is common to count walks on a graph through powers of the adjacency matrix. In undirected graphs, when counting walks, they are classified and counted based on their starting vertex, ending vertex, and length. However, in directed graphs, it is also necessary to consider the directional information. The concept of rotation enables us to count mixed walks while taking into account the difference between the number of times directed edges are traversed in the forward direction and the number of times they are traversed in the reverse direction.

Support that $\alpha \in \mathbb{C}$ and $R \in \mathbb{N}$ are chosen such that $R \geq 3$ and $R$ is the smallest positive integer satisfying $\alpha^R = 1$. In clustering using Hermitian adjacency matrices, it

has been reported that the method is effective when there is a cyclic structure between clusters (Laenen and Sun 2020). Accordingly, we introduce $R$ here to explicitly account for the period of such cyclicity. For $\overrightarrow{uv} \in \bar{E}$, we define its rotation $r(u,v)$ as follows:

$$r(u,v) \coloneqq \begin{cases} 1 & u \rightarrow_G v, \\ -1 & u \leftarrow_G v, \\ 0 & u \leftrightarrow_G v. \end{cases}$$

This value $r(u,v)$ indicates whether traversing from $u$ to $v$ aligns with the direction of the edge in $G$. We note that since this rotation is defined for oriented pairs in $\bar{E} = \vec{E} \cup \vec{E}^{-1}$, the relationship between two vertices falls into one of the three categories mentioned above. We also note that traversals between two vertices connected by bidirectional edges are not counted, and $r(v,u) = -r(u,v)$ holds. The rotation of a mixed walk $W = (v_1, \ldots, v_k)$ is defined by $r(W) \coloneqq \sum_{l=1}^{k-1} r(v_l, v_{l+1}) \bmod R$. This quantity represents the remainder of the difference between the number of times the mixed walk $W$ traverses directed edges in the correct direction and the number of times it traverses them in the reverse direction.

### 3.3.2 Complex non-backtracking Matrix

We also introduce a matrix representation that describes the relationship between directed edges. Let $B$ be the standard NBT matrix defined in Equation (2.1). For $e \in \bar{E}$, we define a directional type $\lambda_e \in \mathbb{C}$ of $e \in \bar{E}$, and propose the complex non-backtracking (CNBT) matrix $B_\alpha \in \mathbb{C}^{2m \times 2m}$ as follows:

$$B_\alpha \coloneqq B\Lambda, \quad \text{where} \quad \lambda_e \coloneqq \begin{cases} 1 & i_e \leftrightarrow_G t_e, \\ \alpha & i_e \rightarrow_G t_e, \\ \bar{\alpha} & i_e \leftarrow_G t_e, \end{cases} \quad \Lambda \coloneqq \operatorname{diag}\begin{pmatrix} \lambda_1 & \ldots & \lambda_{2m} \end{pmatrix},$$

where $2m \coloneqq \left|\bar{E}\right|$ and we treat each element $e$ of $\bar{E}$ and its corresponding index as interchangeable. To simplify notations in the following discussion, we introduce the auxiliary matrices. Let $\overleftrightarrow{A}, \overrightarrow{A}, \overleftarrow{A} \in \mathbb{R}^{n \times n}$ be the matrices where $(\overleftrightarrow{A})_{uv} = 1$ if $u \leftrightarrow_G v$ and 0 otherwise, $(\overrightarrow{A})_{uv} = 1$ if $u \rightarrow_G v$ and 0 otherwise, and $(\overleftarrow{A})_{uv} = 1$ if $u \leftarrow_G v$ and 0

otherwise, respectively. Similarly, define $\overleftrightarrow{B}, \overrightarrow{B}, \overleftarrow{B} \in \mathbb{R}^{2m\times 2m}$ be the matrices where $(\overleftrightarrow{B})_{ef} = (B)_{ef}$ if $i_f \leftrightarrow_G t_f$ and 0 otherwise, $(\overrightarrow{B})_{ef} = (B)_{ef}$ if $i_f \rightarrow_G t_f$ and 0 otherwise, and $(\overleftarrow{B})_{ef} = (B)_{ef}$ if $i_f \leftarrow_G t_f$ and 0 otherwise, respectively. These notations lead $A_\alpha = \overleftrightarrow{A} + \alpha\overrightarrow{A} + \bar{\alpha}\overleftarrow{A}$ and $B_\alpha = \overleftrightarrow{B} + \alpha\overrightarrow{B} + \bar{\alpha}\overleftarrow{B}$ hold.

The number of mixed walks plays a crucial role in proving the results of our works. Let us define matrices $P_{(k,r)} \in \mathbb{Z}^{n\times n},\ k \geq 0, r \in \{0,\dots,R-1\}$ whose $(u,v)$ entry $(P_{(k,r)})_{u,v}$ is the number of NBT mixed walks in $G$ of length $k$ and rotation $r$, starting at $u$ and ending at $v$. We set $P_{(0,0)} = I$ and $P_{(0,r)} = \mathbf{0},\ r \neq 0$, where $I$ represents the identity matrix. Note that $-1 \equiv R-1 \pmod R$, it is clear that $P_{(1,0)} = \overleftrightarrow{A},\ P_{(1,1)} = \overrightarrow{A},\ P_{(1,R-1)} = \overleftarrow{A},\ P_{(1,r)} = \mathbf{0},\ r \neq 0, 1, R-1$ from the definitions of $\overleftrightarrow{A}, \overrightarrow{A}$ and $\overleftarrow{A}$. As similar quantities, let $n_{(k,r)} \in \mathbb{Z}$ be the number of NBT cycles of length $k$ and rotation $r$ without a tail. We also define matrices $Q_{(k,r)} \in \mathbb{Z}^{2m\times 2m},\ k \geq 0, r \in \{0,\dots,R-1\}$ whose $(e,f)$ entry $(Q_{(k,r)})_{ef}$ represents the number of NBT mixed walks of length $k$ and rotation $r$ that end with $f$, to which the oriented pair $e$ can be prepended while maintaining the NBT property. To be more specific, let $W$ be a mixed walk counted in $(Q_{(k,r)})_{ef}$. The definition of $(Q_{(k,r)})_{ef}$ implies that the oriented pair $e$ can be connected to the beginning of $W$, and the resulting mixed walk, formed by concatenating $e$ to the beginning of $W$, remains NBT. We set $Q_{(0,0)} = I$ and $Q_{(0,r)} = \mathbf{0},\ r \neq 0$. It is evident that $Q_{(1,0)} = \overleftrightarrow{B}, Q_{(1,1)} = \overrightarrow{B}, Q_{(1,R-1)} = \overleftarrow{B}$ and $Q_{(1,r)} = \mathbf{0},\ r \neq 0, 1, R-1$ from the definition of $\overleftrightarrow{B}, \overrightarrow{B}$ and $\overleftarrow{B}$.

### 3.3.3 Basic Properties

We show some features about the number of mixed walks. These results extend the properties in Stark and Terras (1996) and Tarfulea and Perlis (2009) to mixed walks in the sense that traversing a directed edge in the reverse direction is permitted.

Let $r_k \coloneqq \sum_{r=0}^{R-1} \alpha^r P_{(k,r)} \in \mathbb{C}^{n\times n}$. The value $(r_k)_{u,v}$ is analogous to the number of NBT walks of length $k$ between two vertices $u$ and $v$ in an undirected graph. However, unlike the undirected case, we must distinguish whether each directed edge in a mixed walk of length $k$ is traversed in the forward or reverse direction. The entry $(r_k)_{u,v}$ represents the $\alpha^r$-weighted sum of the number of NBT mixed walks of length $k$ and rotation $r$ from $u$ to $v$, taken over all possible rotations. With respect to $r_k$, Lemma 3.2 holds.

**Lemma 3.2.** *Let $G$ be a directed graph, then we have*

$$r_1 = A_\alpha, \quad r_2 = A_\alpha^2 - D,$$
$$r_k = r_{k-1}A_\alpha - r_{k-2}(D - I), \quad k \geq 3.$$

The following recurrence relations hold for $Q_{(k,r)}$.

**Lemma 3.3.** *Let $G$ be a directed graph. For any $k \geq 1$ and $r \in \{0, \ldots, R-1\}$, the following holds:*

$$Q_{(k,r)} = \overleftrightarrow{B} Q_{(k-1,r)} + \overrightarrow{B} Q_{(k-1,r-1)} + \overleftarrow{B} Q_{(k-1,r+1)}.$$

Lemma 3.3 leads to Corollary 3.1.

**Corollary 3.1.** *For $k \geq 1$, $(B_\alpha)^k = \sum_{r=0}^{R-1} \alpha^r Q_{(k,r)}$.*

### 3.3.4 Relation between $A_\alpha$ and $B_\alpha$ via the Ihara's formula

We show that the CNBT matrix is a natural extension of the conventional NBT matrix, in that relations analogous to those between the classical adjacency matrix and the NBT matrix hold between the Hermitian adjacency matrix and the CNBT matrix. This is supported by two results, Corollary 3.2 and Theorem 3.4.

We discuss the complex version of the Ihara's formula, which is a theorem that describes the zeta function of a graph can be expressed as a rational function, and it is also one of the equations that connects the adjacency matrix and the NBT matrix. The Ihara zeta function on undirected graphs $Z_X(u)$ is defined as a product over all equivalence classes of NBT cycles without a tail (Stark and Terras 1996):

$$Z_X(u) := \prod_{\substack{[C] \\ \text{primitive} \\ \text{NBT, no tail}}} (1 - u^{|C|})^{-1},$$

where the length $|C|$ of a cycle is defined as the number of edges that constitute the cycle. Corollary 3.2 provides a variant of Ihara's formula, allowing reverse traversal of directed edges. To derive this result, we use the weighted zeta function of directed

graphs as follows (Konno et al. 2019):

$$z_X(u;\alpha) \coloneqq \prod_{\substack{[C] \\ \text{primitive} \\ \text{NBT, no tail}}} (1-\alpha^{r(C)}u^{|C|})^{-1}.$$

In the zeta function for undirected graphs, equivalence classes of NBT cycles are formed based solely on their cycle lengths. In contrast, this paper introduces a concept of rotation for directed graphs, where the equivalence classes of cycles are determined by considering not only their lengths but also their rotations. As a result, the rotation information is encoded as a weight in the weighted zeta function.

The Hermitian adjacency matrix defined as Equation (3.1) is a type of the weighted matrix defined in Konno et al. (2019). Considering Konno et al. (2019, Theorem 3), we obtain the following result, which is a variant of Ihara's formula.

**Corollary 3.2.** *Let $G$ be a directed graph on $n$ vertices and $m$ unoriented edges. Then,*

$$\det(I-uB_\alpha) = (1-u^2)^{m-n}\det\left(I-uA_\alpha+u^2(D-I)\right).$$

This result clarifies that the results of Konno et al. (2019) on general weighted adjacency matrices and Stark and Terras (1996)'s Theorem 3 can also be applied in cases where the weights have specific meanings as powers of roots of unity, as defined in Section 3.3.1.

We further explore the complex number $\alpha$ of the Hermitian adjacency matrix that is consistent with our proposed clustering method in Section 3.4.4.

### 3.3.5 Relation between $A_\alpha$ and $B_\alpha$ via in/out-vectors

Since the NBT matrix represents the relation between pairs, its eigenvector is indexed by pairs in $\bar{E}$. To extract vertex-related information from the eigenvectors of the NBT matrix, we need to convert a vector indexed by pairs to a vector indexed by vertices. Such an approach has been adopted in Krzakala et al. (2013) and M. E. J. Newman (2013), and here we introduce an extension of the approach. The resulting vertex-indexed vectors are utilized in the clustering algorithm proposed in Section 3.4.

Assume that $\boldsymbol{g} \in \mathbb{C}^{2m}$ is a $2m$ dimensional vector, where each index corresponds

to an oriented pair in $\bar{E}$. We introduce the in- and out-vector of $\boldsymbol{g}$, denoted by $\boldsymbol{g}_\alpha^{\text{in}}$, $\boldsymbol{g}_\alpha^{\text{out}} \in \mathbb{C}^n$, respectively. The elements of $\boldsymbol{g}_\alpha^{\text{in}}$ and $\boldsymbol{g}_\alpha^{\text{out}}$ at vertex $u$ are defined as follows:

$$\left(\boldsymbol{g}_\alpha^{\text{in}}\right)_u \coloneqq \sum_{v \in N_u} g_{\overrightarrow{vu}}, \tag{3.2}$$

$$\left(\boldsymbol{g}_\alpha^{\text{out}}\right)_u \coloneqq \sum_{v \in \overleftrightarrow{N_u}} g_{\overrightarrow{uv}} + \alpha \sum_{v \in \overrightarrow{N_u}} g_{\overrightarrow{uv}} + \bar{\alpha} \sum_{v \in \overleftarrow{N_u}} g_{\overrightarrow{uv}}. \tag{3.3}$$

With respect to these two vectors, the following theorem holds.

**Theorem 3.4.** *For any $g \in \mathbb{C}^{2m}$, the following holds:*

$$\begin{pmatrix} (B_\alpha \boldsymbol{g})_\alpha^{out} \\ (B_\alpha \boldsymbol{g})_\alpha^{in} \end{pmatrix} = \begin{pmatrix} A_\alpha & -I \\ D - I & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{g}_\alpha^{out} \\ \boldsymbol{g}_\alpha^{in} \end{pmatrix}. \tag{3.4}$$

Theorem 3.4 is an analogous result that is known to hold between the commonly known adjacency matrix and the NBT matrix of undirected graphs (Krzakala et al. 2013).

**Corollary 3.3.** *Let $(\lambda, \boldsymbol{u}) \in \mathbb{C} \times \mathbb{C}^{2m}$ be an eigenpair for $B_\alpha$. Then, $(\lambda, \begin{pmatrix} \boldsymbol{u}_\alpha^{out} \\ \boldsymbol{u}_\alpha^{in} \end{pmatrix})$ is an eigenpair for the matrix described in the right-hand side of Equation (3.4).*

Corollary 3.3 shows that the in/out vectors of an eigenvector of $B_\alpha$ are eigenvectors of the matrix described in the right-hand side of Equation (3.4). Since the matrix described in the right-hand side of Equation (3.4) is generally smaller than $B_\alpha$, we can efficiently obtain the transformed eigenvectors of $B_\alpha$.

## 3.4 Application in Clustering

We demonstrate that the CNBT matrix is effective for node clustering in directed graphs, especially when the underlying graph is sparse. A graph is said to be sparse if the number of edges is at most a constant multiple of the number of vertices. In the context of undirected graphs, spectral methods based on the adjacency matrix or on the derived graph Laplacians are widely used across various fields. However, it is

known that in sparse graphs, methods such as belief propagation (Yedidia, Freeman, and Weiss 2003) can perform well, whereas the spectral methods fail to estimate clusters accurately (Decelle et al. 2011). To resolve this issue, several approaches have been proposed, including methods based on the NBT matrix (Krzakala et al. 2013; M. E. J. Newman 2013; Saade, Krzakala, and Zdeborová 2014). Regarding the detectability condition based on the edge probabilities between and within clusters, methods using the NBT matrix have been experimentally shown to perform similarly to belief propagation and outperform methods based on the adjacency matrix (Krzakala et al. 2013).

Recently, the problem of directed graph clustering has received significant attention, highlighting the differences in problem compared to undirected graphs. Unlike undirected graph clustering, which focuses on dense intra-cluster and sparse inter-cluster connections, directed graph clustering considers the imbalance of directional information between clusters. Hermitian adjacency matrices are often used to address this problem due to its symmetry and ease of handling. The approach has been shown to perform well in Cucuringu et al. (2020), Laenen and Sun (2020), and Martin, Rogers, and Zanetti (2024).

We illustrate that the similar issue of spectral methods based on the adjacency matrix failing to work for sparse undirected graphs also arises in directed graphs. We experimentally show that a method based on a Hermitian adjacency matrix fail to resolve this issue, whereas the proposed method can mitigate the problem.

### 3.4.1 Our algorithm

We consider the problem of partitioning the vertices of a directed graph into $K$ clusters. In this problem, we regard a partition that groups together vertices with similar out-neighbors or in-neighbors as a good clustering.

We propose a spectral clustering (*CNBT-SC*, Algorithm 6) using the CNBT matrix. We set $\alpha = \exp\left(\frac{2\pi\sqrt{-1}}{K}\right)$ based on experimental trials and the prior research (Laenen and Sun 2020). Furthermore, we utilize $\lfloor \frac{K}{2} \rfloor$ eigenvectors, treating the real and imaginary components separately to obtain a final dimensionality of $K$. Our code is available at https://github.com/KeishiS/CNBT.

**Algorithm 6:** CNBT-SC (in/out)

**Input:** a digraph $G = (V, E)$, the number of clusters $K$
**Output:** clusters $S_1, \dots S_K \subset V$ with $S_i \cap S_j = \phi,\ i \neq j$

$\alpha \leftarrow \exp\left(\frac{2\pi\sqrt{-1}}{K}\right)$
Compute the CNBT matrix $B_\alpha$
Compute the eigenvectors $g_1, \dots, g_{\lfloor \frac{K}{2} \rfloor} \in \mathbb{C}^{2m}$ corresponding to the eigenvalues with the largest real part of $B_\alpha$
Compute $(g_1)_\alpha^{\text{out}}, \dots, (g_{\lfloor \frac{K}{2} \rfloor})_\alpha^{\text{out}} \in \mathbb{C}^n$ (or compute in-vectors)
$X \leftarrow \left((g_1)_\alpha^{\text{out}} \quad \dots \quad (g_{\lfloor \frac{K}{2} \rfloor})_\alpha^{\text{out}}\right) \in \mathbb{C}^{n \times \lfloor \frac{K}{2} \rfloor}$ (or apply to in-vectors)
Normalize each row of $X$ and denote the result as $\tilde{X}$
Use K-means algorithm to $\left(\text{real}(\tilde{X}) \quad \text{imag}(\tilde{X})\right)$

### 3.4.2 Experimental setup

To evaluate the performance of the proposed method, we conduct two experiments using directed graphs generated from stochastic block models (SBM). We use a directed graph generated by SBM as input to the algorithm, after removing the ground-truth cluster assignments for each vertex. The estimated cluster assignments are then evaluated against the ground-truth using the Adjusted Rand Index (Hubert and Arabie 1985; Rand 1971, ARI). The ARI is a measure of similarity between two clusterings, where a value close to 1 implies a high degree of similarity.

The first experiment compares the proposed algorithm with existing methods under the same experimental conditions (Cucuringu et al. 2020). Under the conditions, the directed stochastic block model (Cucuringu et al. 2020, DSBM) is used as a generative model for directed graphs. This model generates a directed graph with $n$ vertices belonging to one of $K$ clusters, and it includes parameters that control the probability of generating directed edges as follows. First, a directed edge is generated between each pair of vertices $\{u, v\}$ with probability $p$. Given that vertices $u$ and $v$ belong to clusters $a$ and $b$, respectively, the direction of the directed edge is determined with the

following probabilities.

$$F_{a,b} \coloneqq \begin{cases} 1-\eta & a+1 \equiv b \bmod K, \\ \eta & b+1 \equiv a \bmod K, \\ 0.5 & \text{otherwise.} \end{cases}$$

This means that the parameter $\eta$ controls the degree of generation of a circular pattern such as $1 \to 2 \to \cdots \to K \to 1$. Smaller values of $\eta$ enhance the circular pattern, whereas values approaching 0.5 gradually diminish the cluster structure.

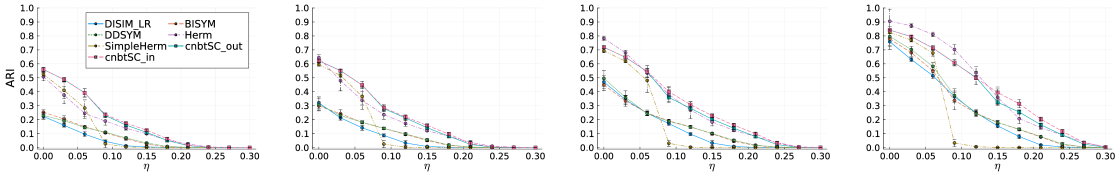
The second experiment investigates the influence of two parameters, $\epsilon$ and $\eta$ under sparse graphs. The parameter $\epsilon$ controls the strength of the circular pattern, and $\eta$ governs the ratio of intra-cluster edge probability to inter-cluster edge probability. In addition to the DSBM used in the first experiment, we also use the degree-corrected stochastic block model (DCSBM, Karrer and M. E. J. Newman (2011)), which tends to generate a few numbers of hub vertices. A generated graph consists of $n = 3000$ vertices divided into $K = 3$ clusters of the same size. Given that vertices $u$ and $v$ belong to cluster $a$ and $b$, respectively, a directed edge from $u$ to $v$ is generated with probability $\frac{\Gamma_{a,b}}{n}$ on DSBM. We note that the inclusion of $n$ in the denominator of the edge probability is the primary factor in generating sparse graphs. To control a circular pattern, we set $\Gamma_{a,b}$ as follows:

$$\Gamma_{a,b} \coloneqq \begin{cases} \Gamma_{\text{correct}} & a+1 \equiv b \bmod K, \\ \Gamma_{\text{reverse}} & b+1 \equiv a \bmod K, \\ \Gamma_{\text{intra}} & \text{otherwise.} \end{cases} \qquad \epsilon \coloneqq \frac{\Gamma_{\text{correct}}}{\Gamma_{\text{reverse}}}, \quad \eta \coloneqq \frac{\Gamma_{\text{reverse}}}{\Gamma_{\text{intra}}}.$$

We set $\Gamma_{\text{correct}}$, $\Gamma_{\text{reverse}}$ and $\Gamma_{\text{intra}}$ such that under the DSBM the expected degree $c$ of the generated graph is 5. Given $\epsilon$ and $\eta$, these three parameters are computed as

$$\Gamma_{\text{correct}} = \frac{2c\epsilon\eta}{1+\eta(1+\epsilon)}, \quad \Gamma_{\text{reverse}} = \frac{2c\eta}{1+\eta(1+\epsilon)}, \quad \Gamma_{\text{intra}} = \frac{2c}{1+\eta(1+\epsilon)}.$$

Under the DCSBM model, we assume that a directed edge from $u$ to $v$ is generated with probability $\frac{\theta_u \theta_v \Gamma_{a,b}}{n}$. The parameter $\theta_u \in \mathbb{R}$ allows for heterogeneous expected degrees across vertices. We assume $\theta_u$ follows a Pareto distribution with scale 1 and exponent




(a) $p = 0.45\%$ (b) $p = 0.5\%$ (c) $p = 0.6\%$ (d) $p = 0.8\%$

Figure 3.2: Recovery rates of the first experiment. The results show the average ARI over 10 independent simulations.

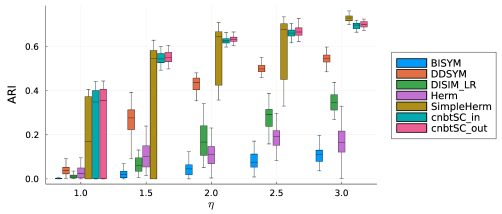
$-2.5$, we reuse the same parametrization of $\Gamma$ above. Note that in the DCSBM, $c$ serves as a scaling constant for $\Gamma$ and does not coincide with the expected degree.

We compare the performance of our algorithm with the other spectral clustering algorithms, a variant of DI-SIM (Rohe, Qin, and Yu 2016), BI-SYM and its normalization version (DD-SYM) (Satuluri and Parthasarathy 2011), Herm (Cucuringu et al. 2020), and SimpleHerm (Laenen and Sun 2020).

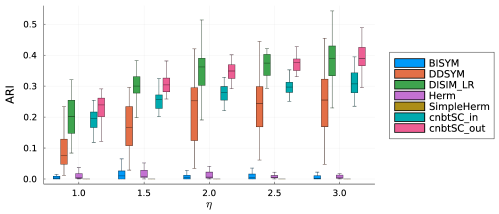
### 3.4.3 Experimental results

Figure 3.2 shows the result of the first experiment. While SimpleHerm exhibits good performance when $\eta$ is small, it is observed to be the most sensitive to the increase of $\eta$, with its performance degrading more rapidly than other methods. The proposed algorithm achieves competitive performance in all scenarios, and it shows the most superior results especially when $p$ is small.

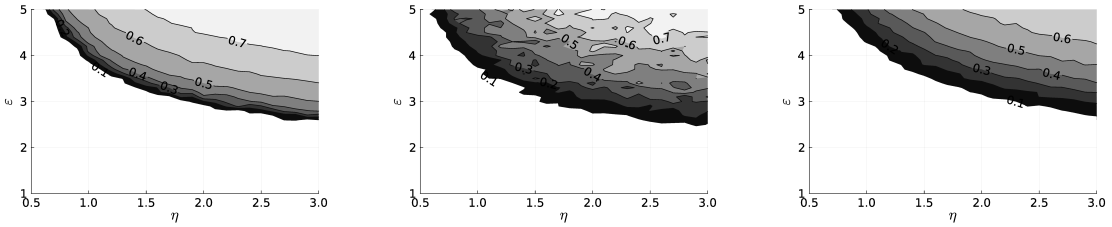
The results of the second experiment are shown in Figure 3.3. For most methods, we observe a general trend of improved performance as the correct circular pattern becomes more pronounced (i.e., with increasing $\epsilon$), and as the ratio of inter-cluster edges grows (i.e., which increasing $\eta$). It is noteworthy that the proposed method consistently achieves good results on both DSBM and DCSBM, while the performance of other methods deteriorates on either one of these models. The stable performance of DD-SYM across both models likely stems from its normalization process, which was specifically designed to handle the presence of hub vertices. Additionally, this result shows that SimpleHerm exhibits high variance in its performance under DSBM with sparse graphs. Further investigation of individual trials reveals that occasionally, the eigenvector used by SimpleHerm failed to identify any cluster structure, which



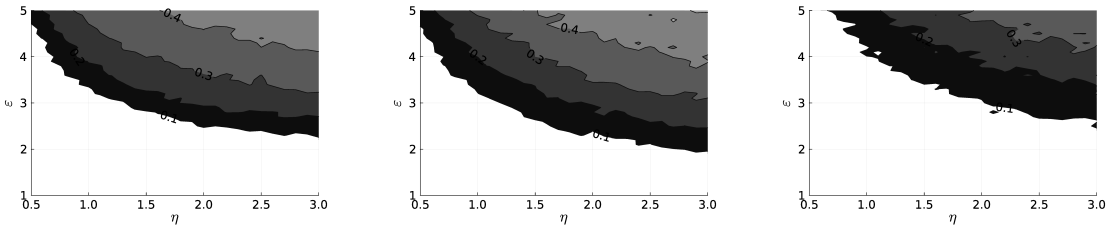

(a) DSBM with $\epsilon = 4$




(b) DCSBM with $\epsilon = 4$




(c) Contour plots over $\epsilon$ and $\eta$ under the DSBM, showing the top three methods from Figure 3.3a (from left to right: CNBT-SC (out), SimpleHerm, DD-SYM)




(d) Contour plots over $\epsilon$ and $\eta$ under the DCSBM, showing the top three methods from Figure 3.3b (from left to right: CNBT-SC (out), DI-SIM, DD-SYM)

Figure 3.3: Comparison of the second experiment. The top row shows boxplots with $\epsilon = 4$. Contour lines represent the average ARI over 30 independent trials.

consequently led to an ARI value of approximately zero.

### 3.4.4 Relationship with Belief Propagation

It has already been demonstrated in Decelle et al. (2011) and Krzakala et al. (2013) that the NBT matrix naturally emerges from the linearization of the belief propagation (Andersen 1991; Mezard and Montanari 2009) update equations. We investigate the idea to elucidate the relationship between the CNBT matrix and clustering.

Let us consider the stochastic block models with $K$ clusters. Let $\psi_u$ denotes the prior distribution indicating which cluster a vertex $u$ belongs to, $t_u$ be a random variable representing the cluster of the vertex $u$, and $P_{ab}$ be the probability of generating

a directed edge from a vertex in cluster $a$ to a vertex in cluster $b$. Then, the joint probability of $\{t_u\}_{u\in V}$ and $A$ can be described as follows:

$$f(\{t_u\}_{u\in V}, A) \propto \left(\prod_{u\in V} \psi_u(t_u)\right)\left(\prod_{u\neq v} P_{t_u t_v}^{A_{uv}} \left(1 - P_{t_u t_v}\right)^{1-A_{uv}}\right).$$

Let a network be sparse by defining the edge probability as $P_{ab} = \frac{c_{ab}}{N}$, where $c_{ab}$ is a constant and $N$ is the number of vertices. Then, as described in Decelle et al. (2011), the update equation for the message $\nu_{u\to v}$ from vertex $u$ to its neighboring vertex $v$ in loopy BP can be approximated as

$$\nu_{u\to v}(a) \propto \psi_u(a) e^{-h(a)} \left\{\prod_{w\in N_u\setminus\{v\}} \sum_b c_{ab}\nu_{w\to u}(b)\right\},$$

where $h(a) \coloneqq \frac{1}{N}\sum_{w\in V}\sum_b c_{ab}\nu_w(b)$ aggregates the influence from non-adjacent vertices, and $\nu_w(b)$ denotes the marginal probability that a vertex $w$ belongs to cluster $b$.

In order to clarify the relationship between the CNBT matrix and BP, we introduce the following assumptions: (i) each vertex shares the same prior and that is uniform, $\psi_u(a) = n_a = \frac{1}{K}$, and (ii) the vertices in each cluster have approximately the same degree, $c \coloneqq \sum_b c_{ab}$. Defining $\nu_{u\to v}(a) = n_a + \delta_{u\to v}(a)$ and linearizing around $n_a$ leads

$$\delta_{u\to v}(a) = \sum_{w\in N_u\setminus\{v\}} \sum_b \frac{c_{ab}}{Kc}\delta_{w\to u}(b).$$

Let $\frac{c_{ab}}{Kc}$ be the $(a,b)$ component of a matrix $T$. A vector $\boldsymbol{\delta} \in \mathbb{C}^{2mK}$ and a function $\mathrm{mat} : \mathbb{C}^{2mK} \to \mathbb{C}^{2m\times K}$ are defined as follows:

$$\boldsymbol{\delta} \coloneqq \begin{pmatrix}\delta_1(1) & \delta_2(1) & \dots & \delta_{2m-1}(K) & \delta_{2m}(K)\end{pmatrix},$$

$$\mathrm{mat}(\boldsymbol{\delta}) \coloneqq \begin{pmatrix}\delta_1(1) & \dots & \delta_1(K) \\ \vdots & \ddots & \\ \delta_{2m}(1) & & \delta_{2m}(K)\end{pmatrix} = \begin{pmatrix}\boldsymbol{\delta}_1 & \dots & \boldsymbol{\delta}_{2m}\end{pmatrix}^\top.$$

The function mat is a function that converts a vector into a matrix. This leads to

$$\boldsymbol{\delta} = (T \otimes B)\boldsymbol{\delta}. \tag{3.5}$$

We note that Equation (3.5) is equivalent to $\mathrm{mat}(\boldsymbol{\delta}) = B\mathrm{mat}(\boldsymbol{\delta})T^\top$.

Here, we consider the following two types of matrices $T_1, T_2 \in \mathbb{R}^{K\times K}$ with $e \in (0.5, 1), f, g \in \mathbb{R}_+, f > g$:

$$(T_1)_{ab} \coloneqq \begin{cases} e & a+1 \equiv b \bmod K \\ 1-e & b+1 \equiv a \bmod K \\ 0.5 & \text{otherwise,} \end{cases} \qquad (T_2)_{ab} \coloneqq \begin{cases} f & a+1 \equiv b \bmod K \\ g & \text{otherwise.} \end{cases} \tag{3.6}$$

These matrices model a scenario where directed edges are more likely to occur in a cyclic orientation, $1 \to 2 \to \cdots \to K \to 1$. The eigenvalues of $T_1, T_2$ are characterized by Lemma 3.5. The proof is presented in Appendix C.

**Lemma 3.5.** *The matrix $T_1$ has a single real eigenvalue and all remaining eigenvalues are purely imaginary. Similarly, the matrix $T_2$ has a single real eigenvalue, with the others denoted as $(f-g)\exp\left(\frac{2\pi\sqrt{-1}}{K}k\right)$, $k = 1, \ldots, K-1$.*

Suppose that we specifically choose $\boldsymbol{\delta}_i$ to be one of the eigenvectors of $T$, with $\mu_i \in \mathbb{C}$ as the corresponding eigenvalue, then

$$B\mathrm{mat}(\boldsymbol{\delta})T^\top = B\ \mathrm{diag}\begin{pmatrix}\mu_1 & \ldots & \mu_{2m}\end{pmatrix}\mathrm{mat}(\boldsymbol{\delta}).$$

In Equation (3.5), if we assume that the matrix $T$ is either $T_1$ or $T_2$, their eigenvalues include real multiples of the imaginary unit $\sqrt{-1}$ or $\exp(\frac{2\pi\sqrt{-1}}{K}k), k = 1, \ldots, K-1$. Consequently, depending on the choice of $\boldsymbol{\delta}_i$ and model parameters, $B\ \mathrm{diag}(\mu_1 \ \ldots \ \mu_{2m})$ and $B_\alpha = B\Lambda$ exhibit a structural similarity. This leads a connection between BP and the eigenvalue problem of the CNBT matrix.

## 3.5 Discussion

In this chapter, we explored a beneficial approach to the developing spectral analysis of directed graphs by introducing the CNBT matrix that combines the characteristics of

the Hermitian adjacency matrix and the NBT matrix. We confirmed the theoretical consistency of the matrix representation by establishing that the relationship holding between the adjacency matrix and the NBT matrix in undirected graphs also holds between the Hermitian adjacency matrix and the CNBT matrix. Numerical experiments showed that the CNBT matrix exhibits robust clustering performance, especially in sparse scenarios. In addition, by examining its connection to belief propagation, we gain insight into why the eigenvectors of the CNBT matrix are well suited for clustering.

## 3.6 Proof of Lemma 3.2

**Proof.** The value of $r_k$ is constructed from $P_{(k,r)}$. Therefore, we simultaneously derive these two quantities recursively.

**For $k = 1$:** from the definition of $P_{(k,r)}$, it is clear that $r_1 = A_\alpha$.

**For $k = 2, R = 3$:** The value $r_2$ is calculated from $P_{(2,0)}, P_{(2,1)}$ and $P_{(2,2)}$. The entry $(P_{(k,r)})_{uv}$ represents the number of non-backtracking mixed walks from vertex $u$ to vertex $v$ with length $k$ and rotation $r$. Thus, we first consider the total number of mixed walks, and then derive $P_{(k,r)}$ by subtracting the backtracking mixed walks.

A mixed walk of length 2 can be considered as an extension of a mixed walk of length 1 by adding an edge. A crucial point to consider is whether backtracking occurs when an edge is added. Mixed walks of length 2 and rotation 0 are composed of three types: (i) adding an edge with rotation 0 to a mixed walk of length 1 and rotation 0, (ii) adding an edge with rotation $-1$ to a mixed walk of length 1 and rotation 1, and (iii) adding an edge with rotation 1 to a mixed walk of length 1 and rotation $-1$. The total number of these mixed walks is represented by the underlined part in Equation (3.7).

We note that these mixed walks contain backtracking mixed walks when the start and end points coincide. This is because a mixed walk of length 2 consists of a sequence of edges that pass through three vertices. A backtracking mixed walk of length 2 that starts and ends at vertex $u$ can be classified into one of three categories: (a) traversing a bidirectional edge to another vertex $v$ and returning to $u$ via the same edge, (b) moving from $u$ to $v$ along an edge in the forward direction, then reversing and returning to $u$ via the same edge, or (c) traversing a directed edge from $v$ to $u$ in the reverse direction, then proceeding in the forward direction back to $u$ via the same edge.

We note that when vertices $u$ and $v$ are distinct, backtracking mixed walks of length 2 do not exist. Thus, the off-diagonal entries in the total number of backtracking mixed walks are zero. From these results, $P_{(2,0)}$ can be expressed as

$$P_{(2,0)} = \underline{P_{(1,0)}\overleftrightarrow{A} + P_{(1,2)}\overrightarrow{A} + P_{(1,1)}\overleftarrow{A}} - D. \tag{3.7}$$

For mixed walks of length 2 with rotation 1 or 2, backtracking mixed walks do not occur. Therefore, $P_{(2,1)}$ and $P_{(2,2)}$ are given as follows:

$$\begin{aligned} P_{(2,1)} &= P_{(1,1)}\overleftrightarrow{A} + P_{(1,0)}\overrightarrow{A} + P_{(1,2)}\overleftarrow{A}, \\ P_{(2,2)} &= P_{(1,2)}\overleftrightarrow{A} + P_{(1,1)}\overrightarrow{A} + P_{(1,0)}\overleftarrow{A}. \end{aligned}$$

As a result, the value $r_2 = P_{(2,0)} + \alpha P_{(2,1)} + \alpha^2 P_{(2,2)} = A_\alpha^2 - D$ holds.

**For $k = 2, R = 4$:** The value $r_2$ is calculated from $P_{(2,0)}, P_{(2,1)}, P_{(2,2)}$ and $P_{(2,3)}$. Each value can be derived similarly to the case of $R = 3$, and as a result, we obtain $r_2 = A_\alpha^2 - D$.

**For $k = 2, R > 4$:** It is obvious that $P_{(2,3)} = \cdots = P_{(2,R-3)} = \mathbf{0}$ since it is impossible to rotate more than twice in two steps. The reason we distinguish cases for $R$ when $k = 2$ is that such $r$ satisfying $P_{(2,r)} = \mathbf{0}$ exist when $2 < R - 2$. Thus, the value $r_2$ is calculated from $P_{(2,0)}, P_{(2,1)}, P_{(2,2)}, P_{(2,R-2)}$ and $P_{(2,R-1)}$. Each value can be derived similarly to the case of $R = 3$, leading to the following:

$$\begin{aligned} P_{(2,0)} &= P_{(1,0)}\overleftrightarrow{A} + P_{(1,R-1)}\overrightarrow{A} + P_{(1,1)}\overleftarrow{A} - D, \\ P_{(2,1)} &= P_{(1,0)}\overrightarrow{A} + P_{(1,1)}\overleftrightarrow{A}, \\ P_{(2,2)} &= P_{(1,1)}\overrightarrow{A}, \\ P_{(2,R-2)} &= P_{(1,R-1)}\overleftarrow{A}, \\ P_{(2,R-1)} &= P_{(1,0)}\overleftarrow{A} + P_{(1,R-1)}\overleftrightarrow{A}. \end{aligned}$$

This leads to $r_2 = \sum_{r=0}^{R-1} \alpha^r P_{(2,r)} = P_{(2,0)} + \alpha P_{(2,1)} + \alpha^2 P_{(2,2)} + \alpha^{R-2} P_{(2,R-2)} + \alpha^{R-1} P_{(2,R-1)} = A_\alpha^2 - D$.

**For $k \geq 3$:** we similarly count the number of mixed walks of length $k - 1$ extended by an edge, and then subtract the number of backtracking mixed walks. When considering $P_{(k,r)}$, there is a slight difference with $k = 2$ in the way backtracking mixed

walks are counted. The one-edge added mixed walks of length $k$ and rotation $r$ are expressed as $P_{(k-1,r)}\overleftrightarrow{A} + P_{(k-1,r-1)}\overrightarrow{A} + P_{(k-1,r+1)}\overleftarrow{A}$. This includes backtracking mixed walks, which are characterized by the absence of backtracking in the length $k-1$, but the addition of an edge induces backtracking. Namely, the backtracking mixed walks have the property that the vertex at length $k-2$ is the same as the vertex at length $k$. For each NBT mixed walk of length $k-2$ with rotation $r$, starting at vertex $u$ and ending at $v$, there exist $d_v - 1$ such backtracking mixed walks. The subtraction of 1 is necessary to prevent backtracking at length $k-1$. Thus,

$$P_{(k,r)} = P_{(k-1,r)}\overleftrightarrow{A} + P_{(k-1,r-1)}\overrightarrow{A} + P_{(k-1,r+1)}\overleftarrow{A} - P_{(k-2,r)}(D-I). \tag{3.8}$$

This result holds in $r = 0, \ldots, R-1$. Substituting Equation (3.8) into the definition of $r_k$ proves the lemma. □

## 3.7 Proof of Lemma 3.3

**Proof.** Since a NBT mixed walk in $(Q_{(k,r)})_{ef}$ can be represented by adding an oriented pair $h$ to a NBT mixed walk of length $k-1$, it can be described as follows:

$$\sum_h \left[ (\overleftrightarrow{B})_{eh}(Q_{(k-1,r)})_{hf} + (\overrightarrow{B})_{eh}(Q_{(k-1,r-1)})_{hf} + (\overleftarrow{B})_{eh}(Q_{(k-1,r+1)})_{hf} \right].$$

This does not contain backtracking mixed walks because added oriented pair $h$ can be connected to a NBT mixed walk of length $k-1$ while maintaining NBT due to the definition of $Q$. □

## 3.8 Proof of Corollary 3.1

**Proof.** We prove Corollary 3.1 by induction. This corollary clearly holds for $k = 1$. Suppose that this corollary holds when $k = k' \geq 1$. Then, we will show $(B_\alpha)^{k'+1} =$

$\sum_{r=0}^{R-1} \alpha^r Q_{(k'+1,r)}$. From lemma 3.3,

$$\sum_{r=0}^{R-1} \alpha^r Q_{(k'+1,r)} = \overleftrightarrow{B} \sum_{r=0}^{R-1} \alpha^r Q_{(k',r)} + \alpha \overrightarrow{B} \sum_{r=0}^{R-1} \alpha^{r-1} Q_{(k',r-1)} + \alpha^{-1} \overleftarrow{B} \sum_{r=0}^{R-1} \alpha^{r+1} Q_{(k',r+1)},$$

note that $\alpha^{-1} = \alpha^{R-1} = \bar{\alpha},\ \alpha^R = \alpha^0 = 1,\ Q_{(k',-1)} = Q_{(k',R-1)},\ Q_{(k',R)} = Q_{(k',0)}$

$$= (\overleftrightarrow{B} + \alpha \overrightarrow{B} + \bar{\alpha} \overleftarrow{B}) \sum_{r=0}^{R-1} \alpha^r Q_{(k',r)} = (B_\alpha)^{k'+1}.$$

By induction, this result proves Corollary 3.1. □

## 3.9 Proof of Corollary 3.2

**Proof.** Our proof follows the general outline of the proof for Ihara's formula described in Stark and Terras (1996). We show that both sides are equal to the reciprocal of $z_X(u;\alpha)$. Since Konno et al. (2019, Theorem 3) shows that the right-hand side is equal to the reciprocal to the zeta function, we prove $z_X(u;\alpha)^{-1} = \det(I - uB_\alpha)$.

Applying the Maclaurin expansion to $z_X(u;\alpha)$,

$$\log z_X(u;\alpha) = -\sum_{\substack{[C] \\ \text{primitive} \\ \text{NBT, no tail}}} \log\left(1 - \alpha^{r(C)} u^{|C|}\right) = \sum_{\substack{[C] \\ \text{primitive} \\ \text{NBT, no tail}}} \sum_{k \geq 1} \frac{\alpha^{kr(C)}}{k} u^{k|C|}.$$

Let $\sharp[C]$ denote the number of elements contained in the equivalence class $[C]$ of cycle $C$. Its value matches the length of the cycle. Moreover, since changing the starting vertex of the cycle does not effect the direction of the edges constituting the cycle, the rotation number also remains the same. Thus, for $C_1, C_2 \in [C],\ r(C_1) = r(C_2)$ and $\sharp[C] = |C|,\ kr(C) = r(C^k), k\,|C| = |C^k|$ hold. Thus,

$$u\frac{d}{du} \log z_X(u;\alpha) = \sum_{\substack{[C] \\ \text{primitive} \\ \text{NBT, no tail}}} \sum_{k \geq 1} |C|\, \alpha^{kr(C)} u^{k|C|} = \sum_{\substack{C \\ \text{NBT, no tail}}} \alpha^{r(C)} u^{|C|}.$$

Note that when partitioning all NBT cycles with no tail by length and rotation, the

NBT cycles within the same group will have the same $r(C)$ and $|C|$. Thus,

$$u\frac{d}{du}\log z_X(u;\alpha) = \sum_{k\geq 1}\sum_{r=0}^{R-1}\alpha^r n_{(k,r)}u^k. \tag{3.9}$$

With respect to $\det(I - uB_\alpha)$, the definition of matrix logarithm and $\exp[\mathrm{tr}\,(M)] = \det e^M$ for a matrix $M$ derive:

$$\log\det(I - uB_\alpha) = -\sum_{k\geq 1}\frac{u^k}{k}\mathrm{tr}\left(B_\alpha^k\right).$$

Applying corollary 3.1 implies

$$u\frac{d}{du}\log\det\left(I - uB_\alpha\right) = -\sum_{k\geq 1}\sum_{r=0}^{R-1}\alpha^r\left(\sum_{e\in\bar{E}}\left(Q_{(k,r)}\right)_{ee}\right)u^k.$$

The sum $\sum_{e\in\bar{E}}\left(Q_{(k,r)}\right)_{ee}$ counts the number of NBT cycles. For each cycle, the initial and terminal vertex is $t_e$, the last edge is $e$, and the initial edge can be connected to the trailing edge $e$ while maintaining NBT. This means the cycles don't have a tail and $\sum_{e\in\bar{E}}\left(Q_{(k,r)}\right)_{ee} = n_{(k,r)}$. From Equation (3.9),

$$\begin{aligned} u\frac{d}{du}\log\det\left(I - uB_\alpha\right) &= u\frac{d}{du}\log z_X(u;\alpha)^{-1}, \\ \therefore\ z_X(u;\alpha)^{-1} &= \det\left(I - uB_\alpha\right). \end{aligned}$$

□

## 3.10 Proof of Theorem 3.4

**Proof.** Note that $((B_\alpha \boldsymbol{g})_\alpha^{\mathrm{out}})_u$ can be written as follows based on the definition of Equation (3.3).

$$\left((B_\alpha \boldsymbol{g})_\alpha^{\mathrm{out}}\right)_u = \sum_{v\in\overleftrightarrow{N_u}}(B_\alpha \boldsymbol{g})_{\overrightarrow{uv}} + \alpha\sum_{v\in\overrightarrow{N_u}}(B_\alpha \boldsymbol{g})_{\overrightarrow{uv}} + \bar{\alpha}\sum_{v\in\overleftarrow{N_u}}(B_\alpha \boldsymbol{g})_{\overrightarrow{uv}}.$$

We show that each term in the above equation can be expressed as follows:

$$\sum_{v\in\overleftrightarrow{N_u}} (B_\alpha \boldsymbol{g})_{\overrightarrow{uv}} = \sum_{v\in\overleftrightarrow{N_u}} \left(\boldsymbol{g}_\alpha^{\text{out}}\right)_v - \sum_{v\in\overleftrightarrow{N_u}} g_{\overrightarrow{vu}}, \tag{3.10}$$

$$\alpha \sum_{v\in\overrightarrow{N_u}} (B_\alpha \boldsymbol{g})_{\overrightarrow{uv}} = \alpha \sum_{v\in\overrightarrow{N_u}} \left(g_\alpha^{\text{out}}\right)_v - \sum_{v\in\overrightarrow{N_u}} g_{\overrightarrow{vu}}, \tag{3.11}$$

$$\bar{\alpha} \sum_{v\in\overleftarrow{N_u}} (B_\alpha \boldsymbol{g})_{\overrightarrow{uv}} = \bar{\alpha} \sum_{v\in\overleftarrow{N_u}} \left(g_\alpha^{\text{out}}\right)_v - \sum_{v\in\overleftarrow{N_u}} g_{\overrightarrow{vu}}. \tag{3.12}$$

First, we organize $(B_\alpha \boldsymbol{g})_{\overrightarrow{uv}}$ used in these three transformations. From $B_\alpha = \overleftrightarrow{B} + \alpha\overrightarrow{B} + \bar{\alpha}\overleftarrow{B}$,

$$\begin{aligned}(B_\alpha \boldsymbol{g})_{\overrightarrow{uv}} &= \sum_{\overrightarrow{xy}\in\bar{E}} (B_\alpha)_{\overrightarrow{uv},\overrightarrow{xy}}\, g_{\overrightarrow{xy}} \\ &= \sum_{\overrightarrow{xy}\in\bar{E}} (\overleftrightarrow{B})_{\overrightarrow{uv},\overrightarrow{xy}}\, g_{\overrightarrow{xy}} + \alpha \sum_{\overrightarrow{xy}\in\bar{E}} (\overrightarrow{B})_{\overrightarrow{uv},\overrightarrow{xy}}\, g_{\overrightarrow{xy}} + \bar{\alpha} \sum_{\overrightarrow{xy}\in\bar{E}} (\overleftarrow{B})_{\overrightarrow{uv},\overrightarrow{xy}}\, g_{\overrightarrow{xy}}\end{aligned}$$

Here, $(\overleftrightarrow{B})_{\overrightarrow{uv},\overrightarrow{xy}}$, $(\overrightarrow{B})_{\overrightarrow{uv},\overrightarrow{xy}}$, and $(\overleftarrow{B})_{\overrightarrow{uv},\overrightarrow{xy}}$ are determined to be 0 or 1 by $\delta_{v,x}(1-\delta_{u,y})$ and $x \leftrightarrow_G y,\ x \rightarrow_G y,\ x \leftarrow_G y$. These mean that $x = v$ and $y \neq u$, so

$$(B_\alpha \boldsymbol{g})_{\overrightarrow{uv}} = \sum_{\substack{\overrightarrow{vy}\in\bar{E}\\ y\neq u\\ v\leftrightarrow_G y}} g_{\overrightarrow{vy}} + \alpha \sum_{\substack{\overrightarrow{vy}\in\bar{E}\\ y\neq u\\ v\rightarrow_G y}} g_{\overrightarrow{vy}} + \bar{\alpha} \sum_{\substack{\overrightarrow{vy}\in\bar{E}\\ y\neq u\\ v\leftarrow_G y}} g_{\overrightarrow{vy}} = \sum_{\substack{w\in V\setminus u\\ v\leftrightarrow_G w}} g_{\overrightarrow{vw}} + \alpha \sum_{\substack{w\in V\setminus u\\ v\rightarrow_G w}} g_{\overrightarrow{vw}} + \bar{\alpha} \sum_{\substack{w\in V\setminus u\\ v\leftarrow_G w}} g_{\overrightarrow{vw}}.$$

Substituting this result into the left-hand side of Equation (3.10),

$$\sum_{v\in\overleftrightarrow{N_u}}(B_\alpha \boldsymbol{g})_{\overrightarrow{uv}} = \sum_{v\in\overleftrightarrow{N_u}}\left(\sum_{\substack{w\in V\setminus u\\ v\leftrightarrow_G w}} g_{\overrightarrow{vw}} + \alpha \sum_{\substack{w\in V\setminus u\\ v\rightarrow_G w}} g_{\overrightarrow{vw}} + \bar{\alpha}\sum_{\substack{w\in V\setminus u\\ v\leftarrow_G w}} g_{\overrightarrow{vw}}\right)$$

note that for $u, v$ satisfying $u \leftrightarrow_G v$, the conditions $v \rightarrow_G u$ $v \leftarrow_G u$ do not hold,

$$= \sum_{v\in\overleftrightarrow{N_u}}\left\{\left(\sum_{\substack{w\in V\\ v\leftrightarrow_G w}} g_{\overrightarrow{vw}}\right) - g_{\overrightarrow{vu}} + \alpha \sum_{\substack{w\in V\\ v\rightarrow_G w}} g_{\overrightarrow{vw}} + \bar{\alpha}\sum_{\substack{w\in V\\ v\leftarrow_G w}} g_{\overrightarrow{vw}}\right\}$$

$$= \sum_{v\in\overleftrightarrow{N_u}}(\boldsymbol{g}_\alpha^{\mathrm{out}})_v - \sum_{v\in\overleftrightarrow{N_u}} g_{\overrightarrow{vu}}.$$

This proves Equation (3.10). Equations (3.11) and (3.12) can be proven using similar steps.

From Equations (3.10), (3.11), and (3.12), we can compute $((B_\alpha \boldsymbol{g})_\alpha^{\mathrm{out}})_u = (A_\alpha \boldsymbol{g}_\alpha^{\mathrm{out}})_u - (\boldsymbol{g}_\alpha^{\mathrm{in}})_u$. Namely,

$$((B_\alpha \boldsymbol{g})_\alpha^{\mathrm{out}})_u = \left\{\sum_{v\in\overleftrightarrow{N_u}}\left(\boldsymbol{g}_\alpha^{\mathrm{out}}\right)_v + \alpha\sum_{v\in\overrightarrow{N_u}}\left(\boldsymbol{g}_\alpha^{\mathrm{out}}\right)_v + \bar{\alpha}\sum_{v\in\overleftarrow{N_u}}\left(\boldsymbol{g}_\alpha^{\mathrm{out}}\right)_v\right\} - \left(\sum_{v\in\overleftrightarrow{N_u}} g_{\overrightarrow{vu}} + \sum_{v\in\overrightarrow{N_u}} g_{\overrightarrow{vu}} + \sum_{v\in\overleftarrow{N_u}} g_{\overrightarrow{vu}}\right)$$

$$= (A_\alpha \boldsymbol{g}_\alpha^{\mathrm{out}})_u - (\boldsymbol{g}_\alpha^{\mathrm{in}})_u.$$

Similarly, $((B_\alpha \boldsymbol{g})_\alpha^{\mathrm{in}})_v$ can be expressed according to the definition of Equation (3.2)

as follows, and each term can be calculated as shown below.

$$\left((B_\alpha \boldsymbol{g})_\alpha^{\text{in}}\right)_v = \sum_{u\in\overleftrightarrow{N_v}} (B_\alpha \boldsymbol{g})_{\overrightarrow{uv}} + \sum_{u\in\overleftarrow{N_v}} (B_\alpha \boldsymbol{g})_{\overrightarrow{uv}} + \sum_{u\in\overrightarrow{N_v}} (B_\alpha \boldsymbol{g})_{\overrightarrow{uv}},$$

$$\begin{cases} \sum_{u\in\overleftrightarrow{N_v}} (B_\alpha \boldsymbol{g})_{\overrightarrow{uv}} = \sum_{u\in\overleftrightarrow{N_v}} \left(g_\alpha^{\text{out}}\right)_v - \sum_{u\in\overleftrightarrow{N_v}} g_{\overrightarrow{vu}}, \\ \sum_{u\in\overleftarrow{N_v}} (B_\alpha \boldsymbol{g})_{\overrightarrow{uv}} = \sum_{u\in\overleftarrow{N_v}} \left(g_\alpha^{\text{out}}\right)_v - \bar{\alpha} \sum_{u\in\overleftarrow{N_v}} g_{\overrightarrow{vu}}, \\ \sum_{u\in\overrightarrow{N_v}} (B_\alpha \boldsymbol{g})_{\overrightarrow{uv}} = \sum_{u\in\overrightarrow{N_v}} \left(g_\alpha^{\text{out}}\right)_v - \alpha \sum_{u\in\overrightarrow{N_v}} g_{\overrightarrow{vu}}. \end{cases}$$

This leads to $((B_\alpha \boldsymbol{g})_\alpha^{\text{in}})_v = (d_v - 1)(\boldsymbol{g}_\alpha^{\text{out}})_v$. From these results, we obtain the theorem. □

## 3.11 Derivation of Equation (3.5)

**Proof.** For the standard BP notation, we define a factor function between vertices $u$ and $v$ as $\psi_{uv}(a,b) \coloneqq P_{ab}^{A_{uv}}(1-P_{ab})^{1-A_{uv}}$, and let $\psi_u$ denote the prior distribution of cluster assignments for vertex $u$. Using this notation, the update equation of a message $\nu_{u\to v}$ and the marginal distribution $\nu_u$ can be expressed as

$$\nu_{u\to v}(t_u) \propto \psi_u(t_u) \prod_{w\in V\setminus\{u,v\}} \sum_{t_w} \psi_{uw}(t_u,t_w)\nu_{w\to u}(t_w),$$
$$\nu_u(t_u) \propto \psi_u(t_u) \prod_{w\in V\setminus\{u\}} \sum_{t_w} \psi_{uw}(t_u,t_w)\nu_{w\to u}(t_w).$$

With the normalization condition $\sum_b \nu_{w\to u}(b) = 1$ and $P_{ab} = \frac{c_{ab}}{N}$, the update equation can be rewritten by

$$\nu_{u\to v}(t_u) \propto \psi_u(t_u) \left\{ \prod_{w\in N_u\setminus\{v\}} \sum_{t_w} c_{t_u t_w}\nu_{w\to u}(t_w) \right\} \left\{ \prod_{\substack{w\in V\setminus N_u \\ w\neq u,v}} \left(1 - \frac{1}{N}\sum_{t_w} c_{t_u t_w}\nu_{w\to u}(t_w)\right) \right\}.$$

Considering whether there exists an edge between vertex $u$ and vertex $v$, we apply different approximations accordingly.

When there does not exist an edge from vertex $u$ to vertex $v$,

$$\nu_{u\to v}(t_u) \propto \frac{\nu_u(t_u)}{1-\frac{1}{N}\sum_{t_v} c_{t_u t_v}\nu_{v\to u}(t_v)} \xrightarrow{N\to\infty} \nu_u(t_u).$$

Let us consider the case where an edge exists. Since the graph is sparse, the degree of each vertex $u$ is $O(1)$, which implies that $\frac{1}{N}\sum_{w\in N_u\cup\{u\}}\sum_{t_w} c_{t_u t_w}\nu_w(t_w) \xrightarrow{N\to\infty} 0$ holds. Furthermore, by analogy with $\lim_{n\to\infty}(1+\frac{x}{n})^n = e^x$, the following approximation holds.

$$\nu_{u\to v}(t_u) \propto \psi_u(t_u)e^{-h(t_u)} \prod_{w\in N_u\setminus\{v\}} \sum_{t_w} c_{t_u t_w}\nu_{w\to u}(t_w),$$
$$\text{where} \quad h(a) \coloneqq \frac{1}{N}\sum_{w\in V}\sum_{b} c_{ab}\nu_w(b).$$

The term $h(a)$ is said to be an auxiliary external field and summarizes the influence from non-adjacent vertices.

Let $\delta_{u\to v}$ be the perturbation from the prior distribution $n_a$, and express the value of the message as $\nu_{u\to v}(a) = n_a + \delta_{u\to v}(a)$. Then,

$$\frac{\nu_{u\to v}(a)}{\nu_{u\to v}(b)} = \frac{n_a e^{-h(a)}\prod_{w\in N_u\setminus\{v\}}\sum_d c_{ad}\nu_{w\to u}(d)}{n_b e^{-h(b)}\prod_{w\in N_u\setminus\{v\}}\sum_d c_{bd}\nu_{w\to u}(d)}$$
$$\Rightarrow \frac{n_a+\delta_{u\to v}(a)}{n_b+\delta_{u\to v}(b)} = \frac{n_a}{n_b}e^{-\{h(a)-h(b)\}}\prod_{w\in N_u\setminus\{v\}}\frac{\sum_d c_{ad}\,(n_d+\delta_{w\to u}(d))}{\sum_d c_{bd}\,(n_d+\delta_{w\to u}(d))}$$
$$\Rightarrow \log\frac{n_a+\delta_{u\to v}(a)}{n_b+\delta_{u\to v}(b)} = \log\frac{n_a}{n_b} - \{h(a)-h(b)\} + \sum_{w\in N_u\setminus\{v\}}\log\frac{\sum_d c_{ad}\,(n_d+\delta_{w\to u}(d))}{\sum_d c_{bd}\,(n_d+\delta_{w\to u}(d))}$$

approximating by $\log(C+x)\approx\log C+\frac{x}{C}$,

$$\Rightarrow \frac{\delta_{u\to v}(a)}{n_a} - \frac{\delta_{u\to v}(b)}{n_b} = -\{h(a)-h(b)\} + \sum_{w\in N_u\setminus\{v\}}\sum_d \frac{c_{ad}-c_{bd}}{c}\delta_{w\to u}(d). \tag{3.13}$$

We consider the approximation of $h(a)$ and $h(b)$ as $N \to \infty$.

$$h(a) = \sum_d c_{ad} \frac{\sum_{w \in V} \nu_w(d)}{N}.$$

$$h(a) \to \frac{\sum_d c_{ad}}{K} = \frac{c}{K}.$$

Thus, the term $h(a) - h(b)$ in Equation (3.13) can be approximated to 0. Substituting this into Equation (3.13) yields

$$\frac{\delta_{u \to v}(a)}{n_a} - \frac{\delta_{u \to v}(b)}{n_b} = \sum_{w \in N_u \setminus \{v\}} \sum_d \frac{c_{ad} - c_{bd}}{c} \delta_{w \to u}(d). \tag{3.14}$$

Here, since $n_a = n_b = 1/K$ and Equation (3.14) holds for any $a \neq b$, summing over $b \neq a$ gives

$$\begin{aligned} K\left(\sum_{b \neq a} \delta_{u \to v}(a) - \delta_{u \to v}(b)\right) &= \sum_{w \in N_u \setminus \{v\}} \sum_d \delta_{w \to u}(d) \frac{1}{c} \sum_{b \neq a} (c_{ad} - c_{bd}) \\ &= \sum_{w \in N_u \setminus \{v\}} \sum_d \delta_{w \to u}(d) \frac{1}{c} \left(K c_{ad} - \sum_b c_{bd}\right) \\ &= \sum_{w \in N_u \setminus \{v\}} \sum_d \frac{K c_{ad}}{c} \delta_{w \to u}(d) - \sum_{w \in N_u \setminus \{v\}} \sum_d \frac{\sum_b c_{bd}}{c} \delta_{w \to u}(d). \end{aligned}$$

In general, messages are normalized to 1, i.e., $\sum_d \nu_{u \to v}(d) = 1$, which implies $\sum_d \delta_{u \to v}(d) = 0$. Using this along with $c = \sum_b c_{bd}$, we obtain

$$\sum_{b \neq a} \delta_{u \to v}(a) - \delta_{u \to v}(b) = K \delta_{u \to v}(a) - \sum_b \delta_{u \to v}(b) = \sum_{w \in N_u \setminus \{v\}} \sum_d \frac{c_{ad}}{c} \delta_{w \to u}(d)$$

$$\therefore \delta_{u \to v}(a) = \sum_{w \in N_u \setminus \{v\}} \sum_d \frac{c_{ad}}{Kc} \delta_{w \to u}(d)$$

□

## 3.12 Proof of Lemma 3.5

**Proof.** Since the matrices $T_1, T_2$ are circulant, let $p_0 = \frac{1}{2},\ p_1 = e,\ p_2 = \cdots = p_{K-2} = \frac{1}{2},\ p_{K-1} = 1 - e$ and $q_0 = g,\ q_1 = f,\ q_2 = \cdots = q_{K-1} = g$, then $(T_1)_{ab} = p_{(b-a) \bmod K}$ and $(T_2)_{ab} = q_{(b-a) \bmod K}$ holds. Furthermore, we denote the eigenvalues of $T_1, T_2$ by $\lambda_k, \mu_k, k = 0, \ldots, K-1$ respectively. A property of circulant matrices (Gray 2005) leads

$$\lambda_k = \sum_{l=0}^{K-1} p_l \exp\left(\frac{2\pi\sqrt{-1}}{K} kl\right), \qquad \mu_k = \sum_{l=0}^{K-1} q_l \exp\left(\frac{2\pi\sqrt{-1}}{K} kl\right).$$

When $k = 0$, $\lambda_0 = \frac{K}{2}$ and $\mu_0 = f + (K-1)g$ hold. For the case where $k \neq 0$, $\lambda_k$ and $\mu_k$ can be calculated as follows:

$$\lambda_k = \sqrt{-1}(2e-1)\sin\left(\frac{2\pi}{K}k\right), \qquad \mu_k = (f-g)\exp\left(\frac{2\pi\sqrt{-1}}{K}k\right).$$

□

# 4

# Acceleration of Categorical Wasserstein Weisfeiler-Lehman Graph Kernel

## 4.1 Background

To apply machine learning frameworks to real-world data that can be represented as graphs, a wide variety of graph kernels have been proposed. Classical approaches began with methods known as R-convolution kernels, which compare substructures of graphs and sum their similarities. More recently, methods that consider both local and global structures of graphs have been developed. In particular, methods that utilize optimal transport have gained attention, as they consider distributional information of substructures that classical graph kernels often overlook, thereby enhancing discriminative power.

On the other hand, it is well known that solving optimal transport problems is computationally expensive, and numerous methods have been proposed specifically to accelerate the computation of the Wasserstein distance. There are two main acceleration

strategies for the computation of Wasserstein distance: (i) the sliced Wasserstein distance (SWD) (Bonneel et al. 2015), which relies on one-dimensional projections for supports of input measures, and exploits the closed-form expression of univariate OT; and (ii) an entropic regularization approach (Cuturi 2013), which can be solved efficiently by the Sinkhorn algorithm (Sinkhorn 1964). While the Sinkhorn algorithm is a method for acceleration, it still requires a computational cost proportional to the square of the number of vertices. Thus, it becomes impractical for large-scale graphs. To further accelerate the computation, various methods have been proposed, such as a greedy strategy (Altschuler, Niles-Weed, and Rigollet 2017), sparsification of the kernel matrix (M. Li et al. 2023), and low-rank factorization of the coupling (Scetbon, Cuturi, and Peyré 2021). Unfortunately, these acceleration methods do not address the increase in the number of graphs in a dataset. This is because computing a graph kernel requires calculating the Wasserstein distance for all pairs of graphs in the dataset, which requires a computational cost proportional to the square of the number of graphs.

### 4.1.1 Contributions

In this chapter, we reveal that the categorical Wasserstein Weisfeiler-Lehman (WWL) graph kernel (Togninalli et al. 2019) can achieve significant computational cost improvements by leveraging its latent structure. The contribution of this work is threefold:

1. We explicitly reveal that the relationship among the WL labels forms a tree structure.

2. We prove that the proposed algorithm, *Tree Wasserstein Weisfeiler-Lehman (TWWL) algorithm*, can exactly compute the Wasserstein distance of the categorical WWL graph kernel, which is designed for graph datasets with categorical node labels. This is achieved without entropic regularization.

3. We show that our algorithm is scalable with respect to both the number of vertices in graphs and the number of graphs in a dataset. As a result, the TWWL algorithm makes it feasible to apply the WWL graph kernel to large-scale graph datasets that existing algorithms cannot handle.

## 4.2 Related Work

Our objective is to accelerate the categorical WWL graph kernel computation. In this section, we provide a brief introduction to the background relevant to our proposal.

### 4.2.1 Tree Wasserstein Distance

The Tree Wasserstein distance is the $L^1$-Wasserstein distance when the underlying metric space is a tree. C. Lozupone and Knight (2005) implicitly utilized this concept by proposing UniFrac, a measure based on phylogenetic tree branch lengths to quantify the differences between two microbial communities. Evans and Matsen (2012) later formalized rigorously that the weighted UniFrac (C. A. Lozupone et al. 2007) can be generalized as the $L^1$-Wasserstein distance on a tree, providing a foundation and generalization within the optimal transport framework.

Let $\mathcal{T} = (V(\mathcal{T}), E(\mathcal{T}))$ be a tree with a weight function $w : E(\mathcal{T}) \to \mathbb{R}_{\geq 0}$, $d_{\mathcal{T}}$ be a path metric on $\mathcal{T}$, and $\mu, \nu$ be probability measures supported on $V(\mathcal{T})$. Throughout this chapter, we assume that $V(\mathcal{T})$ is finite, and we work on the measurable space $(V(\mathcal{T}), 2^{V(\mathcal{T})})$, where $2^{V(\mathcal{T})}$ is the power set of $V(\mathcal{T})$. From Definition 2.5, the Tree Wasserstein distance $W_{d_{\mathcal{T}}}$ is defined as the $L^1$-Wasserstein distance on $\mathcal{T}$:

$$W_{d_{\mathcal{T}}}(\mu, \nu) \coloneqq \inf_{\pi \in \Pi(\mu,\nu)} \int_{V(\mathcal{T}) \times V(\mathcal{T})} d_{\mathcal{T}}(x, y) d\pi(x, y),$$

where $\pi$ is a coupling, i.e., a joint probability measure on $V(\mathcal{T}) \times V(\mathcal{T})$ with marginals $\mu$ and $\nu$. The set of all such couplings is denoted by $\Pi(\mu, \nu)$. The following result by Evans and Matsen (2012) shows that the Tree Wasserstein distance has a closed-form expression.

**Theorem 4.1** ((Evans and Matsen 2012)). *The Tree Wasserstein distance can be written as follows:*

$$W_{d_{\mathcal{T}}}(\mu, \nu) = \sum_{e \in E(\mathcal{T})} w_e \left| \mu(\Gamma(v_e)) - \nu(\Gamma(v_e)) \right|,$$

*where for each edge $e \in E(\mathcal{T})$, $v_e$ denotes the vertex closer to the leaves among the two vertices that constitute edge $e$. $\Gamma(v)$ denotes the set of vertices in the subtree rooted at*


Figure 4.1: An example where the WL algorithm fails to discriminate non-isomorphic graphs.

*vertex* $v \in V(\mathcal{T})$.

By using a post-order traversal, $\mu(\Gamma(v_e))$ and $\nu(\Gamma(v_e))$ can be computed efficiently. Thus, Theorem 4.1 indicates that the computational complexity of the Tree Wasserstein distance is linear in the number of edges, i.e., the number of vertices in the tree. Considering the special case where the tree degenerates into a path, the formula for the Tree Wasserstein distance coincides with the well-known closed-form expression for the one-dimensional $L^1$-Wasserstein distance. This means that the Tree Wasserstein distance is a natural extension of the one-dimensional case.

### 4.2.2 Weisfeiler-Lehman algorithm

The Weisfeiler-Lehman (WL) algorithm is designed for the graph isomorphism problem. While counterexamples exist where the WL test fails to distinguish non-isomorphic graphs (see Figure 4.1), it is known to work effectively for a wide range of graphs. Recently, it has also been used to evaluate the expressive power of graph neural networks. We define graph isomorphism as follows.

**Definition 4.2** (isomorphism (Diestel 2024))**.** *Let* $\mathcal{G}_1 = (V_1, E_1)$ *and* $\mathcal{G}_2 = (V_2, E_2)$ *be two graphs. It is said that* $\mathcal{G}_1$ *and* $\mathcal{G}_2$ *are isomorphic if there exists a bijection* $\varphi : V_1 \longrightarrow V_2$ *such that for any* $u, v \in V_1$, $\{u, v\} \in E_1 \Leftrightarrow \{\varphi(u), \varphi(v)\} \in E_2$.

*When two graphs are labeled by* $l_0$*, it is also required that the labels of the vertices correspond, i.e., for any* $v \in V_1$, $l_0(v) = l_0(\varphi(v))$ *holds.*

The WL algorithm iteratively assigns a new label to each vertex, generating a multiset of vertex labels for the graph $\mathcal{G} = (V, E)$. The isomorphism of two graphs is

**Algorithm 7:** Weisfeiler-Lehman algorithm

**Input:** $\mathcal{G} = (V, E)$: undirected graph with initial vertex labels $l_0 : V \to \Sigma_0$; $H$: the maximum number of iterations

**Output:** multiset of vertex labels at the $H$-th iteration

```
for h ← 1, ..., H do
    for v ∈ V do
        l_h(v) ← hash(l_{h−1}(v), N_{v,h−1})
    end
end
return {{l_H(v) | v ∈ V}}
```

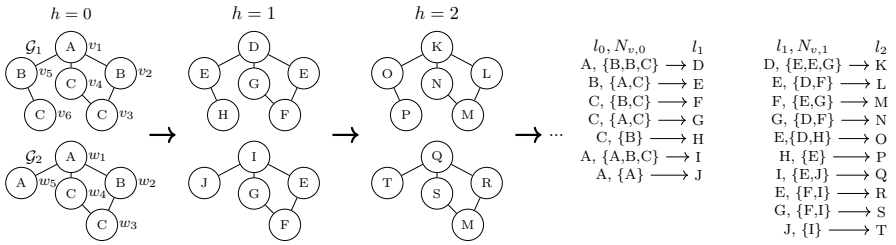



Figure 4.2: An example of the WL algorithm for given graphs with $\Sigma_0 = \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$.

determined by whether the multiset of vertex labels of one graph is identical to that of the other graph. Note that if a graph does not have initial node labels, the WL algorithm can still be applied by assigning the same dummy label to all vertices. At the $h$-th iteration, the new label for a vertex $v \in V$ is generated as follows:

$$l_h(v) \coloneqq \text{hash}\left(l_{h-1}(v), \mathcal{N}_{v,h-1}\right), \tag{4.1}$$

where hash is a hash function commonly used in computer science that produces the same output for identical inputs and different outputs for different inputs with high probability, and $\mathcal{N}_{v,h-1} \coloneqq \{\{l_{h-1}(u) \mid u \in \mathcal{N}_v\}\}$ is the multiset of labels of $\mathcal{N}_v$ at the $(h-1)$-th iteration.

Algorithm 7 presents the pseudocode for the WL algorithm. Let $WL_H(\mathcal{G})$ denote the multiset of vertex labels of $\mathcal{G}$ obtained after $H$ iterations. Then, the WL algorithm determines the isomorphism of two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ by checking whether $WL_H(\mathcal{G}_1)$ and $WL_H(\mathcal{G}_2)$ are identical.

Figure 4.2 illustrates the iterative process of the WL algorithm on labeled graphs

$\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2\}$. In this figure, the new labels generated at each step are represented by Latin characters that are not used in the previous steps, reflecting the injectivity of the hash function. Vertices $v_2$ and $v_5$ start with identical labels and the same multiset of neighbor labels, meaning that they remain indistinguishable after the first iteration. However, as structural information from further neighborhoods is propagated, their local contexts diverge. This divergence leads to them being assigned distinct labels at $h = 2$. This example demonstrates how the WL algorithm effectively propagates structural information to eventually differentiate vertices that initially appear identical from a local viewpoint.

### 4.2.3 Hash function in WL algorithm

In this section, we define the spaces handled by the hash function in Equation (4.1). In the WL algorithm and our problem setting, we assume that node labels are categorical values for which direct comparison or arithmetic operations across labels are not meaningful. Typical examples include chemical elements such as “hydrogen”, “carbon”, and “oxygen”, as well as research fields such as “high energy physics”, “condensed matter physics”, and “astrophysics”. Consequently, replacing each label with an arbitrary symbol does not affect the problem. For this reason, we represent labels as bit strings of length $b$.

The properties required of a hash function depend on its intended use. For example, in cryptography (Menezes, Oorschot, and Vanstone 1996), preimage resistance, second preimage resistance, and collision resistance are central. On the other hand, in algorithmic settings (Carter and Wegman 1979), determinism and low collision rate are emphasized. The hash function used in the WL algorithm and our proposed algorithm falls into the latter category.

Regarding the domain of the hash function in Equation (4.1), let $U \coloneqq \{0,1\}^b$ be the set of all binary strings of length $b$, and assume that the initial label set is a subset of this space, i.e., $\Sigma_0 \subset U$. Generally, for a set $S \subset U$, let $\mathcal{M}(S)$ denote the collection of all multisets formed from elements of $S$. Then, the function $\mathrm{hash} : U \times \mathcal{M}(U) \to U$, is assumed to be deterministic, and efficiently computable, in the sense that it always returns the same output for the same input, and can be evaluated efficiently. We further assume that it is collision-resistant in the algorithmic sense, meaning that the

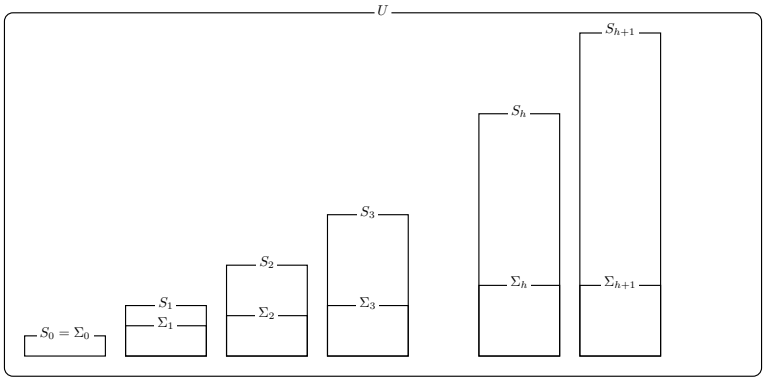



Figure 4.3: Relationship among $U$ and $\{S_h, \Sigma_h\}$.

probability of two distinct inputs producing the same hash value is extremely low.

Let $S \subset U$, and denote by $\mathrm{hash}|_S$ the restriction of hash to the domain $S \times \mathcal{M}(S)$. Using this notation, we define the set of labels that can be generated potentially as follows:

$$S_0 \coloneqq \Sigma_0, \qquad S_h \coloneqq \mathrm{Im}\left(\mathrm{hash}|_{S_{h-1}}\right), \quad h = 1, 2, \ldots.$$

In addition, when applying the WL algorithm to the collection of graphs $\mathcal{G} \coloneqq (\mathcal{G}_1, \ldots, \mathcal{G}_N)$, we denote by $\Sigma_h$ the set of labels that are actually generated at the $h$-th iteration. The potentially generable label set $S_h$ and the actually generated label set $\Sigma_h$ satisfy $\Sigma_h \subset S_h \subset U$. Moreover, when hash can be regarded as injective, the label space $U$ and the sets $\{S_h, \Sigma_h\}$ are related as illustrated in Figure 4.3. We note that the size of the actually generated label set $\Sigma_h$ grows at most linearly. While the size of the potentially generable label set $S_h$ can grow exponentially, as it depends on the number of distinct multiset patterns that may arise when producing new hash values in Equation (4.1), $\Sigma_h$ contains only one label per vertex at each iteration. Consequently, the growth of $\Sigma_h$ is at most linear, and its cardinality is upper bounded by the number of vertices. Therefore, although $S_h$ may collide with previous sets $S_1, \ldots, S_{h-1}$, collisions among the labels in $\Sigma_0, \ldots, \Sigma_h$ can be regarded as negligible in practice.

When Equation (4.1) is applied to actual data and is injective, Lemma 4.3 immediately follows from the definition of the hash function.

**Lemma 4.3.** *For every $\sigma_h \in \Sigma_h$, there exists a unique label sequence $(\sigma_0, \ldots, \sigma_{h-1})$, $\sigma_i \in \Sigma_i$*

*that generates* $\sigma_h$.

For implementing the hash function in Equation (4.1), the built-in hash function in Python or Julia is sufficient. When using this function, however, one must ensure that the result is invariant to the ordering of elements in the multiset, for example by sorting the multiset before hashing. If one wishes to construct a hash function that does not require sorting, one possible approach is to combine a standard fixed-length hash function $h : \{0,1\}^* \to U$ that maps an arbitrary finite-length bit string to a $b$-bit string (e.g., SHA-256) with a binary operator $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ that is commutative and associative. A simple example of such an operator is $f(a,b) \coloneqq a + b$. Together with the left-shift bitwise operator « and the bitwise OR operator $|$, one can define hash as follows:

$$\text{hash}(s, \{\{s_1, \ldots, s_m\}\}) \coloneqq h\left(\{f\left(\ldots f\left(f\left(h(s_1), h(s_2)\right), h(s_3)\right), \ldots, h(s_m)\right) \ll b\} \mid h(s)\right).$$

### 4.2.4 Weisfeiler-Lehman subtree Kernel

The WL subtree kernel, introduced by Shervashidze et al. (2011), is a graph kernel that leverages the vertex labels generated by the WL algorithm. Let $c(\sigma, \mathcal{G})$ denotes the count of a WL label $\sigma \in \bigcup_{h=0}^{H} \Sigma_h$ in $\mathcal{G}$. Then, the WL subtree feature for $\mathcal{G}$ is defined as

$$\phi(\mathcal{G}) \coloneqq \Big(c(\sigma_1, \mathcal{G}), \ldots, c(\sigma_L, \mathcal{G})\Big), \quad \text{where } \bigcup_{h=0}^{H} \Sigma_h = \{\sigma_1, \ldots, \sigma_L\}.$$

We use $\langle \cdot, \cdot \rangle$ to denote an inner product, then the WL subtree kernel on two graphs $\mathcal{G}_1, \mathcal{G}_2$ is defined as

$$k_{\text{WLsubtree}}(\mathcal{G}_1, \mathcal{G}_2) \coloneqq \langle \phi(\mathcal{G}_1), \phi(\mathcal{G}_2) \rangle.$$

The WL subtree kernel is positive semidefinite and can be computed in $O(Hm)$, where $m$ is the total number of edges in $\mathcal{G}_1$ and $\mathcal{G}_2$.

### 4.2.5 Wasserstein Weisfeiler-Lehman Graph Kernel

Many classical graph kernels measure the similarity of graphs by simply aggregating their substructure information. These methods essentially reduce graph information to first-order statistics, such as averages, discarding distributional information like pattern co-occurrences and attribute variations. To incorporate such information into similarity measures, Togninalli et al. (2019) proposed the Wasserstein Weisfeiler-Lehman (WWL) graph kernel. The WWL kernel regards the multiset of vertex labels generated by the WL algorithm as empirical distributions in the label sequence space, and measures the similarity between two graphs using optimal transport. In Togninalli et al. (2019), two types of WWL graph kernels were proposed: one for graphs with discrete node labels and the other for graphs with continuous node attributes. In this dissertation, we focus on the WWL graph kernel for graphs with discrete labels, which we refer to as the *categorical WWL graph kernel*.

Given two undirected labeled graphs $\mathcal{G}_1 = (V_1, E_1)$ and $\mathcal{G}_2 = (V_2, E_2)$ with initial vertex labels $l_0$, let $\{(l_0(v), \ldots, l_H(v)) \mid v \in V_1\}$ and $\{(l_0(w), \ldots, l_H(w)) \mid w \in V_2\}$ be the sets of label sequences generated for each vertex of the graphs after $H$ iterations of the WL algorithm, respectively. In the WWL graph kernel, the distance $d_{\mathrm{Ham}}$ on $\Sigma_H$ is defined as the Hamming distance:

$$d_{\mathrm{Ham}}(\sigma_H, \tau_H) \coloneqq H + 1 - \sum_{h=0}^{H} \delta_{\sigma_h, \tau_h} \quad \sigma_H, \tau_H \in \Sigma_H,$$

where $\delta_{\cdot,\cdot}$ is the Kronecker delta, and $(\sigma_0, \ldots, \sigma_{H-1})$ and $(\tau_0, \ldots, \tau_{H-1})$ are the unique label sequences that generate $\sigma_H$ and $\tau_H$, respectively, as guaranteed by Lemma 4.3. By Lemma 4.3, $d_{\mathrm{Ham}}$ is a valid metric on $\Sigma_H$. Let $\mu \coloneqq \frac{1}{|V_1|} \sum_{v \in V_1} \delta_{l_H(v)}$ and $\nu \coloneqq \frac{1}{|V_2|} \sum_{w \in V_2} \delta_{l_H(w)}$ be the empirical distributions on $\Sigma_H$ corresponding to $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively. Here, $\delta_{\cdot}$ denotes the Dirac delta function. Then, the Wasserstein distance between $\mathcal{G}_1$ and $\mathcal{G}_2$ is defined as follows:

$$W(\mathcal{G}_1, \mathcal{G}_2) \coloneqq W_{d_{\mathrm{Ham}}}(\mu, \nu) = \inf_{\pi \in \Pi(\mu, \nu)} \int_{\Sigma_H^2} d_{\mathrm{Ham}}(\sigma, \tau) d\pi(\sigma, \tau), \tag{4.2}$$

and the WWL graph kernel is defined by

$$k_W(\mathcal{G}_1, \mathcal{G}_2) \coloneqq e^{-\lambda W(\mathcal{G}_1, \mathcal{G}_2)}, \tag{4.3}$$

where $\lambda$ is a positive hyperparameter. Togninalli et al. (2019) proved that the WWL graph kernel is positive semidefinite, and employed either the network simplex algorithm (Orlin 1997) to compute the Wasserstein distance or the Sinkhorn algorithm (Cuturi 2013; Sinkhorn 1964) to approximate it.

## 4.3 Proposed Algorithm

Recent graph kernels are often designed to have high discriminative power and therefore tend to be computationally expensive. As discussed in the previous section, the WWL graph kernel is no exception. In this section, we develop an algorithm, called the Tree Wasserstein Weisfeiler-Lehman (TWWL) algorithm, that efficiently and accurately computes the Wasserstein distance for the categorical WWL graph kernel by exploiting the latent hierarchical structure of WL label sequences. This allows the categorical WWL graph kernel to be applied to large-scale graph datasets. Such datasets were previously impractical for conventional approaches based on linear programming solvers or the Sinkhorn algorithm.

### 4.3.1 Tree Structure of WL Labels

In our algorithm, we focus on the relationship between two WL labels $l_h(v)$ and $l_{h-1}(v)$. The fact that $l_h(v)$ is generated from $l_{h-1}(v)$ forms a kind of parent-child relationship. We first define this as follows:

**Definition 4.4** (Parent-Child relationship among WL labels)**.** *For $\sigma \in \Sigma_h$ and $\tau \in \Sigma_{h+1}$, if there exists a multiset $S \in \mathcal{M}(\Sigma_h)$ that satisfies $\tau = hash(\sigma, S)$, then we call that $\sigma$ is a parent label of $\tau$, or equivalently, $\tau$ is a child label of $\sigma$.*

For WL label sequences associated with vertices, the following is immediate from the properties of the hash function.

**Lemma 4.5.** *Let $u, v$ be arbitrary vertices in $\mathcal{V}$. If there exists $h_0 \leq H$ such that $l_{h_0}(u) \neq l_{h_0}(v)$, $^\forall h \in \{h_0, \ldots, H\}$, $l_h(u) \neq l_h(v)$ holds.*

**Algorithm 8:** Tree Structure for TWWL

**Input:** $\mathcal{G}$: a set of graphs with the initial labels $l_0$; $H$: the maximum number of the WL algorithm

**Output:** Label tree $\mathcal{T}$ rooted at `root`, and mapping $\phi : \Sigma_H \to V(\mathcal{T})$

```
Struct NODE
    depth: INTEGER
    Γ: VECTOR OF N DOUBLES
    children: HASHMAP<LABEL,NODE>
end

Function GETORCREATECHILD(node, label)
    if HAS(node.children, label) then
        return node.children[label]
    end
    child ⟵ NODE(depth=node.depth+1, label=label, Γ=0_N, children=[])
    INSERT(node.children, key=label, value=child)
    return child
end

root ⟵ NODE(depth=-1, label="", Γ=0_N, children=[])
que ⟵ EMPTY FIFO QUEUE⟨node,graph id,vertex⟩()
for G_i in G do
    for v in V_i do
        child ⟵ GETORCREATECHILD(root, l_0(v))
        ENQUEUE(que, (child,i,v))
    end
end
while !ISEMPTY(que) do
    (node,i,v) ⟵ DEQUEUE(que)
    node.Γ_i ⟵ node.Γ_i + 1/n_i
    if node.depth < H then
        h ⟵ node.depth
        child ⟵ GETORCREATECHILD(node,l_{h+1}(v))
        ENQUEUE(que, (child,i,v))
    end
end
return root
```

The TWWL algorithm constructs a tree structure of WL labels based on the parent-child relationship,

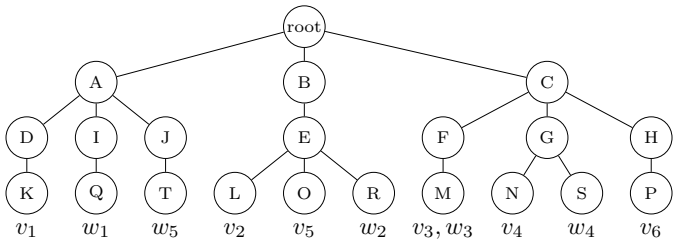



Figure 4.4: An illustration of a label tree based on graphs in Figure 4.2

**Definition 4.6** (Label Tree)**.** *We refer to the relationship among WL labels, organized to satisfy the following four properties, as a* Label Tree*. This structure is constructed by Algorithm 8.*

1. *Except for the root, tree nodes at depth $h$ correspond one-to-one with labels in $\Sigma_{h-1}$.*
   - *Labels in $\Sigma_H$ correspond one-to-one with leaf nodes.*
   - *Let $\phi : \Sigma_H \to V(\mathcal{T})$ denote this correspondence.*
2. *For every label in $\Sigma_0$, there is an edge connecting it to the root.*
3. *There exists an edge between any two labels that satisfy the relationship in Definition 4.4.*
4. *Each edge is assigned a weight of $1/2$.*
   - *Let $d_{\mathcal{T}}(a, b) \coloneqq \frac{|\mathcal{P}(a,b)|}{2}$ denote the path metric between two nodes $a, b \in V(\mathcal{T})$.*

Figure 4.4 illustrates the label tree constructed from the example in Figure 4.2. We show that the label tree satisfying these properties is indeed a tree in the following lemma.

**Lemma 4.7.** *A label tree defined in Definition 4.6 is a tree.*

We now discuss the computational complexity of Algorithm 8. In lines 9–17, each dequeue operation can trigger at most one enqueue operation, during which the depth is incremented by one. Hence, the total number of operations in this part is bounded by the product of the initial queue size and the maximum depth $H$. Since the initial queue elements are inserted one per iteration in lines 3–8, their number is $\sum_{i=1}^{N} n_i = n$,

where $n_i$ is the number of vertices in graph $\mathcal{G}_i$ and $n$ is the total number of vertices across all graphs. Moreover, the average time complexity of search and insertion in hash map is $O(1)$. Therefore, the computational complexity for constructing the label tree is $O(Hn)$, which is linear in the total number of vertices.

### 4.3.2 Tree Wasserstein Distance on WL Labels

This section shows that the distance structure of WL label sequences defined as the Hamming distance can be embedded into a metric space on the label tree. Furthermore, it leads to the result that the Wasserstein distance for the categorical WWL graph kernel can be reduced to the Tree Wasserstein distance on the label tree.

Lemma 4.8 shows that the path metric on the label tree coincides with the Hamming distance on $\Sigma_H$.

**Lemma 4.8.** *The mapping $\phi : \Sigma_H \to V(\mathcal{T})$ defined in Definition 4.6 is distance-preserving.*

The above results imply that the Wasserstein distance for categorical WWL graph kernel can be computed efficiently as follows.

**Proposition 4.1.** *Suppose that $\mu, \nu$ are probability measures supported on $\Sigma_H$. Let $(\mathcal{T}, d_{\mathcal{T}}, \phi)$ denote a label tree, path metric, and distance-preserving mapping from $\Sigma_H$ to $V(\mathcal{T})$ defined in Definition 4.6, respectively. Then, the Wasserstein distance in Equation (4.2) can be written as*

$$W_{d_{Ham}}(\mu, \nu) = \sum_{e \in E(\mathcal{T})} \frac{1}{2} \left| \phi_\sharp \mu(\Gamma(v_e)) - \phi_\sharp \nu(\Gamma(v_e)) \right|, \tag{4.4}$$

*where $\phi_\sharp \mu$ is the push-forward measure of $\mu$ by $\phi$, defined for Borel sets $S$ of $\mathcal{T}$ as $\phi_\sharp \mu(S) \coloneqq \mu(\phi^{-1}(S))$.*

Figure 4.5 illustrates the push-forward measures based on the example in Figure 4.2. For instance, considering the node $\phi_C$ in the label tree corresponding to label C, we have $\Gamma(\phi_C)$ as the set of vertices in the subtree rooted at node $\phi_C$. Thus, $\phi_\sharp \mu(\Gamma(\phi_C)) = 1/2$ and $\phi_\sharp \nu(\Gamma(\phi_C)) = 2/5$.

The positive definiteness proof of the categorical WWL kernel in the original WWL paper already hints at a hierarchical structure of the Wasserstein computation.

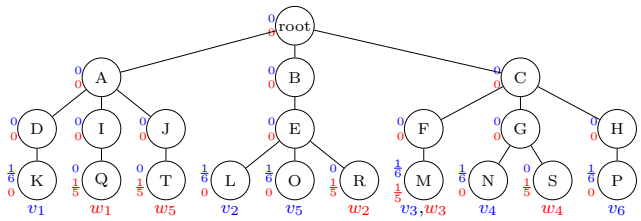



Figure 4.5: An illustration of push-forward measures based on Figure 4.2. The measure $\phi_\sharp\mu$ corresponding to $\mathcal{G}_1$ is shown in blue, and $\phi_\sharp\nu$ corresponding to $\mathcal{G}_2$ is shown in red. For a node $\phi_\sigma$, the values displayed on the left represent $\phi_\sharp\mu(\{\phi_\sigma\})$, while those on the right represent $\phi_\sharp\nu(\{\phi_\sigma\})$.

The proof in Togninalli et al. (2019) shows that, under a Hamming ground cost, the Wasserstein distance between WL embeddings can decompose across the depth of the WL iterations, and the optimal coupling at the final layer induces optimal couplings at earlier layers. In this sense, a hierarchical structure is implicitly present. Proposition 4.1 makes this structure explicit by identifying the label tree. By treating the WL embeddings as probability measures on the tree, we show that applying Theorem 4.1 derives a closed-form expression for the Wasserstein distance.

This result shows that the TWWL algorithm exhibits lower time complexity than the approach described in Togninalli et al. (2019). Given that $\mathcal{G}_1$ and $\mathcal{G}_2$ are labeled graphs with $n_1$ and $n_2$ vertices, respectively. The time complexity of the Sinkhorn algorithm for Equation (4.2) requires $O(n_1 n_2)$. The time complexity of the TWWL algorithm is determined by the number of edges in the label tree and the computation of $\Gamma$. The number of edges is $O(n_1 + n_2)$ even in the worst case, and $\Gamma$ can be computed in $O(n_1 + n_2)$ time beforehand. Consequently, the overall time complexity of our algorithm is $O(n_1 + n_2)$. As the number of vertices in the given graphs increases, our algorithm becomes significantly faster than the existing approach.

Equation (4.3) has been shown to be positive definite for every $\lambda > 0$ by Togninalli et al. (2019). The alternative expression based on the Tree Wasserstein distance offers an independent verification. From Le et al. (2019, Proposition 2), it follows that $W_{d_{\text{Ham}}}(\mu, \nu)$, expressed in Equation (4.4), is negative definite. Furthermore, Berg, Christensen, and Ressel (1984, Theorem 2.2) proves that the graph kernel given in Equation (4.3) is positive definite.

## 4.4 Experiments

In this section, we conduct two numerical experiments to validate the scalability and the performances of our approach.

We benchmark our algorithm against the Wasserstein Weisfeiler-Lehman kernel with the Sinkhorn algorithm (Togninalli et al. 2019, WWL), the Weisfeiler-Lehman subtree kernel (Shervashidze et al. 2011, WL), the Weisfeiler-Lehman Optimal Assignment kernel (Kriege, Giscard, and Wilson 2016, WL-OA), traditional kernels such as the Shortest Path kernel (Borgwardt and Kriegel 2005, SP), and Graphlet sampling kernel (Pržulj 2007, GL), and recent methods such as the Multiscale Laplacian graph kernel (Kondor and Pan 2016, MLG), the Weisfeiler-Lehman Lower Bound distance (S. Chen et al. 2022, WLLB)[1], and the Relaxed Weisfeiler-Lehman kernel (Schulz et al. 2022, R-WL)[2]. For WL, WL-OA, SP, GL and MLG, we utilize the graph kernel implementations provided by Siglidis et al. (2020). All experiments are conducted on an Ubuntu 22.04 machine equipped with an Intel Xeon Gold 6354 CPU and 256GB of RAM. For runtime comparisons against WWL, both our method and the WWL kernel are implemented entirely in Julia to ensure fairness.

We evaluate the proposed methods on graph classification task using real-world datasets summarized in Table 4.1. For datasets without node labels, we assign degree-based dummy labels. Across baselines, node labels are used as follows. WL, WL-OA and WWL use them as inputs to the WL algorithm. R-WL represents WL labels as unfolding trees and assess their similarity via a tree edit metric. SP incorporates them when comparing two shortest paths by taking into account the agreement of node labels at the endpoints of corresponding edges along the paths. In WLLB, node labels instantiate the ground label space from which the hierarchy of probability measures is built. MLG uses node label agreement as the base kernel over vertex features. GL ignores node labels and measures similarity via counts of graphlet patterns. All datasets are downloaded from Morris et al. (2020).

[1] https://github.com/chens5/WL-distance
[2] https://github.com/mlai-bonn/GenWL

Table 4.1: Summary of datasets. We denote by $N$ the number of graphs in the dataset, and by $\bar{n}$ and $\text{avg}\{|E_i|\}$ the average number of vertices and edges per graph, respectively.

| | $N$ | $\bar{n}$ | $\text{avg}\{\|E_i\|\}$ | ♯graphs per class | node labels |
|---|---|---|---|---|---|
| MUTAG | 188 | 17.93 | 19.79 | 63/125 | ✓ |
| PTC-MR | 344 | 14.29 | 14.69 | 192/152 | ✓ |
| ENZYMES | 600 | 32.63 | 62.14 | 100/100/100/100/100/100 | ✓ |
| PROTEINS | 1113 | 39.06 | 72.82 | 663/450 | ✓ |
| DD | 1178 | 284.32 | 715.66 | 691/487 | ✓ |
| NCI1 | 4110 | 29.87 | 32.30 | 2053/2057 | ✓ |
| COLLAB | 5000 | 74.49 | 2457.78 | 2600/775/1625 | - |
| REDDIT-BINARY | 2000 | 429.62 | 497.75 | 1000/1000 | - |
| REDDIT-MULTI-5K | 4999 | 508.51 | 594.87 | 1000/1000/1000/1000/999 | - |
| REDDIT-MULTI-12K | 11929 | 391.40 | 456.89 | 767/2592/1000/1094/902/1205 513/999/1243/1092/522 | - |
| DBLP-v1 | 19456 | 10.48 | 19.65 | 9530/9926 | ✓ |
| github-stargazers | 12725 | 113.79 | 234.64 | 5917/6808 | - |

### 4.4.1 Runtime Comparison

In the first experiment, we benchmark the computation time required for calculating the pairwise Wasserstein distances on real-world datasets. For each combination of datasets and methods, we conduct 10 independent trials and report the average runtime (in seconds) and the corresponding standard deviation. We denote linear programming solver by LP, implemented by Huangfu and Hall (2018), and note that our algorithm provides the same calculated values with the solver. Furthermore, since the computation time of the Sinkhorn algorithm varies depending on the entropic regularization term $\gamma$, we assess the time for the Sinkhorn algorithm with both 0.01 and 1.0. The parameter $H$ for the WL algorithm is fixed at 5 for simplicity. We note that the runtime in Table 4.2 includes only the time taken to compute the pairwise distance matrix, excluding the preprocessing time for LP, WWL, and TWWL. This is because Algorithm 8 used in TWWL as preprocessing is a slight modification of the WL algorithm used for preprocessing in LP and WWL, and its running time is comparable to, or at most slightly slower than that of the WL algorithm.

Table 4.2 shows that TWWL significantly reduces runtime compared to either the Sinkhorn algorithm or linear programming solver. Notably, TWWL completes computations within practical time limits even in challenging scenarios, such as a dataset with numerous graphs (e.g., DBLP-v1), a dataset with high average vertex

Table 4.2: Runtime performance (in seconds) of the Wasserstein distance computation fixed at $H = 5$. Entries labeled "n/a" indicate that a single trial exceeded 24 hours in runtime.

| METHOD | MUTAG | PTC-MR | ENZYMES | PROTEINS | DD | NCI1 |
|---|---|---|---|---|---|---|
| LP | 33.89 ± 1.68 | 85.98 ± 5.73 | 882.56 ± 69.52 | 4,597.12 ± 157.77 | n/a | 34,775.82 ± 337.00 |
| WWL (0.01) | 3.15 ± 0.04 | 4.54 ± 0.01 | 37.01 ± 0.02 | 147.26 ± 0.52 | 9,390.61 ± 28.49 | 2,116.45 ± 6.92 |
| WWL (1.00) | 0.27 ± 0.01 | 0.47 ± 0.00 | 5.33 ± 0.16 | 24.89 ± 0.23 | 1,283.43 ± 9.48 | 229.97 ± 0.59 |
| TWWL | 0.03 ± 0.00 | 0.08 ± 0.00 | 0.67 ± 0.12 | 2.38 ± 0.19 | 26.93 ± 0.40 | 24.33 ± 0.56 |

| | COLLAB | **REDDIT-B** | **REDDIT-M5** | **REDDIT-M12** | DBLP-v1 | **github-stargazers** |
|---|---|---|---|---|---|---|
| LP | n/a | n/a | n/a | n/a | n/a | n/a |
| WWL (0.01) | 7,721.67 ± 37.48 | n/a | n/a | n/a | 1,798.11 ± 5.92 | n/a |
| WWL (1.00) | 1,653.68 ± 27.69 | 8,457.94 ± 469.58 | n/a | n/a | 945.74 ± 8.84 | 25,820.89 ± 65.90 |
| TWWL | 44.18 ± 0.32 | 38.92 ± 0.32 | 333.53 ± 3.02 | 5,535.76 ± 25.07 | 328.89 ± 0.56 | 952.02 ± 2.11 |

counts per graph (e.g., REDDIT-MULTI-5K), and a dataset with both features (e.g., REDDIT-MULTI-12K). For some datasets, such as NCI1, DD, and larger datasets, it was frequently observed that the Sinkhorn algorithm failed to converge within the maximum number of iterations for any value of $\gamma$. A key advantage of our algorithmic approach is its reliability to produce stable solutions.

### 4.4.2 Performance Comparison

In this experiment, we evaluate the performance of TWWL on a graph classification task using 10-fold cross-validation with a support vector machine classifier (Cortes and Vapnik 1995). All hyperparameters are selected via grid search on the training folds only: the SVM regularization parameter, the number of WL iterations $H$, the kernel scale $\lambda$, and the Sinkhorn entropic regularization term in WWL. Since the TWWL algorithm is specifically designed to accurately and efficiently compute the Wasserstein distance, it is expected that our algorithm achieves comparable performance to the WWL kernel. Our aim is to verify that the precise Wasserstein distance does not adversely affect performance when compared to the Sinkhorn algorithm.

Table 4.3 shows that our algorithm achieves classification accuracy competitive with the WWL kernel. Notably, TWWL can scale to large datasets for which the Sinkhorn algorithm is computationally prohibited due to the runtime and/or memory requirements. This result suggests that our proposed approach can serve as a drop-in replacement for the Sinkhorn algorithm in the WWL kernel in the context of categorical

Table 4.3: Classification accuracies on datasets. Entries labeled "OOM (out-of-memory)" indicate that the computation failed with an error that requested more than 256GB of memory.

| METHOD | MUTAG | PTC-MR | ENZYMES | PROTEINS | DD | NCI1 |
|---|---|---|---|---|---|---|
| SP | 81.84 ± 8.62 | 61.00 ± 6.75 | 40.67 ± 6.54 | **75.74 ± 2.75** | **79.80 ± 2.37** | 73.26 ± 1.95 |
| GL | 77.02 ± 9.70 | 55.82 ± 4.15 | 29.83 ± 5.24 | 68.92 ± 3.45 | 72.49 ± 4.52 | 59.61 ± 2.35 |
| WL | 85.00 ± 10.51 | 62.27 ± 11.90 | 54.33 ± 6.95 | 75.65 ± 2.86 | 79.20 ± 3.66 | 85.18 ± 1.51 |
| WL-OA | 85.56 ± 6.92 | 62.24 ± 5.88 | **60.67 ± 5.62** | 75.21 ± 2.58 | 79.46 ± 2.29 | 86.16 ± 1.35 |
| WWL | 87.72 ± 6.91 | 65.12 ± 7.27 | 60.00 ± 5.50 | 74.58 ± 2.54 | n/a | 86.37 ± 1.40 |
| MLG | **88.30 ± 9.21** | 57.82 ± 7.55 | 60.17 ± 6.73 | 75.65 ± 2.55 | n/a | 80.88 ± 2.42 |
| R-WL | 87.69 ± 8.07 | 54.38 ± 7.88 | 45.33 ± 5.65 | 74.67 ± 3.89 | n/a | 78.73 ± 1.34 |
| WLLB | 88.27 ± 8.29 | 53.19 ± 8.82 | 37.83 ± 6.76 | n/a | n/a | n/a |
| TWWL | **88.27 ± 6.16** | **66.00 ± 6.83** | 59.83 ± 4.74 | 74.94 ± 2.98 | 77.17 ± 3.51 | **86.57 ± 1.27** |

| | COLLAB | REDDIT-B | REDDIT-M5 | REDDIT-M12 | DBLP-v1 | github-stargazers |
|---|---|---|---|---|---|---|
| SP | **81.34 ± 2.47** | **88.70 ± 1.93** | 52.33 ± 2.62 | **44.10 ± 1.14** | OOM | **68.06 ± 1.17** |
| GL | n/a | n/a | n/a | n/a | n/a | n/a |
| WL | 78.74 ± 2.51 | 75.05 ± 2.07 | 50.79 ± 1.46 | 39.65 ± 1.33 | 92.97 ± 0.52 | 65.19 ± 1.03 |
| WL-OA | n/a | 88.60 ± 2.33 | n/a | n/a | n/a | n/a |
| WWL | n/a | n/a | n/a | n/a | OOM | n/a |
| MLG | n/a | n/a | n/a | OOM | OOM | n/a |
| R-WL | n/a | n/a | n/a | n/a | OOM | n/a |
| WLLB | n/a | n/a | n/a | n/a | n/a | n/a |
| TWWL | 80.28 ± 1.67 | 85.75 ± 2.87 | **54.47 ± 2.48** | 42.90 ± 0.95 | **93.37 ± 0.47** | 65.55 ± 0.73 |

node labels.

## 4.5 Discussion

In this chapter, we address the problem that recently proposed graph kernels with high discriminative power are difficult to apply to large-scale datasets due to their high computational cost. To this end, we focus on the categorical Wasserstein Weisfeiler-Lehman graph kernel and show that the Weisfeiler-Lehman label sequences and their Hamming distances used in the WWL graph kernel can be efficiently computed by leveraging the underlying tree structure. As a result, the developed algorithm has a computational cost that scales linearly with the number of vertices in the graphs and can compute the Wasserstein distance exactly without using entropic regularization. This significantly expands the range of datasets to which the categorical WWL graph kernel can be applied, enabling the provision of the graph kernel with high discriminative power even for large-scale datasets.

## 4.6 Proof of Lemma 4.7

**Proof.** From property 1 in Definition 4.6, nodes in the label tree, except for the root, correspond to labels in $\bigcup_{h=0}^{H} \Sigma_h$. Thus, we represent nodes in the label tree by their corresponding labels.

We begin by proving the connectivity of the label tree, i.e., that there exists a walk between any label $\sigma \in \bigcup_{h=0}^{H} \Sigma_h$ and the root. Lemma 4.3 implies that there exists a label sequence $(\sigma_0, \ldots, \sigma_{h-1}, \sigma)$ that generates $\sigma$. Properties 2 and 3 ensure the existence of edges between the root and $\sigma_0$, and between adjacent nodes in the label sequence, respectively. Thus, a walk exists between any label and the root. This implies that a walk exists between any two labels via the root.

Next, we show that the label tree contains no cycle. Suppose, for the sake of contradiction, that there exists a cycle $(\sigma, \tau_1, \ldots, \tau_k, \sigma)$ in the label tree that consists of distinct nodes except for the start and end points. Without loss of generality, assume that $\sigma$ is the deepest node in the cycle. Then, both $\tau_1$ and $\tau_k$ must be parent nodes of $\sigma$. By a property of the hash function, $\tau_1 = \tau_k$ holds. However, this equality contradicts the assumption, which requires that the intermediate vertices are distinct. Therefore, no cycle exists in the label tree. □

## 4.7 Proof of Lemma 4.8

**Proof.** We denote $\phi(\sigma)$ simply as $\phi_\sigma$. We show that for any $\sigma_H, \tau_H \in \Sigma_H$, $d_{\mathrm{Ham}}(\sigma_H, \tau_H) = d_{(\mathcal{T},1/2)}(\phi_{\sigma_H}, \phi_{\tau_H})$ holds, where $d_{(\mathcal{T},1/2)}$ denotes the distance on the label tree with all edge weights equal to $1/2$. Lemma 4.3 implies that there exist label sequences $(\sigma_0, \ldots, \sigma_H)$, $(\tau_0, \ldots, \tau_H)$ that satisfy the parent-child relationship. From Lemma 4.5, we consider the following three cases:

Case 1: For any $h \in \{0, \ldots, H\}$, $\sigma_h = \tau_h$. Then, since the two label sequences are identical, we have $d_{\mathrm{Ham}}(\sigma_H, \tau_H) = 0$. Furthermore, since $\phi_{\sigma_H} = \phi_{\tau_H}$ point to the same node in the label tree, we have $d_{(\mathcal{T},1/2)}(\phi_{\sigma_H}, \phi_{\tau_H}) = 0$.

Case 2: $\sigma_0 \neq \tau_0$. Then, it follows from Lemma 4.5 that $d_{\mathrm{Ham}}(\sigma_H, \tau_H) = H + 1$. Additionally, Lemma 4.7 implies that the sequence $(\phi_{\sigma_H}, \ldots, \phi_{\sigma_0}, \mathrm{root}, \phi_{\tau_0}, \ldots, \phi_{\tau_H})$ is a unique path between $\phi_{\sigma_H}$ and $\phi_{\tau_H}$. Thus, $d_{(\mathcal{T},1/2)}(\phi_{\sigma_H}, \phi_{\tau_H}) = H + 1$.

Case 3: There exists $h_0 \in \{1, \ldots, H\}$, $\sigma_{h_0-1} = \tau_{h_0-1}$ and $\sigma_{h_0} \neq \tau_{h_0}$. Then, by

Lemma 4.5, $d_{\mathrm{Ham}}(\sigma_H, \tau_H) = H+1-h_0$ is satisfied and the sequence $(\phi_{\sigma_H}, \ldots, \phi_{\sigma_{h_0}}, \phi_{\sigma_{h_0-1}} = \phi_{\tau_{h_0-1}}, \phi_{\sigma_{h_0}}, \ldots, \phi_{\sigma_H})$ is a unique path between $\phi_{\sigma_H}$ and $\phi_{\tau_H}$. This implies $d_{(\mathcal{T},1/2)}(\phi_{\sigma_H}, \phi_{\tau_H}) = H + 1 - h_0$.

Therefore, Lemma 4.8 holds for all cases. □

## 4.8 Proof of Proposition 4.1

We first present lemmas necessary for the proof of Proposition 4.1.

**Lemma 4.9.** *Let $(X, \mathcal{A}, \mu)$ be a measure space, let $(Y, \mathcal{B})$ and $(Z, \mathcal{C})$ be measurable spaces, and let $f : X \to Y$ and $g : Y \to Z$ be measurable maps. Then, $g_\sharp\left(f_\sharp\mu\right) = (g \circ f)_\sharp\,\mu$ holds.*

**Proof of Lemma 4.9.** For any $C \in \mathcal{C}$, we show that $\left\{g_\sharp\left(f_\sharp\mu\right)\right\}(C) = \left\{(g \circ f)_\sharp\,\mu\right\}(C)$ holds.

$$\begin{aligned}
&\left\{g_\sharp\left(f_\sharp\mu\right)\right\}(C) = \left(f_\sharp\mu\right)\left(g^{-1}(C)\right) = \mu\left(f^{-1}\left(g^{-1}(C)\right)\right) \\
&= \mu\left(\left\{x \in X \mid f(x) \in g^{-1}(C)\right\}\right) = \mu\left(\{x \in X \mid (g \circ f)(x) \in C\}\right) \\
&= \left\{(g \circ f)_\sharp\mu\right\}(C)
\end{aligned}$$

□

In general, let $(X, \mathcal{A})$ be a measurable space, $\pi$ be a product measure on $X^2$, $\mu, \nu$ be measures on $X$, and $p_1(x, y) \coloneqq x$ and $p_2(x, y) \coloneqq y$ be the natural projections. It is known that $\pi$ is a coupling of $\mu$ and $\nu$ if and only if $p_{1\sharp}\pi = \mu$ and $p_{2\sharp}\pi = \nu$ (Villani 2009, Definition 1.1). From this fact and Lemma 4.9, the following lemma holds.

**Lemma 4.10.** *Let $(X, \mathcal{A})$ and $(Y, \mathcal{B})$ be measurable spaces, and $f : X \to Y$ be a measurable mapping. Suppose $\mu, \nu$ are measures on $X$, and $\pi$ is a coupling of $\mu$ and $\nu$. Then, $(f \times f)_\sharp\pi$ is a coupling of $f_\sharp\mu$ and $f_\sharp\nu$.*

**Proof of Lemma 4.10.** Since $f$ is measurable, $f \times f$ is also measurable, and thus $(f \times f)_\sharp\pi$ is a measure on $Y^2$. We show that $p_{1\sharp}\left((f \times f)_\sharp\pi\right) = f_\sharp\mu$ and $p_{2\sharp}\left((f \times f)_\sharp\pi\right) = f_\sharp\nu$ hold. From Lemma 4.9, we have $p_{1\sharp}\left((f \times f)_\sharp\pi\right) = (p_1 \circ (f \times f))_\sharp\,\pi$. For any $(x, y) \in X^2$, we have $p_1 \circ (f \times f)(x, y) = f(x) = (f \circ p_1)(x, y)$. Thus, $(p_1 \circ (f \times f))_\sharp\,\pi =$

$(f \circ p_1)_\sharp \pi = f_\sharp (p_{1\sharp}\pi)$ holds. Since $\pi$ is a coupling of $\mu$ and $\nu$, we have $p_{1\sharp}\pi = \mu$. This implies that $p_{1\sharp}((f \times f)_\sharp \pi) = f_\sharp \mu$ holds.

Similarly, we can show that $p_{2\sharp}((f \times f)_\sharp \pi) = f_\sharp \nu$ holds. □

**Lemma 4.11.** *Let $(X, \mathcal{A}, \mu)$ be a measure space, $(Y, \mathcal{B})$ be a measurable space, $f : X \to Y$ be a measurable mapping, and $g : X \to f(X)$ be the mapping obtained by restricting the codomain of $f$ to $f(X)$. If $f(X) \in \mathcal{B}$, then $g_\sharp \mu = (f_\sharp \mu)|_{f(X)}$ holds.*

**Proof of Lemma 4.11.** We first discuss the necessity of the condition $f(X) \in \mathcal{B}$. The measure $(f_\sharp \mu)|_{f(X)}$ is the restriction of the measure $f_\sharp \mu$ on $Y$ to $f(X)$, and is defined by $(f_\sharp \mu)|_{f(X)}(B) \coloneqq (f_\sharp \mu)(B \cap f(X))$ for any $B \in \mathcal{B}$. Thus, for $(f_\sharp \mu)|_{f(X)}$ to be well-defined, it is necessary that $B \cap f(X) \in \mathcal{B}$. A sufficient condition for this is that $f(X) \in \mathcal{B}$.

Since the mapping $f$ is measurable, the mapping $g$ is measurable with respect to $(f(X), \mathcal{B}|_{f(X)})$. For any $E \in \mathcal{B}|_{f(X)}$, we show that $(g_\sharp \mu)(E) = \{(f_\sharp \mu)|_{f(X)}\}(E)$ holds.

$$\{(f_\sharp \mu)|_{f(X)}\}(E) = (f_\sharp \mu)(E) = \mu(f^{-1}(E)) = \mu(g^{-1}(E)) = (g_\sharp \mu)(E)$$

□

**Lemma 4.12.** *Let $(X, \mathcal{A})$ be a measurable space, $A \in \mathcal{A}$ be a measurable set, and $p_1(x, y) \coloneqq x$ and $p_2(x, y) \coloneqq y$ be the natural projections. Given a measure $\gamma$ on $X^2$, write $\gamma' \coloneqq \gamma|_{A^2}$. If $\gamma(X \times (X \setminus A)) = \gamma((X \setminus A) \times X) = 0$, then $p_{1\sharp}\gamma' = (p_{1\sharp}\gamma)|_A$ and $p_{2\sharp}\gamma' = (p_{2\sharp}\gamma)|_A$ hold. In other words, when the measure $\gamma$ is concentrated on $A^2$, restriction and projection are interchangeable.*

**Proof of Lemma 4.12.** For any $E \in \mathcal{A}|_A$, we show that $(p_{1\sharp}\gamma')(E) = \{(p_{1\sharp}\gamma)|_A\}(E)$ holds. First, we consider the left-hand side.

$$(p_{1\sharp}\gamma')(E) = \gamma'(p_1^{-1}(E)) = \gamma((E \times X) \cap (A \times A)) = \gamma(E \times A).$$

Next, we consider the right-hand side.

$$\begin{aligned}
&\left\{\left(p_{1\sharp}\gamma\right)|_A\right\}(E) = \left(p_{1\sharp}\gamma\right)(E \cap A) = \left(p_{1\sharp}\gamma\right)(E) = \gamma\left(p_1^{-1}(E)\right) = \gamma(E \times X) \\
&= \gamma(E \times A) + \gamma(E \times (X \setminus A)), \\
&\text{by the assumption that } \gamma(X \times (X \setminus A)) = 0, \\
&\therefore \left\{\left(p_{1\sharp}\gamma\right)|_A\right\}(E) = \gamma(E \times A).
\end{aligned}$$

Thus, we have $p_{1\sharp}\gamma' = \left(p_{1\sharp}\gamma\right)|_A$. Similarly, we can show that $p_{2\sharp}\gamma' = \left(p_{2\sharp}\gamma\right)|_A$ holds. □

Using the above lemmas, we present the proof of Proposition 4.1.

**Proof of Proposition 4.1.** Since $W_{d_\mathcal{T}}(\phi_\sharp\mu, \phi_\sharp\nu) = \sum_{e\in E(\mathcal{T})} \frac{1}{2}\left|\phi_\sharp\mu(\Gamma(v_e)) - \phi_\sharp\nu(\Gamma(v_e))\right|$ holds by Theorem 4.1, we show that $W_{d_{\mathrm{Ham}}}(\mu,\nu) = W_{d_\mathcal{T}}(\phi_\sharp\mu, \phi_\sharp\nu)$.

Let $\pi$ be a coupling of $\mu$ and $\nu$. Since $\phi$ is distance-preserving, we note that $\phi$ is injective and measurable, and $\Phi \coloneqq \phi \times \phi$ is also measurable. Thus, $\Phi_\sharp\pi$ is a push-forward measure on $V(\mathcal{T})^2$, and from Lemma 4.10, $\Phi_\sharp\pi$ is a coupling of $\phi_\sharp\mu$ and $\phi_\sharp\nu$.

Since $\phi$ is distance-preserving, we have $d_{\mathrm{Ham}} = d_\mathcal{T} \circ \Phi$. From this fact and the change-of-variables formula (Tao 2011, Exercise 1.4.38), the following holds:

$$\begin{aligned}
&\int_{\Sigma_H^2} d_{\mathrm{Ham}}(\sigma,\tau)d\pi(\sigma,\tau) = \int_{\Sigma_H^2} (d_\mathcal{T}\circ\Phi)(\sigma,\tau)d\pi(\sigma,\tau) \\
&= \int_{V(\mathcal{T})^2} d_\mathcal{T}(a,b)d(\Phi_\sharp\pi)(a,b).
\end{aligned}$$

For any $\pi \in \Pi(\mu,\nu)$, we have $\Phi_\sharp\pi \in \Pi(\phi_\sharp\mu, \phi_\sharp\nu)$. This implies the following:

$$\begin{aligned}
W_{d_{\mathrm{Ham}}}(\mu,\nu) &= \inf_{\pi\in\Pi(\mu,\nu)} \int_{\Sigma_H^2} d_{\mathrm{Ham}}(\sigma,\tau)d\pi(\sigma,\tau) = \inf_{\pi\in\Pi(\mu,\nu)} \int_{V(\mathcal{T})^2} d_\mathcal{T}(a,b)d(\Phi_\sharp\pi)(a,b) \\
&\geq \inf_{\gamma\in\Pi(\phi_\sharp\mu,\phi_\sharp\nu)} \int_{V(\mathcal{T})^2} d_\mathcal{T}(a,b)d\gamma(a,b) = W_{d_\mathcal{T}}(\phi_\sharp\mu,\phi_\sharp\nu).
\end{aligned}$$

Therefore, $W_{d_{\mathrm{Ham}}}(\mu,\nu) \geq W_{d_\mathcal{T}}(\phi_\sharp\mu,\phi_\sharp\nu)$ holds.

Next, we show that $W_{d_{\mathrm{Ham}}}(\mu,\nu) \leq W_{d_\mathcal{T}}(\phi_\sharp\mu,\phi_\sharp\nu)$. Since $\Sigma_H$ is the set of actually generated labels, it is finite. We note that $V(\mathcal{T})$ is a finite discrete set whose $\sigma$-algebra is the power set $2^{V(\mathcal{T})}$. Thus, $\phi(\Sigma_H)$ is a measurable set. Therefore, the measures $\phi_\sharp\mu$

and $\phi_\sharp \nu$ are concentrated on $\phi(\Sigma_H)$. Moreover, for any coupling $\gamma$ of $\phi_\sharp \mu$ and $\phi_\sharp \nu$, the measure $\gamma$ is concentrated on $\phi(\Sigma_H)^2$, i.e.,

$$
\begin{aligned}
&(\phi_\sharp \mu)\left(V(\mathcal{T}) \setminus \phi(\Sigma_H)\right) = \mu\left(\phi^{-1}(V(\mathcal{T}) \setminus \phi(\Sigma_H))\right) = \mu(\emptyset) = 0, \\
&(\phi_\sharp \nu)\left(V(\mathcal{T}) \setminus \phi(\Sigma_H)\right) = \nu\left(\phi^{-1}(V(\mathcal{T}) \setminus \phi(\Sigma_H))\right) = \nu(\emptyset) = 0, \\
&\therefore \gamma\left(V(\mathcal{T}) \times (V(\mathcal{T}) \setminus \phi(\Sigma_H))\right) = \gamma\left((V(\mathcal{T}) \setminus \phi(\Sigma_H)) \times V(\mathcal{T})\right) = 0.
\end{aligned}
$$

We write $\gamma' \coloneqq \gamma|_{\phi(\Sigma_H)^2}$, then from Lemma 4.12, the following holds:

$$
\begin{cases}
p_{1\sharp}\gamma' = \left(p_{1\sharp}\gamma\right)|_{\phi(\Sigma_H)}, \\
p_{2\sharp}\gamma' = \left(p_{2\sharp}\gamma\right)|_{\phi(\Sigma_H)}.
\end{cases} \tag{4.5}
$$

Moreover, let $\psi : \Sigma_H \to \phi(\Sigma_H)$ be the mapping obtained by restricting the codomain of the mapping $\phi : \Sigma_H \to V(\mathcal{T})$ to $\phi(\Sigma_H)$. Then, $\psi$ is bijective, and its inverse mapping $\psi^{-1} : \phi(\Sigma_H) \to \Sigma_H$ exists and is also a distance-preserving measurable mapping. Furthermore, $\Psi^{-1} \coloneqq \psi^{-1} \times \psi^{-1}$ is also a measurable mapping, and we have $d_{\mathcal{T}} = d_{\mathrm{Ham}} \circ \Psi^{-1}$. Thus,

$$
\int_{V(\mathcal{T})^2} d_{\mathcal{T}}(a,b) d\gamma(a,b) = \int_{\phi(\Sigma_H)^2} d_{\mathcal{T}}(a,b) d\gamma'(a,b) = \int_{\phi(\Sigma_H)^2} \left(d_{\mathrm{Ham}} \circ \Psi^{-1}\right)(a,b) d\gamma'(a,b)
$$

by the change-of-variables formula(Tao 2011, Exercise 1.4.38)

$$
\int_{V(\mathcal{T})^2} d_{\mathcal{T}}(a,b) d\gamma(a,b) = \int_{\Sigma_H^2} d_{\mathrm{Ham}}(\sigma,\tau) d\left(\Psi_\sharp^{-1}\gamma'\right)(\sigma,\tau). \tag{4.6}
$$

We show that $\Psi_\sharp^{-1}\gamma'$ is a coupling of $\mu$ and $\nu$, i.e., $p_{1\sharp}\left(\Psi_\sharp^{-1}\gamma'\right) = \mu$ and $p_{2\sharp}\left(\Psi_\sharp^{-1}\gamma'\right) = \nu$ hold. First, we note that $\left(p_1 \circ \Psi^{-1}\right)(a,b) = \psi^{-1}(a) = \left(\psi^{-1} \circ p_1\right)(a,b)$. From this fact

and Lemma 4.9, we have

$$p_{1\sharp}\left(\Psi_{\sharp}^{-1}\gamma'\right) = \left(p_1 \circ \Psi^{-1}\right)_{\sharp}\gamma' = \left(\psi^{-1} \circ p_1\right)_{\sharp}\gamma' = \psi_{\sharp}^{-1}\left(p_{1\sharp}\gamma'\right)$$

since Equation (4.5) holds for $\gamma'$,

$$p_{1\sharp}\left(\Psi_{\sharp}^{-1}\gamma'\right) = \psi_{\sharp}^{-1}\left(\left(p_{1\sharp}\gamma\right)|_{\phi(\Sigma_H)}\right)$$

since $\gamma$ is a coupling of $\phi_{\sharp}\mu$ and $\phi_{\sharp}\nu$,

$$p_{1\sharp}\left(\Psi_{\sharp}^{-1}\gamma'\right) = \psi_{\sharp}^{-1}\left(\left(\phi_{\sharp}\mu\right)|_{\phi(\Sigma_H)}\right)$$

by Lemma 4.11,

$$p_{1\sharp}\left(\Psi_{\sharp}^{-1}\gamma'\right) = \psi_{\sharp}^{-1}\left(\psi_{\sharp}\mu\right) = \left(\psi^{-1} \circ \psi\right)_{\sharp}\mu = \mu.$$

Similarly, we can show that $p_{2\sharp}\left(\Psi_{\sharp}^{-1}\gamma'\right) = \nu$ holds. For any $\gamma \in \Pi(\phi_{\sharp}\mu, \phi_{\sharp}\nu)$, we have $\Psi_{\sharp}^{-1}\gamma' \in \Pi(\mu, \nu)$. Considering Equation (4.6), we have

$$\begin{aligned} W_{d_{\mathcal{T}}}(\phi_{\sharp}\mu, \phi_{\sharp}\nu) &= \inf_{\gamma \in \Pi(\phi_{\sharp}\mu, \phi_{\sharp}\nu)} \int_{V(\mathcal{T})^2} d_{\mathcal{T}}(a, b) d\gamma(a, b) \\ &= \inf_{\gamma \in \Pi(\phi_{\sharp}\mu, \phi_{\sharp}\nu)} \int_{\Sigma_H^2} d_{\mathrm{Ham}}(\sigma, \tau) d\left(\Psi_{\sharp}^{-1}\gamma'\right)(\sigma, \tau) \\ &\geq \inf_{\pi \in \Pi(\mu, \nu)} \int_{\Sigma_H^2} d_{\mathrm{Ham}}(\sigma, \tau) d\pi(\sigma, \tau) = W_{d_{\mathrm{Ham}}}(\mu, \nu). \end{aligned}$$

Therefore, $W_{d_{\mathcal{T}}}(\phi_{\sharp}\mu, \phi_{\sharp}\nu) \geq W_{d_{\mathrm{Ham}}}(\mu, \nu)$ holds. □

# 5 Conclusion

This dissertation demonstrated that by incorporating information inherent to graph data into models and algorithms, we can develop superior methods to address the challenges faced by machine learning on graphs.

In Chapter 3, motivated by the vulnerability of existing methods to sparse directed graphs in clustering tasks, we proposed the Complex non-backtracking (CNBT) matrix that combines the characteristics of the Hermitian adjacency matrix and the non-backtracking (NBT) matrix. We showed that it extends the relationship between the adjacency matrix and the NBT matrix in undirected graphs to directed graphs using complex representations. Numerical experiments confirmed that the clustering method using the eigenvectors of the CNBT matrix is robust even for sparse directed graphs, and discussions based on belief propagation suggested the relationship between the eigenvectors of the CNBT matrix and cluster estimation.

To further explore our findings, several issues need to be addressed. First, there remain many unclear points regarding the relationship between the CNBT matrix and belief propagation. In particular, assuming that these two are related, a probabilistic

interpretation of complex-valued messages in belief propagation is required. Since the validity of matrix representations using complex numbers is not yet fully established, careful consideration is needed. Second, a more in-depth theoretical analysis of the relationship between eigenvalues and cluster structures is necessary. In our study, we focused on the eigenvalue with the largest real part when performing spectral clustering using the eigenvectors of the CNBT matrix. Interestingly, when a directed graph has a cluster structure, it has been confirmed through numerical experiments that the imaginary part of that eigenvalue is almost zero. A theoretical analysis of this phenomenon is an important issue both for exploring the spectral theory of directed graphs and for improving clustering methods.

In Chapter 4, we addressed the trade-off between high discriminative power and computational efficiency faced by recent graph kernels, and focused on accelerating the categorical Wasserstein Weisfeiler-Lehman graph kernel by leveraging its inherent structure. Our study is the first to make explicit the tree structure underlying the WL labels and point out its usefulness for computations in the distance space defined by the Hamming distance. While previous approaches had to rely on entropic regularized approximate solutions for practical use, our approach enables the exact computation of the Wasserstein distance efficiently. Notably, our numerical experiments confirmed that the proposed method scales well with respect to both the number of vertices in the graphs and the number of graphs in the dataset.

We believe that the proposed acceleration makes the categorical Wasserstein Weisfeiler-Lehman graph kernel a more practical graph kernel, but there still remain fundamental issues faced by graph kernels. First, our approach is specialized for the categorical WWL graph kernel and cannot be applied when the information assigned to vertices is continuous attributes rather than categorical labels. This is because our approach is based on the fact that the space defined by the WL labels and the Hamming distance can be represented as a tree structure that preserves the topological structure. In the case of continuous attributes, it is challenging to construct a tree structure that preserves the topological structure of the original space, so we need to allow for some approximation. Thus, it is an important issue to consider whether we should devise the structure of the tree to be constructed or make the edge weights variable in order to make the Wasserstein Weisfeiler-Lehman graph kernel a more general method. Moreover, the categorical WWL graph kernel is not the only graph kernel that

requires improved computational efficiency. For example, methods applying the Fused Gromov-Wasserstein distance to graph kernels also have high expressiveness but face challenges in computational efficiency. There is also a method by S. Chen et al. (2022) that utilizes WL labels, but it constructs a hierarchical probability distribution and requires solving multiple optimal transport problems to compute the distance between graph pairs. It is an interesting direction for future work to see whether the acceleration method using tree structures can be applied to such complex structures.

From a broader perspective, it is also crucial to translate the domain-specific information leveraged in this dissertation into more general-purpose methods such as graph neural networks (GNNs). For example, the non-backtracking matrix is known to mitigate the influence of hub nodes by suppressing backtracking, and GNN architectures that encode this property have already been explored. The Weisfeiler-Lehman algorithm is used as a benchmark for evaluating the expressive power of GNNs, and the message-passing mechanism in GNNs is analogous to that of belief propagation. Therefore, we believe that identifying, isolating, and incorporating underutilized graph information into models in a mathematically tractable manner is essential, not only for solving specific problems but also for enhancing general-purpose methods.

# References


Altschuler, Jason, Jonathan Niles-Weed, and Philippe Rigollet (Dec. 2017). "Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1961–1971.

Andersen, Stig Kjær (Feb. 1991). "Probabilistic reasoning in intelligent systems: Networks of plausible inference". In: *Artificial intelligence* 48.1, pp. 117–124. DOI: 10.1016/0004-3702(91)90084-w.

Aronszajn, N (1950). "Theory of reproducing kernels". In: *Transactions of the American mathematical society* 68.3, pp. 337–404. DOI: 10.1090/s0002-9947-1950-0051437-7.

Berg, Christian, Jens Peter Reus Christensen, and Paul Ressel (June 1984). *Harmonic analysis on semigroups: Theory of positive definite and related functions*. 1984th ed. Springer. DOI: 10.1007/978-1-4612-1128-0.

Berlinet, Alain and Christine Thomas-Agnan (June 2011). *Reproducing kernel Hilbert spaces in probability and statistics*. 2004th ed. Springer. DOI: 10.1007/978-1-4419-9096-9.

Bonneel, Nicolas et al. (Jan. 2015). "Sliced and Radon Wasserstein Barycenters of Measures". In: *Journal of mathematical imaging and vision* 51.1, pp. 22–45. DOI: 10.1007/s10851-014-0506-3.

Borgwardt, Karsten M and Hans-Peter Kriegel (2005). "Shortest-Path Kernels on Graphs". In: *Proceedings of the 5th IEEE International Conference on Data Mining*, pp. 74–81. DOI: 10.1109/ICDM.2005.132.

Carter, J Lawrence and Mark N Wegman (Apr. 1979). "Universal classes of hash functions". In: *Journal of Computer and System Sciences* 18.2, pp. 143–154. DOI: 10.1016/0022-0000(79)90044-8.

Chen, Li et al. (Dec. 2023). "Almost-linear-time algorithms for maximum flow and minimum-cost flow". In: *Communications of the ACM* 66.12, pp. 85–92. DOI: 10.1145/3610940.

Chen, Samantha et al. (July 2022). "Weisfeiler-Lehman Meets Gromov-Wasserstein". In: *Proceedings of the 39th International Conference on Machine Learning*. Vol. 162, pp. 3371–3416.

Chung, Fan R K (Dec. 1996). *Spectral Graph Theory*. Vol. 92. American Mathematical Society. DOI: 10.1090/cbms/092.

Cortes, Corinna and Vladimir Vapnik (Sept. 1995). "Support-vector networks". In: *Machine learning* 20.3, pp. 273–297. DOI: 10.1007/bf00994018.

Cucuringu, Mihai et al. (June 2020). "Hermitian matrices for clustering directed graphs: insights and applications". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 983–992.

Cuturi, Marco (Dec. 2013). "Sinkhorn Distances: Lightspeed Computation of Optimal Transport". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems*. Vol. 2, pp. 2292–2300.

Decelle, Aurelien et al. (Dec. 2011). "Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications". In: *Physical review. E, Statistical, nonlinear, and soft matter physics* 84.6, p. 066106. DOI: 10.1103/PhysRevE.84.066106.

Delling, Daniel et al. (Nov. 2015). "An exact combinatorial algorithm for minimum graph bisection". In: *Mathematical Programming* 153.2, pp. 417–458. DOI: 10.1007/s10107-014-0811-z.

Diestel, Reinhard (Dec. 2024). *Graph theory*. 6th ed. Springer. DOI: 10.1007/978-3-662-70107-2.

Evans, Steven N and Frederick A Matsen (June 2012). "The phylogenetic Kantorovich–Rubinstein metric for environmental sequence samples". In: *Journal of the Royal Statistical Society. Series B, Statistical methodology* 74.3, pp. 569–592. DOI: 10.1111/j.1467-9868.2011.01018.x. eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9868.2011.01018.x.

Gray, Robert M (2005). "Toeplitz and circulant matrices: A review". In: *Foundations and Trends® in Communications and Information Theory* 2.3, pp. 155–239. DOI: 10.1561/0100000006.

Guo, Krystal and Bojan Mohar (May 2017). "Hermitian adjacency matrix of digraphs and mixed graphs". In: *Journal of graph theory* 85.1, pp. 217–248. DOI: 10.1002/jgt.22057.

Hashimoto, Kiichiro (Jan. 1989). "Zeta Functions of Finite Graphs and Representations of $p$-Adic Groups". In: *Automorphic Forms and Geometry of Arithmetic Varieties*. Vol. 15. Mathematical Society of Japan, pp. 211–281. DOI: 10.2969/aspm/01510211.

Haussler, D (1999). "Convolution kernels on discrete structures". In: *Tech. Rep.*

Huangfu, Qi and Julian Hall (Mar. 2018). "Parallelizing the dual revised simplex method". In: *Mathematical programming computation* 10.1, pp. 119–142. DOI: 10.1007/s12532-017-0130-5.

Hubert, Lawrence and Phipps Arabie (Dec. 1985). "Comparing partitions". In: *Journal of Classification* 2.1, pp. 193–218. DOI: 10.1007/BF01908075.

Karrer, Brian and M E J Newman (Jan. 2011). "Stochastic blockmodels and community structure in networks". In: *Physical review. E, Statistical, nonlinear, and soft matter physics* 83.1, p. 016107. DOI: 10.1103/PhysRevE.83.016107.

Kondor, Risi and Horace Pan (Dec. 2016). "The Multiscale Laplacian Graph Kernel". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. Vol. 29.

Konno, Norio et al. (June 2019). "A new weighted Ihara zeta function for a graph". In: *Linear algebra and its applications* 571, pp. 154–179. DOI: 10.1016/j.laa.2019.02.022.

Kriege, Nils M, Pierre-Louis Giscard, and Richard Wilson (Dec. 2016). "On Valid Optimal Assignment Kernels and Applications to Graph Classification". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 1623–1631.

Krzakala, Florent et al. (Dec. 2013). "Spectral redemption in clustering sparse networks". In: *Proceedings of the National Academy of Sciences of the United States of America* 110.52, pp. 20935–20940. DOI: 10.1073/pnas.1312486110.

Laenen, Steinar and He Sun (Dec. 2020). "Higher-order spectral clustering of directed graphs". In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. Curran Associates Inc., pp. 941–951.

Le, Tam et al. (Dec. 2019). "Tree-Sliced Variants of Wasserstein Distances". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 12304–12315.

Li, Mengyu et al. (2023). "Importance Sparsification for Sinkhorn Algorithm". In: *Journal of Machine Learning Research* 24.247, pp. 1–44.

Liu, Jianxi and Xueliang Li (Feb. 2015). "Hermitian-adjacency matrices and Hermitian energies of mixed graphs". In: *Linear algebra and its applications* 466, pp. 182–207. DOI: 10.1016/j.laa.2014.10.028.

Lozupone, Catherine and Rob Knight (Dec. 2005). "UniFrac: a New Phylogenetic Method for Comparing Microbial Communities". In: *Applied and Environmental Microbiology* 71.12, pp. 8228–8235. DOI: 10.1128/AEM.71.12.8228-8235.2005. eprint: https://journals.asm.org/doi/pdf/10.1128/aem.71.12.8228-8235.2005.

Lozupone, Catherine A et al. (Mar. 2007). "Quantitative and qualitative beta diversity measures lead to different insights into factors that structure microbial communities". In: *Applied and environmental microbiology* 73.5, pp. 1576–1585. DOI: 10.1128/AEM.01996-06.

Luxburg, Ulrike von (Dec. 2007). "A tutorial on spectral clustering". In: *Statistics and computing* 17.4, pp. 395–416. DOI: 10.1007/s11222-007-9033-z.

Martin, James, Tim Rogers, and Luca Zanetti (Feb. 2024). "An iterative spectral algorithm for digraph clustering". In: *Journal of complex networks* 12.2. DOI: 10.1093/comnet/cnae016.

Menezes, Alfred John, Paul C van Oorschot, and Scott A Vanstone (Oct. 1996). *Handbook of applied cryptography*. CRC Press.

Mezard, Marc and Andrea Montanari (Jan. 2009). *Information, Physics, and Computation*. Oxford University Press. DOI: 10.1093/acprof:oso/9780198570837.001.0001.

Mohar, Bojan (Jan. 2020). "A new kind of Hermitian matrices for digraphs". In: *Linear algebra and its applications* 584, pp. 343–352. DOI: 10.1016/j.laa.2019.09.024.

Morris, Christopher et al. (July 2020). "TUDataset: A collection of benchmark datasets for learning with graphs". In: *ICML Workshop on Graph Representation Learning and Beyond*.

Newman, M E J (Aug. 2013). *Spectral community detection in sparse networks*. arXiv: 1308.6494.

Newman, Mark (25 3 2010). *Networks: An Introduction*. Oxford University Press. DOI: 10.1093/acprof:oso/9780199206650.001.0001.

Ng, Andrew, Michael Jordan, and Yair Weiss (Dec. 2001). "On Spectral Clustering: Analysis and an algorithm". In: *Proceedings of the 15th International Conference on Neural Information Processing Systems*. Vol. 14.

Orlin, James B (Aug. 1997). "A polynomial time primal network simplex algorithm for minimum cost flows". In: *Mathematical programming* 78.2, pp. 109–129. DOI: 10.1007/bf02614365.

Oyelade, Jelili et al. (July 2019). *Data clustering: Algorithms and its applications*. IEEE. DOI: 10.1109/iccsa.2019.000-1.

Peyré, Gabriel and Marco Cuturi (2020). *Computational Optimal Transport*. arXiv: 1803.00567.

Pržulj, Nataša (Jan. 2007). "Biological network comparison using graphlet degree distribution". In: *Bioinformatics* 23.2, e177–e183. DOI: 10.1093/bioinformatics/btl301. eprint: https://academic.oup.com/bioinformatics/article-pdf/23/2/e177/49820586/bioinformatics_23_2_e177.pdf.

Rand, William M (Dec. 1971). "Objective criteria for the evaluation of clustering methods". In: *Journal of the American Statistical Association* 66.336, p. 846. DOI: 10.2307/2284239.

Rohe, Karl, Tai Qin, and Bin Yu (Nov. 2016). "Co-clustering directed graphs to discover asymmetries and directional communities". In: *Proceedings of the National Academy of Sciences of the United States of America* 113.45, pp. 12679–12684. DOI: 10.1073/pnas.1525793113.

Saade, Alaa, Florent Krzakala, and Lenka Zdeborová (Dec. 2014). "Spectral clustering of graphs with the Bethe Hessian". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*. MIT Press, pp. 406–414.

Sando, Keishi and Hideitsu Hino (Aug. 2025). "Complex non-backtracking matrix for directed graphs". In: *Journal of complex networks* 13.4, cnaf012. DOI: 10.1093/comnet/cnaf012.

Satuluri, Venu and Srinivasan Parthasarathy (Mar. 2011). "Symmetrizations for clustering directed graphs". In: *Proceedings of the 14th International Conference on Extending Database Technology*. Association for Computing Machinery, pp. 343–354. DOI: 10.1145/1951365.1951407.

Scetbon, Meyer, Marco Cuturi, and Gabriel Peyré (July 2021). "Low-Rank Sinkhorn Factorization". In: *Proceedings of the 38th International Conference on Machine Learning*. Vol. 139, pp. 9344–9354.

Schulz, Till Hendrik et al. (July 2022). "A generalized Weisfeiler-Lehman graph kernel". In: *Machine learning* 111.7, pp. 2601–2629. DOI: 10.1007/s10994-022-06131-w.

Shervashidze, Nino et al. (2011). "Weisfeiler-Lehman Graph Kernels". In: *Journal of Machine Learning Research* 12.77, pp. 2539–2561. DOI: 10.5555/1953048.2078187.

Siglidis, Giannis et al. (2020). "GraKeL: A Graph Kernel Library in Python". In: *Journal of Machine Learning Research* 21.54, pp. 1–5.

Sinkhorn, Richard (June 1964). "A Relationship Between Arbitrary Positive Matrices and Doubly Stochastic Matrices". In: *The Annals of Mathematical Statistics* 35.2, pp. 876–879.

Stark, H M and A A Terras (July 1996). "Zeta Functions of Finite Graphs and Coverings". In: *Advances in mathematics* 121.1, pp. 124–165. DOI: 10.1006/aima.1996.0050.

Tao, Terence (2011). *An Introduction to Measure Theory*. American Mathematical Society.

Tarfulea, Andrei and Robert Perlis (July 2009). "An Ihara formula for partially directed graphs". In: *Linear algebra and its applications* 431.1, pp. 73–85. DOI: 10.1016/j.laa.2009.02.006.

Titouan, Vayer et al. (May 2019). "Optimal Transport for structured data with application on graphs". In: *Proceedings of the 36th International Conference on Machine Learning*. PMLR, pp. 6275–6284.

Togninalli, Matteo et al. (Dec. 2019). "Wasserstein Weisfeiler-Lehman Graph Kernels". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 6439–6449.

Villani, Cedric (Dec. 2009). *Optimal Transport: Old and New*. Springer. DOI: 10.1007/978-3-540-71050-9.

Yedidia, Jonathan S, William T Freeman, and Yair Weiss (Jan. 2003). "Understanding belief propagation and its generalizations". In: *Exploring artificial intelligence in the new millennium*. Morgan Kaufmann Publishers Inc., pp. 239–269. DOI: 10.5555/779343.779352.