Winter 12-15-2019

Kernel Methods for Graph-structured Data Analysis

Zhen Zhang
Washington University in St. Louis

WASHINGTON UNIVERSITY IN ST. LOUIS

School of Engineering and Applied Science

Department of Electrical & Systems Engineering

Dissertation Examination Committee:
Arye Nehorai, Chair
Ulugbek Kamilov
Neal Patwari
Lingfei Wu
Xuan Zhang

Kernel Methods for Graph-structured Data Analysis

by

Zhen Zhang

A dissertation presented to
The Graduate School
of Washington University in
partial fulfillment of the
requirements for the degree
of Doctor of Philosophy

December 2019
Saint Louis, Missouri

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to express my deep and sincere gratitude to my research advisor, Dr. Arye Nehorai. Since I joined his lab in August 2015, he has been continuously providing professional guidance and support for my Ph.D. research. I would like to thank him for his valuable suggestions and inspirations. I would like to thank him for always encouraging me to take mathematical courses and to learn new statistical tools, which greatly helped me in conducting in-depth research. I am also very grateful for being given enough freedom to explore various interesting machine learning topics, which I really enjoyed and made me decide to continue to work on in the future.

I wish to thank my dissertation defense committee members, Dr. Neal Patwari, Dr. Ulugbek Kamilov, Dr. Xuan Zhang, and Dr. Lingfei Wu, for their insightful comments on improving my dissertation. I wish to thank Dr. Lingfei Wu for also being my collaborator of my research project. I could always find inspirations and exciting ideas during the discussions with him.

I would like to thank my labmates, Alex, Jichuan, Mengxue, Mianzhi, Prateek, Yijian, Zhenqi, Yiqi, Hesam, and Eric for their help and support. I always feel fortunate to collaborate and work with them. We had many exciting times when we exchange ideas in the front of the white board, which I will never forget.

I am also deeply thankful to my parents for their endless love. They always encouraged me and gave me confidence and support whenever I encountered obstacles during my pursuit for my dream.

Zhen Zhang

Washington University in Saint Louis

December 2019

Dedicated to my parents

ABSTRACT OF THE DISSERTATION

Kernel Methods for Graph-structured Data Analysis

by

Zhen Zhang

Doctor of Philosophy in Electrical Engineering

Washington University in St. Louis, 2019

Professor Arye Nehorai, Chair

Structured data modeled as graphs arise in many application domains, such as computer vision, bioinformatics, and sociology. In this dissertation, we focus on three important topics in graph-structured data analysis: graph comparison, graph embeddings, and graph matching, for all of which we propose effective algorithms by making use of kernel functions and the corresponding reproducing kernel Hilbert spaces.

For the first topic, we develop effective graph kernels, named as "RetGK," for quantitatively measuring the similarities between graphs. Graph kernels, which are positive definite functions on graphs, are powerful similarity measures, in the sense that they make various kernel-based learning algorithms, for example, clustering, classification, and regression, applicable to structured data. Our graph kernels are obtained by two-step embeddings. In the first step, we represent the graph nodes with numerical vectors in Euclidean spaces. To do this, we revisit the concept of random walks and introduce a new node structural role descriptor, the return probability feature. In the second step, we represent the whole graph

with an element in reproducing kernel Hilbert spaces. After that, we can naturally obtain our graph kernels. The advantages of our proposed kernels are that they can effectively exploit various node attributes, while being scalable to large graphs. We conduct extensive graph classification experiments to evaluate our graph kernels. The experimental results show that our graph kernels significantly outperform state-of-the-art approaches in both accuracy and computational efficiency.

For the second topic, we develop scalable attributed graph embeddings, named as "SAGE." Graph embeddings are Euclidean vector representations, which encode the attributed and the topological information. With graph embeddings, we can apply all the machine learning algorithms, such as neural networks, regression/classification trees, and generalized linear regression models, to graph-structured data. We also want to highlight that SAGE considers both the edge attributes and node attributes, while RetGK only considers the node attributes. "SAGE" is a extended work of "RetGK," in the sense that it is still based on the return probabilities of random walks and is derived from graph kernels. But "SAGE" uses a totally different strategy, i.e., the "distance to kernel and embeddings" algorithm, to further represent graphs. To involve the edge attributes, we introduce the adjoint graph, which can help convert edge attributes to node attributes. We conduct classification experiments on graphs with both node and edge attributes. "SAGE" achieves the better performances than all previous methods.

For the third topic, we develop a new algorithm, named as "KerGM," for graph matching. Typically, graph matching problems can be formulated as two kinds of quadratic assignment problems (QAPs): Koopmans-Beckmann's QAP or Lawler's QAP. In our work, we

provide a unifying view for these two problems by introducing new rules for array operations in Hilbert spaces. Consequently, Lawler's QAP can be considered as the Koopmans-Beckmann's alignment between two arrays in reproducing kernel Hilbert spaces, making it possible to efficiently solve the problem without computing a huge affinity matrix. Furthermore, we develop the entropy-regularized Frank-Wolfe algorithm for optimizing QAPs, which has the same convergence rate as the original Frank-Wolfe algorithm while dramatically reducing the computational burden for each outer iteration. Furthermore, we conduct extensive experiments to evaluate our approach, and show that our algorithm has superior performance in both matching accuracy and scalability.

Chapter 1

Introduction

Graph is a popular data structure and is employed extensively in many domains, such as bioinformatics, computer vision, and sociology. Graph is a powerful tool for capturing the interaction (i.e., edges) between individuals (nodes). For example, in bioinformatics, a protein molecule can be modeled as a graph, where graph nodes represent amino acids and graph edges characterize the biological interaction between nodes. In computer vision, an image can also be represented as a graph, where graph nodes are the detected feature points (usually called "landmarks") and graph edges are generated based on the distances between landmarks. In sociology, social networks are popular in describing interconnections among people, groups, and organizations, where graph nodes represent people and graph edges represent the collaborations or the friendships between them. On the other hand, many other fundamental data structures, for example, strings and trees, are considered as special graph instances. Sometimes, time series, vectors, and matrices [1, 2] can also be treated as graphs by properly defining the nodes and edges. Therefore, graphs are universal data structures.

The connectivity structure plays a central role in a graph, which can help, among many other applications, to design drugs, to analyze human behaviors through their social networks, and to align images taken from different views. Notably, graphs are usually coupled

with node and edge attributes, which are called "attributed graphs" in literature. For example, a chemical compound may have both discrete and continuous node attributes, which respectively describe the type and position of atoms. In the graph representations of images [3], the edge attribute can be the distance between landmarks or the angle between the edge and the horizontal line. Almost all graph-related tasks are trying to extract or analyze the structure and attributes information.

Graph comparison is a fundamental problem. A quantitative similarity or distance measure between graphs is the building block for many graph-related applications. Such a measure can discriminate neurological disorders of brain networks [4], can identify structurally close molecules that have similar functions, for drug design [5], and can track the changes in dynamic climate networks [6]. Unfortunately, graph comparison is very challenging because graphs lacks the properties of vector spaces. As we know, it is straightforward to obtain the distance between Euclidean vectors by taking the difference of corresponding components, e.g., $\text{dist}(\vec{\boldsymbol{x}}, \vec{\boldsymbol{y}}) = \sqrt{\sum_{i=1}^{n} (\vec{\boldsymbol{x}}_i - \vec{\boldsymbol{y}}_i)^2}$. However, finding the corresponding nodes between graphs is a (even more) difficult problem, known as "graph matching." One class of methods for graph comparison are based on feature extraction of networks. The quantitative measure is then computed by comparing feature vectors. These features include degree distribution, entropy, clustering coefficient [7], and the histogram of length of paths [8]. However, the structural information provided by these features are often limited or incomplete. The node and edge attributes are also not considered in these features. In recent years, graph kernels are getting more attention [9]. Roughly speaking, graph kernels are similarity measures on graphs. More specifically, a graph kernel is a positive semi-definite function defined on the space of graphs, $\mathcal{X}_G$. This function can be expressed as the inner product between vectors in a Hilbert space, $\mathcal{H}$. That is, for any graph kernel $K : \mathcal{X}_G \times \mathcal{X}_G \to \mathbb{R}$, there exist an (implicit)

feature map, $\psi : \mathcal{X}_G \to \mathcal{H}$, such that

$$K(\mathcal{G}_1, \mathcal{G}_2) = \langle \psi(\mathcal{G}_1), \psi(\mathcal{G}_2) \rangle_{\mathcal{H}}, \forall \mathcal{G}_1, \mathcal{G}_2 \in \mathcal{X}_G. \tag{1.1}$$

Graph kernels can usually well capture both the topological and attribute information. Another advantage is that graph kernels allow all the kernel-based machine learning algorithms, e.g., Gaussian process regression [10], kernel support vector machine [11], and kernel principal component analysis [12], to work directly on graph-structured data. In this dissertation, we develop effective graph kernels, which show superior performance in real-data experiments.

Graph embedding is another popular topic in machine learning [13]. In literature, the graph embedding can be roughly divided into two groups: (1) Euclidean vector represents of nodes, e.g., the works in [14, 15], which are also known as "node embeddings," (2) Euclidean vector representations of the whole graph, e.g., the works in [16, 17]. The node embedding can be applied to node classification [18], clustering [19], and link prediction [20]. The whole graph embedding can be applied to the tasks defined on the dataset of multiple graphs. In this dissertation, when we say "graph embeddings," we refer to the second group. In our work, we develop a scalable attributed graph embedding algorithm, which can encode all types of attribute information and the connectivity information of graphs into an Euclidean vector.

Graph matching, which aims at finding the correspondence between nodes of two given graphs, is another longstanding problem in graph theory. It plays an important role in many applications, ranging from optical character recognition [21] in computer vision, domain adaptation [22] in machine learning, and evolutionary conserved pathways investigation across species [23] in biology. Graph matching problems can be divided into two categories: exact graph matching and inexact graph matching [24]. Exact graph matching requires the

correspondence between nodes to be "edge-preserving" in the sense that if two nodes are linked by an edge in the first graph, their corresponding nodes must be linked by an edge in the second graph. This problem is known as "graph isomorphism." [25] However, the strict requirement of exact graph matching highly limits its applications in practice. For example, protein-protein interaction networks are unlikely to be isomorphic. But it is still of great importance to match proteins across species. Therefore, inexact graph matching is more popular. Instead of searching for the *exact* ways of matching nodes, inexact graph matching searches for the *best* ways, which provides much flexibility in formulating and solving real-world graph matching problems. We note that inexact graph matching is closely related to graph comparison. As mentioned above, graph comparison can be conducted by first finding the node correspondences and then taking the summation of the "differences" [1] of each pair of nodes and each pair of edges. Typically, inexact graph matching problems can be formulated as two kinds of quadratic assignment problems (QAPs): Koopmans-Beckmann's QAP [26] or Lawler's QAP [27]. Koopmans-Beckmann's QAP is the structural alignment between two adjacency matrices, which, as a result, can be written as the standard Frobenius inner product between two permutated matrices of the size $n \times n$, where n denotes the number of nodes. However, Koopmans-Beckmann's QAP cannot incorporate complex edge attribute information, which is usually very important in characterizing the relation between nodes. Lawler's QAP can tackle this issue, because it attempts to maximize the overall similarity that well encodes all the attribute information. However, the key concern of the Lawler's QAP is that it needs to estimate the $n^2 \times n^2$ pairwise affinity matrix, limiting its application to very small graphs.

[1] The differences may be obtained by comparing the corresponding node and edge attributes.

1.1 Contributions of This Work

In this dissertation, we develop novel algorithms for solving the graph comparison, graph embedding, and inexact graph matching problem, which are named "RetGK," "SAGE," and "KerGM," respectively. For each algorithm, we conduct comprehensive experiments on both the synthetic and real datasets to demonstrate its effectiveness. We summarize the main contributions as follows.

Graph kernels based on return probabilities of random walks (RetGK)[28] We propose efficient graph kernels. To do this, we revisit the concept of random walks, introducing a new node structural role descriptor, the return probability feature (RPF). We rigorously show that the RPF is isomorphism-invariant and encodes very rich connectivity information. Moreover, RPF allows us to consider node-attributed and non-attributed graphs in a unified framework. With the RPF, we can embed (non-)attributed graphs into a Hilbert space. After that, we naturally obtain our return probability-based graph kernels. Making use of the approximate feature maps technique, we represent each graph with a multi-dimensional tensor and design a family of computationally efficient graphs kernels.

Scalable attributed graph embeddings (SAGE) [17] We propose scalable attributed graph embeddings for graphs with any type of node attributes and edge attributes. SAGE is able to encode both the topological connectivity and attributes information of graphs. We introduce a novel strategy of converting edge-attributed graphs to node-attributed graphs, opening a door to involving edge attributes information for many other graph embeddings that are derived based on the node attributes. We mathematically and empirically show that our proposed graph embedding SAGE exhibits linear complexity in the number of graphs.

Kernelized graph matching (KerGM) [29] We focus on solving the Lawler's QAP. To do this, we derive an equivalent formulation of Lawler's QAP, based on a very mild assumption that edge affinities are characterized by kernels. After introducing new rules for array operations in Hilbert spaces, named as $\mathcal{H}-$operations, we rewrite Lawler's QAP as the Koopmanns-Beckmann's alignment between two arrays in a reproducing kernel Hilbert space (RKHS), which allows us to solve the Lawler's QAP without computing the huge affinity matrix. Taking advantage of the $\mathcal{H}-$operations, we develop a path-following strategy for mitigating the local maxima issue of QAPs. In addition to the kernelized graph matching formulation, we propose a numerical optimization algorithm, the entropy-regularized Frank-Wolfe (EnFW) algorithm, for solving large-scale QAPs. The EnFW has the same convergence rate as the original Frank-Wolfe algorithm, with far less computational burden in each iteration. Extensive experimental results show that our KerGM, together with the EnFW algorithm, achieves superior performance in both matching accuracy and scalability.

1.2 Organization of This Dissertation

The rest of this dissertation is organized as follows. In Chapter 2, we present the attributed graphs and give a brief introduction to the concept of kernels and the corresponding reproducing kernel Hilbert spaces. In Chapter 3, we introduce the return probability feature and investigate its properties. We show how to represent node-attributed graphs with Hilbert vectors and multi-dimensional tensors, for both of which we develop the corresponding graph kernels. In Chapter 4, we present a novel algorithm for embedding attributed graphs with Euclidean vectors. This algorithm can handle both the node and the edge attributes of graphs. In Chapter 5, we consider the matchig problem between attributed graphs. We

derive the effective kernelized graph matching algorithm, and develop computationally efficient optimization method to solve quadratic assignment problems. Finally, in Chapter 6, we summarize the dissertation and propose potential future directions.

1.3 Notations

We use a capital italic bold letter, e.g., $\boldsymbol{X}$, to represent a matrix and a lowercase italic bold letter with an arrow, e.g., $\vec{\boldsymbol{v}}$, to represent a Euclidean vector. For a given matrix $\boldsymbol{X}$, $\boldsymbol{X}^T$ denotes its transpose. For two matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$, $\langle \boldsymbol{X}, \boldsymbol{Y} \rangle_{\mathrm{F}}$, $\boldsymbol{X}\boldsymbol{Y}$, $\boldsymbol{X} \circ \boldsymbol{Y}$, and $\boldsymbol{X} \otimes \boldsymbol{Y}$ denote the Frobenius inner product, the usual matrix multiplication, the Hadamard product, and the tensor product, respectively.

We use a calligraphic symbol, e.g., $\mathcal{H}$, to represent a Hilbert space and a lowercase Greek symbol, e.g., ψ, to represent a Hilbert vector. For two given Hilbert vectors ψ and φ, $\langle \psi, \varphi \rangle_{\mathcal{H}}$ denotes the Hilbert inner product between them.

Chapter 2

Background

2.1 Attributed Graphs

We consider the attributed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{A}_N, \mathcal{A}_E\}$ of n nodes and m edges, where $\mathcal{V} = \{v_1, v_2, ..., v_n\}$ is the set of nodes, $\mathcal{E}$ is the set of edges, and $\mathcal{A}_N$ and $\mathcal{A}_E$ are respectively the node attributes and edge attributes set. The connectivity information can be characterized by the adjacency matrix[2] $\boldsymbol{A} \in \{0,1\}^{n \times n}$, or the incidence matrix, $\boldsymbol{C} \in \{0,1\}^{n \times m}$. The adjacency matrix $\boldsymbol{A}$ is defined as

$$\boldsymbol{A}(i,j) = \begin{cases} 1, & \text{if } (v_i, v_j) \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases}, \tag{2.1}$$

The incidence matrix $\boldsymbol{C}$ describes the node-edge correspondence, i.e.,

$$\boldsymbol{C}(v_i, e) = \boldsymbol{C}(v_j, e) = \begin{cases} 1, & \text{if } e = (v_i, v_j) \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases}. \tag{2.2}$$

[2]Note that $\boldsymbol{A}$ is a binary matrix, even for weighted graphs. The weights are treated as edge attributes.

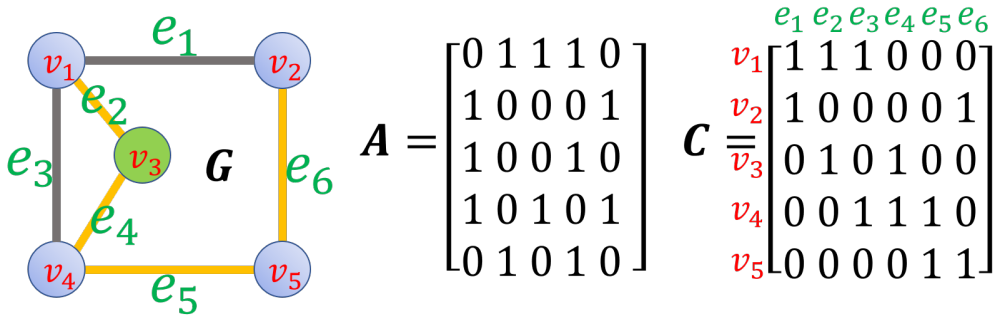


Figure 2.1: Left: A graph $\mathcal{G}$. Middle: The corresponding adjacency matrix $\boldsymbol{A}$. Right: The corresponding incidence matrix $\boldsymbol{C}$.

The degree matrix, $\boldsymbol{D}$, is a diagonal matrix, which is defined such that $\boldsymbol{D}(i,i) = \sum_{(v_i,v_j)\in\mathcal{E}} 1$. Moreover, it can be easily checked that

$$\boldsymbol{A} = \boldsymbol{C}\boldsymbol{C}^T - \boldsymbol{D}. \tag{2.3}$$

In Fig. 2.1, we show a graph and its adjacency matrix and incidence matrix. The blue and green colors of nodes represent two different node attributes, i.e., $\mathcal{A}_N = \{\text{“}blue\text{”}, \text{“}green\text{”}\}$. The yellow and grey colors of edges represent two different edge attributes, i.e., $\mathcal{A}_E = \{\text{“}yellow\text{”}, \text{“}grey\text{”}\}$.

2.2 Kernels and Reproducing Kernel Hilbert Spaces

Definition 2.1. *Let $\mathcal{X}$ be a nonempty set, and let $\mathcal{H}$ be a Hilbert space of $\mathbb{R}-$valued function defined on $\mathcal{X}$. A function $k : \mathcal{X} \times \mathcal{X}$ is called a reproducing kernel of $\mathcal{H}$, and $\mathcal{H}$ is a reproducing kernel Hilbert space, if k satisfies:*

1. $\forall x \in \mathcal{X}$, $k(\cdot, x) \in \mathcal{H}$,

2. $\forall x \in \mathcal{X}$, $f \in \mathcal{H}$, $\langle f, k(\cdot, x)\rangle_{\mathcal{H}} = f(x)$.

If we define the feature map $\psi : \mathcal{X} \to \mathcal{H}$ *as* $\psi(x) = k(\cdot, x)$*, then we have* $\langle \psi(x), \psi(y)\rangle_{\mathcal{H}} = k(x, y)$, $\forall x, y \in \mathcal{X}$.

Note that the 2nd property introduced above is called "reproducing property," which is a special property of RKHS. Note also that in practice, we have access to only the kernel function k. The feature map ψ is usually implicit and (potentially) infinite-dimensional.

Proposition 2.1. *The kernel* k *defined above satisfies the positive semi-definite property, i.e.,* $\forall n \in \mathbb{N}$, $\forall x_1, x_2, ..., x_n \in \mathcal{X}$, *and* $\forall c_1, c_2, ..., c_n \in \mathbb{R}$,

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k(x_i, x_j) \geq 0. \tag{2.4}$$

Proof.

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k(x_i, x_j) &= \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \langle \psi(x_1), \psi(x_j)\rangle_{\mathcal{H}} \\ &= \| \sum_{i=1}^{n} c_i \psi(x_i)\|_{\mathcal{H}}^2 \geq 0\end{aligned} \tag{2.5}$$

□

The following Moore–Aronszajn theorem [30] introduces RKHS from a converse direction.

Theorem 2.1. *(Moore–Aronszajn) Suppose* k *is a symmetric, positive definite function on a set* $\mathcal{X}$*. Then there is a unique Hilbert space* $\mathcal{H}$ *of functions on* $\mathcal{X}$ *for which* k *is a reproducing kernel.*

2.2.1 Basic Properties

Property 2.1. *(sum of kernels) Let k_1 and k_2 be reproducing kernels on $\mathcal{X}$, then $\forall \alpha_1, \alpha_2 \geq 0$, $k = \alpha_1 k_1 + \alpha_2 k_2$ is also a kernel on $\mathcal{X}$.*

Proof. Based on the Moore–Aronszajn theorem, it is sufficient to show k is positive semi-definite. $\forall x_1, x_2, ..., x_n \in \mathcal{X}$, and $\forall c_1, c_2, ..., c_n \in \mathbb{R}$, we have

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k(x_i, x_j) &= \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \big[\alpha_1 k_1(x_i, x_j) + \alpha_2 k_2(x_i, x_j)\big] \\ &= \alpha_1 \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k_1(x_i, x_j) + \alpha_2 \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k_2(x_i, x_j) \\ &\geq 0,\end{aligned} \tag{2.6}$$

where the last inequality holds because k_1 and k_2 are positive semi-definite. □

Property 2.2. *(tensor product of kernels) Let k_x and k_y be reproducing kernels on $\mathcal{X}$ and $\mathcal{Y}$, respectively. Then $k = k_x \otimes k_y$ is a reproducing kernel on $\mathcal{X} \times \mathcal{Y}$. Note that $k_x \otimes k_y$ is defined such that $k_x \otimes k_y\big((x_1, y_1), (x_2, y_2)\big) = k_x(x_1, x_2) k_y(y_1, y_2)$, $\forall (x_1, y_1), (x_2, y_2) \in \mathcal{X} \times \mathcal{Y}$.*

Before we prove the above property, we first introduce a lemma.

Lemma 2.1. *If both $\boldsymbol{A}$ and $\boldsymbol{B}$ are $n \times n$ positive semi-definite matrices, then their Hadamard product (see [31]), $\boldsymbol{A} \circ \boldsymbol{B}$, is also positive semi-definite.*

Proof. Let $\boldsymbol{B} = \sum_{k=1}^{n} \lambda_k \vec{\boldsymbol{u}}_k \vec{\boldsymbol{u}}_k^T$ be the eigen-decomposition of $\boldsymbol{B}$. Then $\boldsymbol{A} \circ \boldsymbol{B}$ can be written as

$$\boldsymbol{A} \circ \boldsymbol{B} = \sum_{k=1}^{n} \lambda_k \boldsymbol{A} \circ (\vec{\boldsymbol{u}}_k \vec{\boldsymbol{u}}_k^T) = \sum_{k=1}^{n} \lambda_k \mathrm{diag}(\vec{\boldsymbol{u}}_k) \boldsymbol{A} \mathrm{diag}(\vec{\boldsymbol{u}}_k). \tag{2.7}$$

For any vector $\vec{\boldsymbol{c}} \in \mathbb{R}^n$, we have

$$\vec{\boldsymbol{c}}^T(\boldsymbol{A} \circ \boldsymbol{B})\vec{\boldsymbol{c}} = \sum_{k=1}^{n} \lambda_k \vec{\boldsymbol{c}}^T \big[\mathrm{diag}(\vec{\boldsymbol{u}}_k)\boldsymbol{A}\mathrm{diag}(\vec{\boldsymbol{u}}_k)\big]\vec{\boldsymbol{c}} = \sum_{k=1}^{n} \lambda_k \big[\mathrm{diag}(\vec{\boldsymbol{u}}_k)\vec{\boldsymbol{c}}\big]^T \boldsymbol{A}\big[\mathrm{diag}(\vec{\boldsymbol{u}}_k)\vec{\boldsymbol{c}}\big] \geq 0. \tag{2.8}$$

□

In the following, we prove Property 2.2.

Proof. Still based on the Moore–Aronszajn theorem, it is sufficient to show $k = k_1 \otimes k_2$ is positive semi-definite.

$\forall (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$, and $\forall c_1, c_2, ..., c_n \in \mathbb{R}$, we have

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k\big((x_i, y_i), (x_j, y_j)\big) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k_x(x_1, x_2) k_y(y_1, y_2) = \vec{\boldsymbol{c}}^T(\boldsymbol{K}_x \circ \boldsymbol{K}_y)\vec{\boldsymbol{c}} \geq 0, \tag{2.9}$$

where $\vec{\boldsymbol{c}} = [c_1, c_2, ..., c_n]^T \in \mathbb{R}^n$, $\boldsymbol{K}_X \in \mathbb{R}^n$ is the kernel matrix of the values of k_x on $x_1, x_2, ..., x_n$, and $\boldsymbol{K}_y \in \mathbb{R}^n$ is the kernel matrix of the values of k_y on $y_1, y_2, ..., y_n$. Note that the last inequality holds because of Lemma 2.1. □

2.2.2 Kernels on Euclidean Spaces

Euclidean spaces are the most common spaces in machine learning. In this part, we introduce some kernels on Euclidean spaces, i.e., $\mathcal{X} = \mathbb{R}^n$, which both of our two works are based on.

1. polynomial kernels: $k(\vec{\boldsymbol{x}}, \vec{\boldsymbol{y}}) = (\langle \vec{\boldsymbol{x}}, \vec{\boldsymbol{y}} \rangle + c)^m$, where $c \geq 0$, $m \in \mathbb{N}$.

2. exponential kernel: $k(\vec{\boldsymbol{x}}, \vec{\boldsymbol{y}}) = \exp(\langle \vec{\boldsymbol{x}}, \vec{\boldsymbol{y}} \rangle)$.

3. Gaussian kernel: $k(\vec{\boldsymbol{x}}, \vec{\boldsymbol{y}}) = \exp(-\gamma\|\vec{\boldsymbol{x}} - \vec{\boldsymbol{y}}\|^2)$, where $\gamma \geq 0$.

4. Laplacian kernel: $k(\vec{\boldsymbol{x}}, \vec{\boldsymbol{y}}) = \exp(-\gamma\|\vec{\boldsymbol{x}} - \vec{\boldsymbol{y}}\|)$, where $\gamma \geq 0$.

We refer to [32, 33] for the validity of the above kernel functions.

2.2.3 Random Fourier Features

In this part, we introduce the algorithm of generating the random Fourier features [34], which is popular for accelerating the computation of kernel values. The basic idea is to find a randomized feature map $\hat{\psi} : \mathbb{R}^n \to \mathbb{R}^D$, so that the Euclidean inner product between the transformed points will approximate their kernel evaluations, i.e.,

$$\langle \hat{\psi}(\vec{\boldsymbol{x}}), \hat{\psi}(\vec{\boldsymbol{y}}) \rangle \approx k(\vec{\boldsymbol{x}}, \vec{\boldsymbol{y}}), \forall \vec{\boldsymbol{x}}, \vec{\boldsymbol{y}} \in \mathbb{R}^n. \tag{2.10}$$

It was shown that if the kernel function is shift-invariant, i.e., $k(\vec{\boldsymbol{x}}, \vec{\boldsymbol{y}}) = k(\vec{\boldsymbol{x}} - \vec{\boldsymbol{y}})$, then $\hat{\psi}$ is the random Fourier feature map [35, 34]. The procedure is detailed in Algorithm 1.

Note that if k is the Gaussian kernel, i.e., $k(\vec{\boldsymbol{x}}, \vec{\boldsymbol{y}}) = \exp(-\gamma\|\vec{\boldsymbol{x}} - \vec{\boldsymbol{y}}\|^2)$, then its Fourier transform p is a Gaussian distribution, i.e., $p(\omega) = N(\vec{\mathbf{0}}, \gamma^2 \boldsymbol{I})$.

Algorithm 1 Random Fourier Features

1: Input: a shift-invariant kernel k.
2: Output: a randomized feature map $\hat{\psi} : \mathbb{R}^n \to \mathbb{R}^D$, so that $\langle \hat{\psi}(\vec{\boldsymbol{x}}), \hat{\psi}(\vec{\boldsymbol{y}}) \rangle \approx k(\vec{\boldsymbol{x}} - \vec{\boldsymbol{y}})$.
3: Procedure:
4: step 1: Compute the Fourier transform p of the kernel, i.e., $p(\omega) = \frac{1}{2\pi} \int e^{-j\omega^T \delta} k(\delta).\mathrm{d}\delta$
5: step 2: Draw D i.i.d. samples $\omega_1, \omega_2, ..., \omega_D \in \mathbb{R}^n$ from p, and D i.i.d. samples $b1, b2, ..., b_D \in \mathbb{R}$ from the uniform distribution on $[0, 2\pi]$.
6: step 3: $\hat{\psi}(\vec{\boldsymbol{x}}) = \sqrt{\frac{2}{D}} \big[\cos(\omega_1^T \vec{\boldsymbol{x}} + b_1), ..., \cos(\omega_D^T \vec{\boldsymbol{x}} + b_D) \big]^T$.

2.2.4 Kernel-based Machine Learning Methods

In this part, we briefly introduce two supervised kernel-based machine learning methods [36]: kernel ridge regression and kernel support vector machine. We note that for both methods, we need to have access to only the pairwise kernel values and don't need to care about the data representation forms.

Let $(x_1^{\text{tr}}, y_1^{\text{tr}}), (x_2^{\text{tr}}, y_2^{\text{tr}}), ..., (x_n^{\text{tr}}, y_n^{\text{tr}})$ be the training samples, where x_i^{tr} and y_i^{tr} are the sample and corresponding label, respectively. Let $x_1^{\text{te}}, x_2^{\text{te}}, ..., x_m^{\text{te}}$ be the testing samples. For convenience, we write $\vec{\boldsymbol{y}}^{\text{tr}} = [y_1^{\text{tr}}, y_2^{\text{tr}}, ..., y_n^{\text{tr}}]^T$. The goal is to predict their labels. Let $\boldsymbol{K}_{rr} \in \mathbb{R}^{n \times n}$ be the kernel matrix of training samples, i.e., $[\boldsymbol{K}_{rr}]_{ij} = k(x_i^{\text{tr}}, x_j^{\text{tr}})$, and let $\boldsymbol{K}_{re} \in \mathbb{R}^{n \times m}$ be the cross kernel matrix of training and testing samples, i.e., $[\boldsymbol{K}_{re}]_{ij} = k(x_i^{\text{tr}}, x_j^{\text{te}})$.

For the kernel ridge regression, predicted labels $\hat{\vec{\boldsymbol{y}}}^{\text{te}} = [\hat{y}_1^{\text{te}}, \hat{y}_2^{\text{te}}, ..., \hat{y}_m^{\text{te}}]^T$ is

$$\hat{\vec{\boldsymbol{y}}}^{\text{te}} = \boldsymbol{K}_{re}(\boldsymbol{K}_{rr} + \gamma \boldsymbol{I})^{-1} \vec{\boldsymbol{y}}^{\text{tr}}. \tag{2.11}$$

For the kernel support vector machine, after training a classifier based on $(\boldsymbol{K}_{rr}, \vec{\boldsymbol{y}}^{\text{tr}})$, we can make prediction of the testing samples by

$$\hat{\vec{\boldsymbol{y}}}^{\text{te}} = \boldsymbol{K}_{re}(\vec{\boldsymbol{\alpha}} \circ \vec{\boldsymbol{y}}^{\text{tr}}) + \vec{\boldsymbol{b}}, \tag{2.12}$$

where $\circ$ denotes the Hadamard product, $\vec{\boldsymbol{\alpha}}$ is the Lagrangian multiplier and $\vec{\boldsymbol{b}}$ is the bias.

Chapter 3

RetGK: Graph Kernels Based on Return Probabilities of Random Walks

3.1 Introduction

In this chapter, we develop our return probability-based graph kernels. We first describe the general challenges and related works of graph kernels.

3.1.1 Challenges

1. When designing graph kernels, one might come across the graph isomorphism problem, a well-known NP problem. The kernels should satisfy the isomorphism-invariant property, while being informative on the topological structure difference.

2. Graphs are usually coupled with multiple types of node attributes, e.g., discrete[3] or continuous attributes. For example, a chemical compound may have both discrete and continuous attributes, which respectively describe the type and position of atoms. A crucial problem is how to integrate the graph structure and node attribute information in graph kernels.

3. In some applications, e.g., social networks, graphs tend to be very large, with thousands or even millions of nodes, which requires strongly scalable graph kernels.

3.1.2 Related Works

There are various graph kernels, many of which explore the R-convolutional framework [37]. The key idea is decomposing a whole graph into small substructures and building graph kernels based on the similarities among these components. Such kernels differ from each other in the way they decompose graphs. For example, graphlet kernels [38] are based on small subgraphs up to a fixed size. Weisfeiler-Lehman graph kernels [39] and tree-based kernels [40] are developed with subtree patterns. Shortest path kernels [41] are derived by comparing the paths between graphs. Still other graph kernels, such as [9] and [42], are developed by counting the number of common random walks on direct product graphs. Recently, subgraph matching kernels [43] and graph invariant kernels [44] were proposed for handling continuous attributes. However, all the above R-convolution based graph kernels suffer from a drawback. As pointed out in [45], increasing the size of substructures will largely decrease the probability that two graphs contain similar substructures, which usually results in the "diagonal dominance issue" [46]. Our return probability based kernels are significantly

[3]In the literature, the discrete node attributes are usually called "labels".

different from the above ones. We measure the similarity between two graphs by directly comparing their node structural role distributions, avoiding substructures decomposition.

More recently, new methods have been proposed for comparing graphs, which is done by quantifying the dissimilarity between the distributions of pairwise distances between nodes. [8] uses the shortest path distance, and [47] uses the diffusion distance. However, these methods can be applied only to non-attributed (unlabeled) graphs, which largely limits their applications in the real world.

Organization. In Section 3.2, we introduce the random walk on graphs and basic concept about tensor algebra. In Section 3.3, we discuss the favorable properties of and computational methods for return probability features. In Section 3.4, we present the Hilbert space embedding of graphs, and develop the corresponding graph kernels. In Section 3.5, we show the tensor representation of graphs, and derive computational efficient graph kernels. In Section 3.6, we report the experimental results on 21 benchmark datasets.

3.2 Prerequisites

3.2.1 Random Walk on Graphs

Let $\mathcal{G}$ be a undirected and connected graph with the node set $\mathcal{V}$ and the edge set $\mathcal{E}$. Let $\boldsymbol{A}_G$ and $\boldsymbol{D}_G$ be the adjacency matrix and degree matrix of $\mathcal{G}$, respectively (see Chapter 2.1 for the definition).

An S-step walk starting from node v_0 is a sequence of nodes $\{v_0, v_1, v_2, ..., v_S\}$, with $(v_s, v_{s+1}) \in \mathcal{E}, 0 \leq s \leq S-1$. A random walk on $\mathcal{G}$ is a Markov chain $(X_0, X_1, X_2, ...)$, whose transition

probabilities are

$$\Pr(X_{i+1} = v_{i+1}|X_i = v_i, ..., X_0 = v_0) = \Pr(X_{i+1} = v_{i+1}|X_i = v_i) = \frac{1}{\boldsymbol{D}_G(i,i)}, \tag{3.1}$$

which induces the transition probability matrix $\boldsymbol{P}_G = \boldsymbol{D}_G^{-1}\boldsymbol{A}_G$. More generally, $\boldsymbol{P}_G^s = \boldsymbol{P}_G \times \boldsymbol{P}_G \times ... \times \boldsymbol{P}_G$ is the s-step transition matrix, where $\boldsymbol{P}_G^s(i,j)$ is the transition probability in s steps from node v_i to v_j. The followings are the properties of random walks on graphs.

Proposition 3.1. *A random walk on a connected graph is a irreducible Markov chain, i.e., $\forall v_i, v_j \in \mathcal{V}$, there exist $t > 0$, such that $\boldsymbol{P}_G^t(i,j) > 0$.*

Proof. Because $\mathcal{G}$ is connected, for any two nodes v_i and v_j, there exist at least one path, $path(v_i \to v_j)$, starting from v_i and ending at v_j. Let t_0 be the length of $path(v_i \to v_j)$, then we immediately have $\boldsymbol{P}_G^{t_0}(i,j) > 0$. □

Proposition 3.2. *Let n be the number of nodes of $\mathcal{G}$ and let $\mathrm{Vol}_G = \sum_{i=1}^n \boldsymbol{D}_G(i,j)$ be the volume of $\mathcal{G}$. We define an n-dimensional vector $\vec{\boldsymbol{\pi}}$ such that its ith component $\vec{\boldsymbol{\pi}}_i$ satisfies $\vec{\boldsymbol{\pi}}_i = \frac{\boldsymbol{D}_G(i,i)}{\mathrm{Vol}_G}$, then $\vec{\boldsymbol{\pi}}$ is the unique stationary distribution, i.e., $\lim_{s\to+\infty} \boldsymbol{P}_G^s(i,j) = \vec{\boldsymbol{\pi}}_j$.*

Proof. It is sufficient to show that $\vec{\boldsymbol{\pi}}$ is the solution to the "Time-reversibility" set of equations [48],

$$\vec{\boldsymbol{\pi}}_i^T \boldsymbol{P}_G(i,j) = \vec{\boldsymbol{\pi}}_j^T \boldsymbol{P}_G(j,i). \tag{3.2}$$

The above equality is proved as follows.

$$\vec{\boldsymbol{\pi}}_i^T \boldsymbol{P}_G(i,j) = \frac{\boldsymbol{D}_G(i,i)}{\mathrm{Vol}_G} \times \frac{1}{\boldsymbol{D}_G(i,i)} = \frac{\boldsymbol{D}_G(j,j)}{\mathrm{Vol}_G} \times \frac{1}{\boldsymbol{D}_G(j,j)} = \vec{\boldsymbol{\pi}}_j^T \boldsymbol{P}_G(j,i). \tag{3.3}$$

□

3.2.2 Tensor Algebra

A tensor [49] is a multidimensional array, which has multiple indices.[4] We use $\mathbb{R}^{I_1 \times I_2 \times ... \times I_N}$ to denote the set of tensors of order N with dimension $(I_1, I_2, ..., I_N)$. If $U \in \mathbb{R}^{I_1 \times I_2 \times ... \times I_N}$, then $U_{i_1 i_2, ..., i_N} \in \mathbb{R}$, where $1 \leq i_1 \leq I_1, ..., 1 \leq i_N \leq I_N$.

The inner product between tensors $U, V \in \mathbb{R}^{I_1 \times I_2 \times ... \times I_N}$ is defined such that

$$\langle U, V \rangle_{\mathcal{T}} = \text{vec}(U)^T \text{vec}(V) = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} ... \sum_{i_N=1}^{I_N} U_{i_1 i_2, ..., i_N} V_{i_1 i_2, ..., i_N}. \tag{3.4}$$

A rank-one tensor $W \in \mathbb{R}^{I_1 \times I_2 \times ... \times I_N}$ is the tensor (outer) product of N vectors, $\vec{\boldsymbol{w}}^{(1)}, ..., \vec{\boldsymbol{w}}^{(N)}$. That is , $W = \vec{\boldsymbol{w}}^{(1)} \otimes \vec{\boldsymbol{w}}^{(2)} \otimes ... \otimes \vec{\boldsymbol{w}}^{(N)}$, $W_{i_1 i_2, ..., i_N} = \vec{\boldsymbol{w}}^{(1)}_{i_1} \vec{\boldsymbol{w}}^{(2)}_{i_2} ... \vec{\boldsymbol{w}}^{(N)}_{i_N}$.

3.3 Return Probabilities of Random Walks

Given a graph $\mathcal{G}$, as we can see from (3.1), the transition probability matrix, $\boldsymbol{P}_G$, encodes all the connectivity information, which leads to a natural intuition: We can compare two graphs by quantifying the difference between their transition probability matrices. However, big technical difficulties exist, since the sizes of two matrices are not necessarily the same, and their rows or columns do not correspond in most cases.

To tackle the above issues, we make use of the S-step return probabilities of random walks on $\mathcal{G}$. To do this, we assign each node $v_i \in \mathcal{V}$ an S-dimensional feature called "return

[4] A vector $\vec{\boldsymbol{u}} \in \mathbb{R}^D$ is a first-order tensor, and a matrix $\boldsymbol{A} \in \mathbb{R}^{D_1 \times D_2}$ is a second-order tensor.

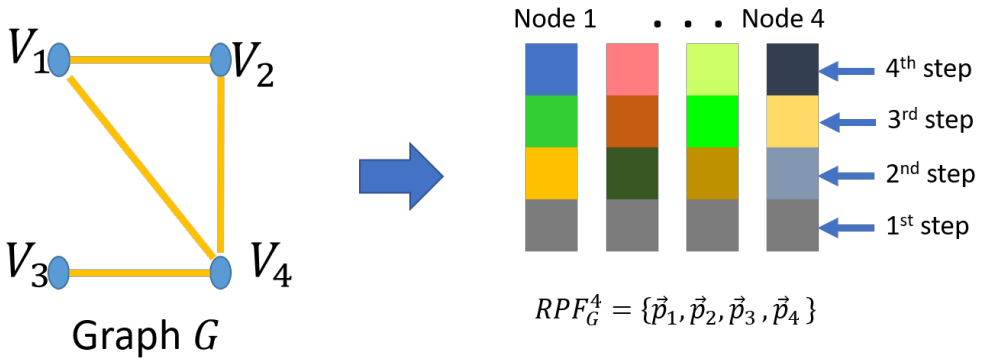


Figure 3.1: Left: a graph $\mathcal{G}$ of four nodes. Right: 4-dimensional return probability feature vector set, RPF_G^4.

probability feature" ("RPF" for short), which describes the "structural role" of v_i, i.e.,

$$\vec{\boldsymbol{p}}_i = [\boldsymbol{P}_G^1(i,i), \boldsymbol{P}_G^2(i,i), ..., \boldsymbol{P}_G^S(i,i)]^T, \tag{3.5}$$

where $\boldsymbol{P}_G^s(i,i)$, $s = 1, 2, ..., S$, is the return probability of a s-step random walk starting from v_i. Now each graph is represented by a set of feature vectors in $\mathbb{R}^S$:

$$\text{RPF}_G^S = \{\vec{\boldsymbol{p}}_1, \vec{\boldsymbol{p}}_2, ..., \vec{\boldsymbol{p}}_n\}. \tag{3.6}$$

In Fig. 3.1, we use a toy example to visualize the set of return probability feature vectors of a graph $\mathcal{G}$.

3.3.1 Properties of RPF

The RPF has three nice properties: isomorphism-invariance, multi-resolution, and informativeness.

Isomorphism-invariance The isomorphism-invariance property of return probability features is summarized in the following proposition.

Proposition 3.3. *Let G and H be two isomorphic graphs of n nodes, and let $\tau : \{1, 2, ..., n\} \to \{1, 2, ..., n\}$ be the corresponding isomorphism. Then,*

$$\forall v_i \in V_G,\ s = 1, 2, ..., \infty,\ \boldsymbol{P}_G^s(i, i) = \boldsymbol{P}_H^s(\tau(i), \tau(i)). \tag{3.7}$$

Proof. Let $\boldsymbol{\Pi}$ be the permutation matrix induced by τ, i.e., $\boldsymbol{\Pi}(i, j) = \delta_{j=\tau(i)}$, then, $\boldsymbol{\Pi}^T\boldsymbol{\Pi} = \boldsymbol{\Pi}\boldsymbol{\Pi}^T = \boldsymbol{I}_n$. Since G and H are isomorphic, we have $\boldsymbol{A}_H(\tau(i), \tau(j)) = \boldsymbol{A}_G(i, j)$, and $\boldsymbol{D}_H(\tau(i), \tau(i)) = \boldsymbol{D}_G(i, i)$, which is equivalent with $\boldsymbol{A}_H = \boldsymbol{\Pi}^T\boldsymbol{A}_G\boldsymbol{\Pi}$ and $\boldsymbol{D}_H = \boldsymbol{\Pi}^T\boldsymbol{D}_G\boldsymbol{\Pi}$.

Then $\boldsymbol{P}_H = \boldsymbol{D}_H^{-1}\boldsymbol{A}_H = (\boldsymbol{\Pi}^T\boldsymbol{D}_G\boldsymbol{\Pi})^{-1}(\boldsymbol{\Pi}^T\boldsymbol{A}_G\boldsymbol{\Pi}) = \boldsymbol{\Pi}^T\boldsymbol{P}_G\boldsymbol{\Pi}$. So $\boldsymbol{P}_H^s = (\boldsymbol{\Pi}^T\boldsymbol{P}_G\boldsymbol{\Pi})^s = \boldsymbol{\Pi}^T\boldsymbol{P}_G^s\boldsymbol{\Pi}$, which implies $\boldsymbol{P}_G^s(i, i) = \boldsymbol{P}_H^s(\tau(i), \tau(i))$. □

Clearly, isomorphic graphs have the same set of RPF, i.e., $\text{RPF}_G^S = \text{RPF}_H^S$, $\forall S = 1, 2, ..., \infty$. Such a property can be used to check graph isomorphism, i.e., if $\exists S$, s.t. $\text{RPF}_G^S \neq \text{RPF}_H^S$, then G and H are not isomorphic. Moreover, Proposition 3.3 allows us to directly compare the structural role of any two nodes in different graphs, without considering the matching problems.

Multi-resolution RPF characterizes the "structural role" of nodes with multi-resolutions. Roughly speaking, $\boldsymbol{P}_G^s(i, i)$ reflects the interaction between node v_i and the subgraph involving v_i. With an increase in s, the subgraph becomes larger. We use a toy example to illustrate our idea. Fig. 3.2(a) presents an unweighted graph $\mathcal{G}$, and C_1, C_2, and C_3 are three center nodes in $\mathcal{G}$, which play different structural roles. In Fig. 3.2(b), we plot their s-step return probabilities, $s = 1, 2, ..., 200$. C_1, C_2, and C_3 have the same degree, as do their

neighbors. Thus their first two return probabilities are the same. Since C_1 and C_2 share the similar neighbourhoods at larger scales, their return probability values are close until the eighth step. Because C_3 plays a very different structural role from C_1 and C_2, its return probabilities values deviate from those of C_1 and C_2 in early steps.

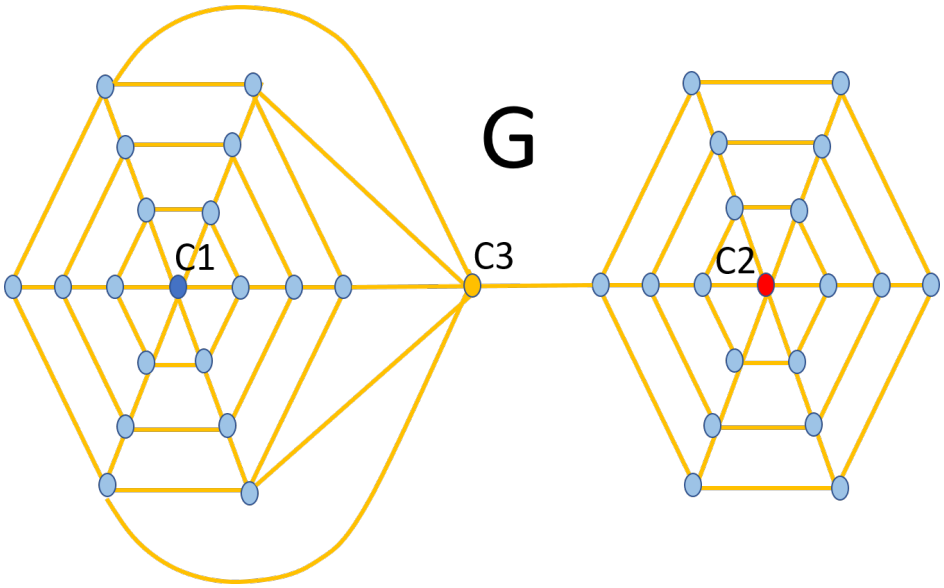
In addition, as shown in Fig. 3.2(b), when the random walk step s approaches infinity, the return probability $\boldsymbol{P}_G^s(i,i)$ will not change much and will converge to a certain value, which is known as the stationary probability as we discussed in Proposition 3.2. Therefore, if s is already sufficiently large, we gain very little new information from the RPF by increasing s.

Informativeness The RPF provides very rich information on the graph structure, in the sense that if two graphs has the same RPF sets, they share very similar spectral properties.

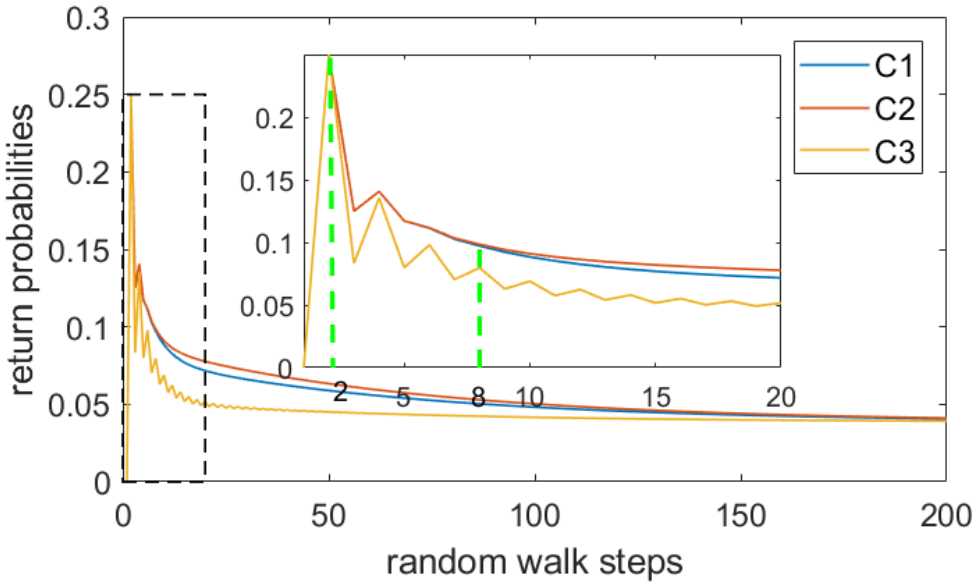
Theorem 3.1. *Let G and H be two connected graphs of the same size n and volume* Vol*, and let $\boldsymbol{P}_G$ and $\boldsymbol{P}_H$ be the corresponding transition probability matrices. Let $\{(\lambda_k, \vec{\boldsymbol{\psi}}_k)\}_{k=1}^n$ and $\{(\mu_k, \vec{\boldsymbol{\varphi}}_k)\}_{k=1}^n$ be eigenpairs of $\boldsymbol{P}_G$ and $\boldsymbol{P}_H$, respectively. Let $\tau : \{1,2,...,n\} \to \{1,2,...,n\}$ be a permutation map. If $\boldsymbol{P}_G^s(i,i) = \boldsymbol{P}_H^s(\tau(i),\tau(i)), \forall v_i \in V_G, \forall s = 1,2,...,n$, i.e.,* $\mathrm{RPF}_G^n = \mathrm{RPF}_H^n$*, then,*

1. $\mathrm{RPF}_G^S = \mathrm{RPF}_H^S$, $\forall S = n+1, n+2, ..., \infty$;
2. $\{\lambda_1, \lambda_2, ..., \lambda_n\} = \{\mu_1, \mu_2, ..., \mu_n\}$;
3. *If the eigenvalues sorted by their magnitudes satisfy:* $|\lambda_1| > |\lambda_2| > ... > |\lambda_m| > 0$, $|\lambda_{m+1}| = ... = |\lambda_n| = 0$, *then we have that* $|\vec{\boldsymbol{\psi}}_k(i)| = |\vec{\boldsymbol{\varphi}}_k(\tau(i))|$, $\forall v_i \in V_G$, $\forall k = 1,2,...,m$.

Proof. See Appendix A. □



(a)



(b)

Figure 3.2: (a) Toy Graph $\mathcal{G}$; (b) The s-step return probability of the nodes C_1, C_2 and C_3 in the toy graph, $s = 1, 2, ..., 200$. The nested figure is a close-up view of the rectangular region.

The first conclusion states that the graph structure information contained in RPF_G^n and RPF_G^S, $S \geq n$ are the same, coinciding with our previous discussions on RPF with large random walk steps. The second and third conclusions bridge the RPF with spectral representations of graphs [50], which contains almost all graph structure information.

Relation to eigenvector embeddings (EE) One popular way of embedding graph nodes in a Euclidean space uses the eigenvectors of Laplacian or adjacent matrices as the coordinates. In [51], a class of graph kernels are developed based on the eigenvector embeddings. From Theorem 3.1, we see that both RPF and EE encode the spectral information of graphs. However, our RPF has several advantages over EE. (i) The eigenvector embeddings reflect the closeness among nodes in the same graph, which makes it difficult to compare node across graphs. (ii) The EE representations, which are computed up to a change in sign (or more generally, orthonormal transformation in the eigenspace), may not be invariant under graph isomorphisms. A counterexample is shown in Fig. 3.3. $\mathcal{G}$ and $\mathcal{G}'$ are two isomorphic graphs, we visualize their first three-dimensional embeddings with RPF and EE [5]. It can be seen that RPFs are invariant while eigenvectors are not. (iii) The eigenvector embeddings are unstable. The perturbation theory says that two eigenvectors may switch if their eigenvalues are close.

3.3.2 Computation of RPF

Given a graph $\mathcal{G}$, the brute-force computation of RPF_G^S requires $(S-1) \times n \times n$ matrix multiplication of $\boldsymbol{P}_G$. Therefore, the time complexity is $(S-1)n^3$, which is quite high when S is large.

[5] Note that since the signs of these eigenvectors are not fixed, we use the absolute value as in [51]

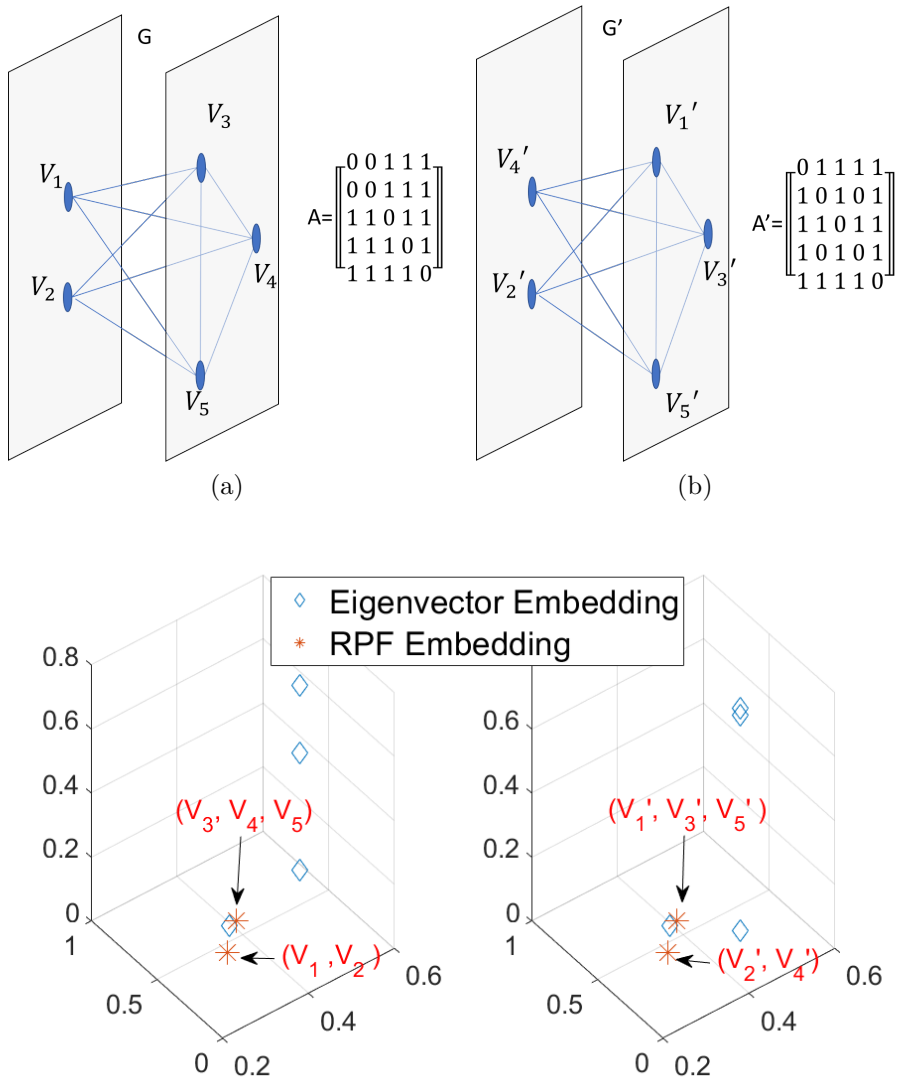


Figure 3.3: Toy graph $\mathcal{G}$ and its adjacent matrix; (b) Toy graph $\mathcal{G}'$ and its adjacent matrix; (c) 3-D eigenvector and RPF embeddings of nodes in $\mathcal{G}$ and $\mathcal{G}'$, respectively. We can see that our RPF correctly reflects the structural roles. That is, the nodes V_3, V_4, V_5 in graph G and the nodes V_1', V_3', V_5' in graph $\mathcal{G}'$ have the same structural role. And the nodes V_1, V_2 in graph $\mathcal{G}$ and the nodes V_2', V_4' in graph $\mathcal{G}'$ have the same structural role.

Since only the diagonal terms of transition matrices are needed, we have efficient techniques. Write

$$\boldsymbol{P}_G = \boldsymbol{D}_G^{-1}\boldsymbol{A}_G = \boldsymbol{D}_G^{-\frac{1}{2}}(\boldsymbol{D}_G^{-\frac{1}{2}}\boldsymbol{A}_G\boldsymbol{D}_G^{-\frac{1}{2}})\boldsymbol{D}_G^{\frac{1}{2}} = \boldsymbol{D}_G^{-\frac{1}{2}}\boldsymbol{B}_G\boldsymbol{D}_G^{\frac{1}{2}}, \tag{3.8}$$

where $\boldsymbol{B}_G = \boldsymbol{D}_G^{-\frac{1}{2}}\boldsymbol{A}_G\boldsymbol{D}_G^{-\frac{1}{2}}$ is a symmetric matrix. Then $\boldsymbol{P}_G^s = \boldsymbol{D}_G^{-\frac{1}{2}}\boldsymbol{B}_G^s\boldsymbol{D}_G^{\frac{1}{2}}$. Let $\{(\lambda_k, \vec{\boldsymbol{u}}_k)\}_{k=1}^n$ be the eigenpairs of $\boldsymbol{B}_G$, i.e., $\boldsymbol{B}_G = \sum_{k=1}^n \lambda_k \vec{\boldsymbol{u}}_k \vec{\boldsymbol{u}}_k^T$. Then the return probabilities are

$$\boldsymbol{P}_G^s(i,i) = \boldsymbol{B}_G^s(i,i) = \sum_{k=1}^{n} \lambda_k^s \big[\vec{\boldsymbol{u}}_k(i)\big]^2, \forall v_i \in V_G, \forall s = 1, 2, ..., S. \tag{3.9}$$

Let $\boldsymbol{U} = [\vec{\boldsymbol{u}}_1, \vec{\boldsymbol{u}}_2, ..., \vec{\boldsymbol{u}}_n]$, let $\boldsymbol{V} = \boldsymbol{U} \circ \boldsymbol{U}$, where $\circ$ denotes Hadamard product, and let $\vec{\boldsymbol{\Lambda}}_s = [\lambda_1^s, \lambda_2^s, ..., \lambda_n^s]^T$. Then we can obtain all nodes' s-step return probabilities in the vector $\boldsymbol{V}\vec{\boldsymbol{\Lambda}}_s$. The eigen-decomposition of $\boldsymbol{B}_G$ requires time $O(n^3)$. Computing $\boldsymbol{V}$ or $\boldsymbol{V}\vec{\boldsymbol{\Lambda}}_s$, $\forall s = 1, 2, ..., S$, takes time $O(n^2)$. So the total time complexity of the above computational method is $O\big(n^3 + (S+1)n^2\big)$.

Monte Carlo simulation method If the graph node number, n, is large, i.e., $n > 10^5$, the eigendecomposition of an $n \times n$ matrix is relatively time-consuming. To make RPF scalable to large graphs, we use the Monte Carlo method to simulate random walks. Given a graph $\mathcal{G}$, for each node $v_i \in V_G$, we can simulate a random walk of length S based on the transition probability matrix $\boldsymbol{P}_G$. We repeat the above procedure M times, obtaining M sequences of random walks. For each step $s = 1, 2, ..., S$, we use the relative frequency of returning to the starting point as the estimation of the corresponding s-step return probability. The random walk simulation is parallelizable and can be implemented efficiently, characteristics of which both contribute to the scalability of RPF.

3.4 Hilbert Space Embeddings of Graphs

In this section, we introduce the Hilbert space embeddings of graphs, based on the RPF. With such Hilbert space embeddings, we can naturally obtain the corresponding graph kernels.

As discussed in Section 3, the structural role of each node v_i can be characterized by an $S-$dimensional return probability vector $\vec{\boldsymbol{p}}_i$ (see (3.5)), and thus a nonattributed graph can be represented by the set $\mathrm{RPF}_G^S = \{\vec{\boldsymbol{p}}_i\}_{i=1}^n$. Since the isomorphism-invariance property allows direct comparison of nodes' structural roles across different graphs, we can view the RPF as a special type of attribute, namely, "the structural role attribute" (whose domain is denoted as $\mathcal{A}_0$), associated with nodes. Clearly, $\mathcal{A}_0 = \mathbb{R}^S$.

The nodes of attributed graphs usually have other types of attributes, which are obtained by physical measurements. Let $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_L$ be their attribute domains. When combined with RPF, an attributed graph can be represented by the set

$$\{(\vec{\boldsymbol{p}}_i, a_i^1, ..., a_i^L)\}_{i=1}^n \subseteq \mathcal{A}_0 \times \mathcal{A}_1 \times ... \times \mathcal{A}_L \quad (= \times_{l=0}^L \mathcal{A}_l). \tag{3.10}$$

Such a representation allows us to consider both attributed and nonattributed graphs in a unified framework, since if $L = 0$, the above set just degenerates to the nonattributed case. The set representation forms an empirical distribution

$$\mu = \frac{1}{n}\sum_{i=1}^n \delta_{(\vec{\boldsymbol{p}}_i, a_i^1, ..., a_i^L)} \quad \text{on} \quad \mathcal{A} = \times_{l=0}^L \mathcal{A}_l, \tag{3.11}$$

which can be embedded into a reproducing kernel Hilbert space (RKHS) by kernel mean embedding [52].

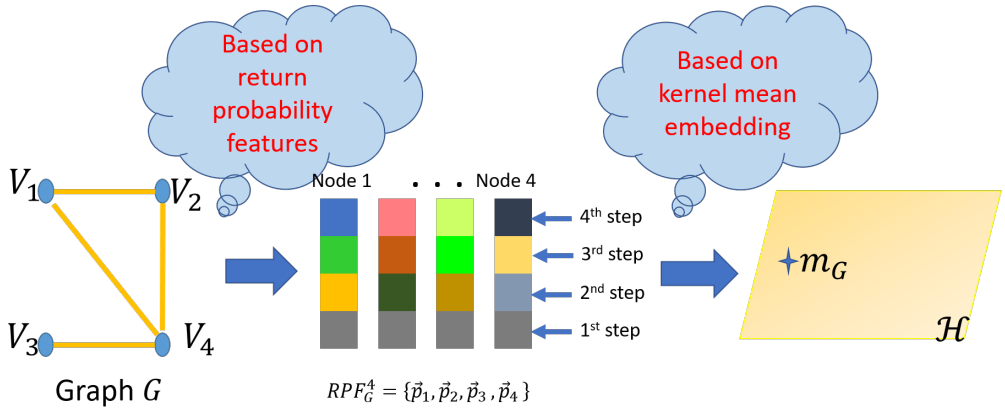


Figure 3.4: Summary of the two-step embeddings.

Let k_l, $l = 0, 1, ..., L$ be a kernel on $\mathcal{A}_l$. Let $\mathcal{H}_l$ and ϕ_l be the corresponding RKHS and implicit feature map, respectively. Then we can define a kernel on $\mathcal{A}$ through the tensor product of kernels [53], $k = \otimes_{l=0}^{L} k_l$, i.e.,

$$k\big[(\vec{\boldsymbol{p}}, a^1, a^2, ..., a^L), (\vec{\boldsymbol{q}}, b^1, b^2, ..., b^L)\big] = k_0(\vec{\boldsymbol{p}}, \vec{\boldsymbol{q}}) \prod_{l=1}^{L} k_l(a^l, b^l). \tag{3.12}$$

Its associated RKHS, $\mathcal{H}$, is the tensor product space generated by $\mathcal{H}_l$, i.e., $\mathcal{H} = \otimes_{l=0}^{L} \mathcal{H}_l$. Let $\phi : \mathcal{A} \to \mathcal{H}$ be the implicit feature map. Then given a graph G, we can embed it into $\mathcal{H}$ in the following procedure,

$$G \to \mu_G \to m_G, \text{ and } m_G = \int_{\mathcal{A}} \phi \mathrm{d}\mu_G = \frac{1}{n} \sum_{i=1}^{n} \phi(\boldsymbol{p}_i, a_i^1, ..., a_i^L). \tag{3.13}$$

Before proceeding to the computational graph kernels, we summarize our two-step graph embeddings. In the first step, we present the graph with a set of vectors, based on return probability features. In the second step, we further embed the set in a RKHS, based on the kernel mean embedding. Therefore, any graph (with node attributes) can be represented by a vector in a Hilbert space. The above procedures is sketched in Fig. 3.4.

3.4.1 Graph Kernels (I)

An important benefit of Hilbert space embedding of graphs is that it is straightforward to generalize the positive definite kernels defined on Euclidean spaces to the set of graphs.

Given two graphs G and H, let $\{\triangle_i^G\}_{i=1}^{n_G}$ and $\{\triangle_i^H\}_{j=1}^{n_H}$ be the respective set representations $\big(\triangle_i^G = (\vec{\boldsymbol{p}}_i, a_i^1, a_i^2, ..., a_i^L)$ and likewise $\triangle_j^H\big)$. Let $\boldsymbol{K}_{GG}$, $\boldsymbol{K}_{HH}$, and $\boldsymbol{K}_{GH}$ be the kernel matrices, induced by the embedding kernel k. That is, they are defined such that $(\boldsymbol{K}_{GG})_{ij} = k(\triangle_i^G, \triangle_j^G)$, $(\boldsymbol{K}_{HH})_{ij} = k(\triangle_i^H, \triangle_j^H)$, and $(\boldsymbol{K}_{GH})_{ij} = k(\triangle_i^G, \triangle_j^H)$.

Proposition 3.4. *Let $\mathcal{X}_G$ be the set of graphs with attribute domains $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_L$. Let G and H be two graphs in $\mathcal{X}_G$. Let m_G and m_H be the corresponding graph embeddings. Then the following functions are positive definite graph kernels defined on $\mathcal{X}_G \times \mathcal{X}_G$.*

$$K_1(G, H) = (c + \langle m_G, m_H \rangle_{\mathcal{H}})^d = (c + \frac{1}{n_G n_H} \vec{\mathbf{1}}_{n_G}^T \boldsymbol{K}_{GH} \vec{\mathbf{1}}_{n_H})^d, c \geq 0, d \in \mathbb{N}, \tag{3.14a}$$

$$K_2(G, H) = \exp(-\gamma \|m_G - m_H\|_{\mathcal{H}}^p) = \exp\big[-\gamma \mathrm{MMD}^p(\mu_G, \mu_H)\big], \gamma > 0, 0 < p \leq 2, \tag{3.14b}$$

where

$$\mathrm{MMD}(\mu_G, \mu_H) = (\frac{1}{n_G^2} \vec{\mathbf{1}}_{n_G}^T \boldsymbol{K}_{GG} \vec{\mathbf{1}}_{n_G} + \frac{1}{n_H^2} \vec{\mathbf{1}}_{n_H}^T \boldsymbol{K}_{HH} \vec{\mathbf{1}}_{n_H} - \frac{2}{n_G n_H} \vec{\mathbf{1}}_{n_G}^T \boldsymbol{K}_{GH} \vec{\mathbf{1}}_{n_H})^{\frac{1}{2}}$$

is the maximum mean discrepancy (MMD) [52].

Proof. **(a).** We first consider two kernels $K_\alpha(G, H) = \langle m_G, m_H \rangle_{\mathcal{H}}$ and $K_\beta(G, K) = c$. It can be easily observed that K_α and K_β are positive definite graph kernels. Since the sum and multiplication of positive definite kernels are still positive definite, we conclude that (3.14a) are positive definite.

(b). The positive definiteness of (3.14b) is obtained from Corollary 3 in [33]. □

Kernel selection In real applications, such as bioinformatics, graphs may have discrete labels and (multi-dimensional) real-valued attributes. Hence, three attributes domains are involved in the computation of our graph kernels: the structural role attribute domain $\mathcal{A}_0$, the discrete attribute domain $\mathcal{A}_d$, and the continuous attribute domain $\mathcal{A}_c$. For $\mathcal{A}_d$, we can use the Delta kernel $k_d(a,b) = I_{\{a=b\}}$. For $\mathcal{A}_0$ and $\mathcal{A}_c$, which are just the Euclidean spaces, we can use the Gaussian RBF kernel, the Laplacian RBF kernel, or the polynomial kernel.

3.5 Approximated Hilbert Space Embedding of Graphs

Based on the above discussions, we see that obtaining a graph kernel value between each pair of graphs requires calculating the inner product or the L_2 distance between two Hilbert embeddings (see (3.14a) and (3.14b)), both of which scale quadratically to the node numbers. Such time complexity precludes application to large graph datasets. To tackle the above issues, we employ the approximate explicit feature maps.

For a kernel k_l on the attribute domain $\mathcal{A}_l$, $l = 0, 1, ..., L$, we find an explicit map $\hat{\phi} : \mathcal{A}_l \to \mathbb{R}^{D_l}$, so that

$$\forall a, b \in \mathcal{A}_l, \langle \hat{\phi}(a), \hat{\phi}(b) \rangle = \hat{k}_l(a,b), \text{ and } \hat{k}_l(a,b) \to k_l(a,b) \text{ as } D_l \to \infty. \tag{3.15}$$

The explicit feature maps will be directly used to compute the approximate graph embeddings, by virtue of tensor algebra (see Section 3.2.2). The following theorem says that the

approximate explicit graph embeddings can be written as the linear combination of rank-one tensors.

Theorem 3.2. *Let G and H be any two graphs in $\mathcal{X}_G$. Let $\{(\vec{\boldsymbol{p}}_i, a_i^1, a_i^2, ..., a_i^L)\}_{i=1}^{n_G}$ and $\{(\vec{\boldsymbol{q}}_j, b_j^1, b_j^2, ..., b_j^L)\}_{j=1}^{n_H}$ be the respective set representations of G and H. Then their approximate explicit graph embeddings, $\hat{m}_G$ and $\hat{m}_H$, are tensors in $\mathbb{R}^{D_0 \times D_1 \times ... \times D_L}$, and can be written as*

$$\hat{m}_G = \frac{1}{n_G} \sum_{i=1}^{n_G} \hat{\phi}_0(\vec{\boldsymbol{p}}_i) \otimes \hat{\phi}_1(a_i^1) \otimes ... \otimes \hat{\phi}_L(a_i^L), \quad \hat{m}_H = \frac{1}{n_H} \sum_{j=1}^{n_H} \hat{\phi}_0(\vec{\boldsymbol{q}}_j) \otimes \hat{\phi}_1(b_j^1) \otimes ... \otimes \hat{\phi}_L(b_j^L). \tag{3.16}$$

That is, as $D_0, D_1, ..., D_L \to \infty$, we have $\langle \hat{m}_G, \hat{m}_H \rangle_{\mathcal{T}} \to \langle m_G, m_H \rangle_{\mathcal{H}}$.

Before proceeding to the main proof, we first introduce a lemma about the inner product of multi-dimensional tensors.

Lemma 3.1. *Let $U = \vec{\boldsymbol{u}}^{(0)} \otimes \vec{\boldsymbol{u}}^{(1)} \otimes ... \otimes \vec{\boldsymbol{u}}^{(L)}$ and $V = \vec{\boldsymbol{v}}^{(0)} \otimes \vec{\boldsymbol{v}}^{(1)} \otimes ... \otimes \vec{\boldsymbol{v}}^{(L)}$ be two rank-one tensors in $\mathbb{R}^{D_0 \times D_1 \times ... \times D_L}$. Then we have $\langle U, V \rangle_{\mathcal{T}} = \langle \vec{\boldsymbol{u}}^{(0)}, \vec{\boldsymbol{v}}^{(0)} \rangle \langle \vec{\boldsymbol{u}}^{(1)}, \vec{\boldsymbol{v}}^{(1)} \rangle ... \langle \vec{\boldsymbol{u}}^{(L)}, \vec{\boldsymbol{v}}^{(L)} \rangle$.*

Proof.

$$\begin{aligned}
&\langle U, V \rangle_{\mathcal{T}} \\
=&\sum_{i_0=1}^{D_0} \sum_{i_1=1}^{D_1} ... \sum_{i_L=1}^{D_L} U_{i_1 i_2, ..., i_L} V_{i_1 i_2, ..., i_L} \\
=&\sum_{i_0=1}^{D_0} \sum_{i_1=1}^{D_1} ... \sum_{i_L=1}^{D_L} \vec{\boldsymbol{u}}_{i_0}^{(0)} \vec{\boldsymbol{u}}_{i_1}^{(1)} ... \vec{\boldsymbol{u}}_{i_L}^{(L)} \vec{\boldsymbol{v}}_{i_0}^{(0)} \vec{\boldsymbol{v}}_{i_1}^{(1)} ... \vec{\boldsymbol{v}}_{i_L}^{(L)} \\
=&(\sum_{i_0=1}^{D_0} \vec{\boldsymbol{u}}_{i_0}^{(0)} \vec{\boldsymbol{v}}_{i_0}^{(0)}) (\sum_{i_1=1}^{D_1} \vec{\boldsymbol{u}}_{i_1}^{(1)} \vec{\boldsymbol{v}}_{i_1}^{(1)}) ... (\sum_{i_L=1}^{D_L} \vec{\boldsymbol{u}}_{i_L}^{(L)} \vec{\boldsymbol{v}}_{i_L}^{(L)}) \\
=&\langle \vec{\boldsymbol{u}}^{(0)}, \vec{\boldsymbol{v}}^{(0)} \rangle \langle \vec{\boldsymbol{u}}^{(1)}, \vec{\boldsymbol{v}}^{(1)} \rangle ... \langle \vec{\boldsymbol{u}}^{(L)}, \vec{\boldsymbol{v}}^{(L)} \rangle.
\end{aligned} \tag{3.17}$$

□

Now we prove theorem 3.2.

Proof. We first calculate $\langle \hat{m}_G, \hat{m}_H \rangle_{\mathcal{T}}$.

$$
\begin{aligned}
&\langle \hat{m}_G, \hat{m}_H \rangle_{\mathcal{T}} \\
=&\Big\langle \frac{1}{n_G} \sum_{i=1}^{n_G} \hat{\phi}_0(\vec{\boldsymbol{p}}_i) \otimes \hat{\phi}_1(a_i^1) \otimes ... \otimes \hat{\phi}_L(a_i^L), \frac{1}{n_H} \sum_{j=1}^{n_H} \hat{\phi}_0(\vec{\boldsymbol{q}}_j) \otimes \hat{\phi}_1(b_j^1) \otimes ... \otimes \hat{\phi}_L(b_j^L) \Big\rangle_{\mathcal{T}} \\
=&\frac{1}{n_G n_H} \sum_{i=1}^{n_G} \sum_{j=1}^{n_H} \Big\langle \hat{\phi}_0(\vec{\boldsymbol{p}}_i) \otimes \hat{\phi}_1(a_i^1) \otimes ... \otimes \hat{\phi}_L(a_i^L), \hat{\phi}_0(\vec{\boldsymbol{q}}_j) \otimes \hat{\phi}_1(b_j^1) \otimes ... \otimes \hat{\phi}_L(b_j^L) \Big\rangle_{\mathcal{T}} \\
=&\frac{1}{n_G n_H} \sum_{i=1}^{n_G} \sum_{j=1}^{n_H} \langle \hat{\phi}_0(\vec{\boldsymbol{p}}_i), \hat{\phi}_0(\vec{\boldsymbol{q}}_j) \rangle \langle \hat{\phi}_1(a_i^1), \hat{\phi}_1(b_j^1) \rangle ... \langle \hat{\phi}_1(a_i^L), \hat{\phi}_1(b_j^L) \rangle \\
=&\frac{1}{n_G n_H} \sum_{i=1}^{n_G} \sum_{j=1}^{n_H} \hat{k}_0(\vec{\boldsymbol{p}}_i, \vec{\boldsymbol{q}}_j) \hat{k}_1(a_i^1, b_j^1) ... \hat{k}_L(a_i^L, b_j^L),
\end{aligned}
\tag{3.18}
$$

where the 3rd equality holds because of lemma 3.1.

We next calculate $\langle m_G, m_H \rangle_{\mathcal{H}}$.

$$
\begin{aligned}
&\langle m_G, m_H \rangle_{\mathcal{H}} \\
=&\langle \frac{1}{n_G} \sum_{i=1}^{n_G} \phi(\boldsymbol{p}_i, a_i^1, ..., a_i^L), \frac{1}{n_H} \sum_{j=1}^{n_H} \phi(\boldsymbol{q}_j, b_j^1, ..., b_j^L) \rangle_{\mathcal{H}} \\
=&\frac{1}{n_G n_H} \sum_{i=1}^{n_G} \sum_{j=1}^{n_H} \langle \phi(\boldsymbol{p}_i, a_i^1, ..., a_i^L), \phi(\boldsymbol{q}_j, b_j^1, ..., b_j^L) \rangle_{\mathcal{H}} \\
=&\frac{1}{n_G n_H} \sum_{i=1}^{n_G} \sum_{j=1}^{n_H} k\big[(\boldsymbol{p}_i, a_i^1, ..., a_i^L), (\boldsymbol{q}_j, b_j^1, ..., b_j^L) \big] \\
=&\frac{1}{n_G n_H} \sum_{i=1}^{n_G} \sum_{j=1}^{n_H} k_0(\vec{\boldsymbol{p}}_i, \vec{\boldsymbol{q}}_j) k_1(a_i^1, b_j^1) ... k_L(a_i^L, b_j^L),
\end{aligned}
\tag{3.19}
$$

where the last equality holds because of the definition of the embedding kernel $k = \otimes_{l=0}^{L} k_l$. Since as $D_0, D_1, ..., D_L \to \infty$, $\hat{k}_0(\vec{\boldsymbol{p}}_i, \vec{\boldsymbol{q}}_j) \to k_0(\vec{\boldsymbol{p}}_i, \vec{\boldsymbol{q}}_j)$, $\hat{k}_1(a_i^1, b_j^1) \to k_1(a_i^1, b_j^1)$, ..., $\hat{k}_L(a_i^L, b_j^L) \to k_L(a_i^L, b_j^L)$, we conclude that $\langle \hat{m}_G, \hat{m}_H \rangle_{\mathcal{T}} \to \langle m_G, m_H \rangle_{\mathcal{H}}$. □

3.5.1 Graph Kernels (II)

With approximate tensor embeddings (3.16), we obtain new graph kernels.

Proposition 3.5. *The following functions are positive definite graph kernels defined on* $\mathcal{X}_G \times \mathcal{X}_G$.

$$\hat{K}_1(G, H) = (c + \langle \hat{m}_G, \hat{m}_H \rangle_{\mathcal{T}})^d = \left[c + \mathrm{vec}(\hat{m}_{\mathcal{G}})^T \mathrm{vec}(\hat{m}_{\mathcal{H}})\right]^d, c \geq 0, d \in \mathbb{N}, \tag{3.20a}$$

$$\hat{K}_2(G, H) = \exp(-\gamma \|\hat{m}_G - \hat{m}_H\|_{\mathcal{T}}^p) = \exp(-\gamma \|\mathrm{vec}(\hat{m}_G) - \mathrm{vec}(\hat{m}_H)\|_2^p), \gamma > 0, 0 < p \leq 2.. \tag{3.20b}$$

Moreover, as $D_0, D_1, ..., D_L \to \infty$, *we have* $\hat{K}_1(G, H) \to K_1(G, H)$ *and* $\hat{K}_2(G, H) \to K_2(G, H)$.

Proof. The positive definiteness of $\hat{K}_1$ and $\hat{K}_2$ can be proved in the same way with Proposition 3.4. The convergence property can be obtained by Theorem 3.2. □

The vectorization of $\hat{m}_G$ (or $\hat{m}_H$) can be easily implemented by the Kronecker product, i.e., $\mathrm{vec}(\hat{m}_G) = \frac{1}{n_G} \sum_{i=1}^{n_G} \mathrm{Kronecker}\big(\hat{\phi}_0(\vec{\boldsymbol{p}}_i), \hat{\phi}_1(a_i^1), ..., \hat{\phi}_L(a_i^L)\big)$. To obtain above graph kernels, we need only to compute the Euclidean inner product or distance between vectors. More notably, the size of the tensor representation does not depends on node numbers, making it scalable to large graphs.

Approximate explicit feature map selection. For the Delta kernel on the discrete attribute domain, we directly use the one-hot vector. For shift-invariant kernels, i.e., $k(\vec{\boldsymbol{x}}, \vec{\boldsymbol{y}}) = k(\vec{\boldsymbol{x}} - \vec{\boldsymbol{y}})$, on Euclidean spaces, e.g., $\mathcal{A}_0$ and $\mathcal{A}_c$, we make use of random Fourier feature map (see 2.2.3).

3.6 Experiments

In this section, we conduct extensive experiments to demonstrate the effectiveness of our graph kernels. We run all the experiments on a laptop with an Intel i7-7820HQ, 2.90GHz CPU and 64GB RAM. We implement our algorithms in Matlab, except for the Monte Carlo based computation of RPF (see Section 3.3.2), which is implemented in C++.

3.6.1 Datasets

We conduct graph classification on four types of benchmark datasets [54]. (i) Non-attributed (unlabeled) graphs datasets: COLLAB, IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, REDDIT-MULTI(5K), and REDDIT-MULTI(12K) [55] are generated from social networks. (ii) Graphs with discrete attributes (labels): DD [56] are proteins. MUTAG [57], NCI1 [39], PTC-FM, PTC-FR, PTC-MM, and PTC-MR [58] are chemical compounds. (iii) Graphs with continuous attributes: FRANK is a chemical molecule dataset [59]. SYNTHETIC and Synthie are synthetic datasets based on random graphs, which were first introduced in [60] and [61], respectively. (iv) Graphs with both discrete and continuous attributes: ENZYMES and PROTEINS [62] are graph representations of proteins. BZR, COX2, and DHFR [63]

are chemical compounds. Detailed descriptions of these 21 datasets are provided in the Appendix B.

3.6.2 Experimental Setup

We demonstrate both the graph kernels (I) and (II) introduced in Section 3.4.1 and Section 3.5.1, which are denoted by RetGK_{I} and RetGK_{II}, respectively. The Monte Carlo computation of return probability features, denoted by RetGK_{II}(MC), is also considered. In our experiments, we repeat 200 Monte Carlo trials, i.e., $M = 200$, for obtaining RPF. For handling the isolated nodes, whose degrees are zero, we artificially add a self-loop for each node in graphs.

Parameters In all experiments, we set the random walk step $S = 50$. For RetGK_{I}, we use the Laplacian RBF kernel for both the structural role domain $\mathcal{A}_0$, and the continuous attribute domain $\mathcal{A}_c$, i.e., $k_0(\vec{\boldsymbol{p}}, \vec{\boldsymbol{q}}) = \exp(-\gamma_0 \|\vec{\boldsymbol{p}} - \vec{\boldsymbol{q}}\|_2)$ and $k_c(\vec{\boldsymbol{a}}, \vec{\boldsymbol{b}}) = \exp(-\gamma_c \|\vec{\boldsymbol{a}} - \vec{\boldsymbol{b}}\|_2)$. We set γ_0 to be the inverse of the median of all pairwise distances, and set γ_c to be the inverse of the square root of the attributes' dimension, except for the FRANK dataset, whose γ_c is set to be the recommended value $\sqrt{0.0073}$ in the paper [44] and [61]. For RetGK_{II}, on the first three types of graphs, we set the dimensions of random Fourier feature maps on $\mathcal{A}_0$ and $\mathcal{A}_c$ both to be 200, i.e., $D_0 = D_c = 200$, except for the FRANK dataset, whose D_c is set to be 500 because its attributes lie in a much higher dimensional space. On the graphs with both discrete and continuous attributes, for the sake of computational efficiency, we set $D_0 = D_c = 100$. For both RetGK_{I} and RetGK_{II}, we make use of the graph kernels with exponential forms, $\exp(-\gamma\|\cdot\|^p)$, (see (3.14b) and (3.20b)). We select p from $\{1, 2\}$, and set $\gamma = \frac{1}{\text{dist}^p}$, where dist is the median of all the pairwise graph embedding distances.

We compare our graph kernels with many state-of-the-art graph classification algorithms:

1. the shortest path kernel (SP) [41],
2. the Weisfeiler-Lehman subtree kernel (WL) [39],
3. the graphlet count kernel (GK)[38],
4. deep graph kernels (DGK) [55],
5. PATCHY-SAN convolutional neural network (PSCN) [64],
6. deep graph convolutional neural network (DGCNN) [65],
7. graph invariant kernels (GIK) [44],
8. hashing Weisfeiler-Lehman graph kernels (HGK(WL)) [61],
9. subgraph matching kernels (CSM) [43].

For all kinds of graph kernels, we employ SVM [66] as the final classifier. The tradeoff parameter C is selected from $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3\}$. We perform 10-fold cross-validations, using 9 folds for training and 1 for testing, and repeat the experiments 10 times. We report average classification accuracies and standard errors.

3.6.3 Experimental Results

The classification results[6] on four types of datasets are shown in Tables 3.1, 3.2, 3.3, and 3.4. The best results are highlighted in bold. We also report the total time of computing the graph kernels of all the datasets in each table. It can be seen that graph kernels

[6]The accuracies of WL, SP and GK are obtained from our own experiments. For others competing algorithms, we directly quote the values from their papers.

RetGK_{I} and RetGK_{II} both achieve superior or comparable performance on all the benchmark datasets. Especially on the datasets COLLAB, REDDIT-BINARY, REDDIT-MULTI(12K), Synthie, BZR, COX2, our approaches significantly outperform other state-of-the-art algorithms. The classification accuracies of our approaches on these datasets are at least six percentage points higher than those of the best baseline algorithms. Moreover, we see that RetGK_{II} and RetGK_{II}(MC) are faster than baseline methods. Their running times remain perfectly practical. On the large social network datasets (see Table 3.1), RetGK_{II}(MC) is almost one order of magnitude faster than the Weisfeiler-Lehman subtree kernel, which is well known for its computational efficiency.

Table 3.1: Classification results (in %) for non-attributed (unlabeled) graph datasets

Datasets	WL	GK	DGK	PSCN	RetGK_{I}	RetGK_{II}	RetGK_{II}(MC)
COLLAB	74.8(0.2)	72.8(0.3)	73.1(0.3)	72.6(2.2)	**81.0(0.3)**	80.6(0.3)	73.6(0.3)
IMDB-BINARY	70.8(0.5)	65.9(1.0)	67.0(0.6)	71.0(2.3)	71.9(1.0)	**72.3(0.6)**	71.0(0.6)
IMDB-MULTI	**49.8(0.5)**	43.9(0.4)	44.6(0.5)	45.2(2.8)	47.7(0.3)	48.7(0.6)	46.7(0.6)
REDDIT-BINARY	68.2(0.2)	77.3(0.2)	78.0(0.4)	86.3(1.6)	**92.6(0.3)**	91.6(0.2)	90.8(0.2)
REDDIT-MULTI(5K)	51.2(0.3)	41.0(0.2)	41.3(0.2)	49.1(0.7)	**56.1(0.5)**	55.3(0.3)	54.2(0.3)
REDDIT-MULTI(12K)	32.6(0.3)	31.8(0.1)	32.2(0.1)	41.3(0.4)	**48.7(0.2)**	47.1(0.3)	45.9(0.2)
Total time	2h3m	–	–	–	48h14m	17m14s	**6m9s**

Table 3.2: Classification results (in %) for graph datasets with discrete attributes

Datasets	SP	WL	GK	CSM	DGCNN	PSCN	RetGK_{I}	RetGK_{II}
ENZYMES	38.6(1.5)	53.4(0.9)	–	**60.4(1.6)**	–	–	**60.4(0.8)**	59.1(1.1)
PROTEINS	73.3(0.9)	71.2(0.8)	71.7(0.6)	–	75.5(0.9)	75.0(2.5)	**75.8(0.6)**	75.2(0.3)
MUTAG	85.2(2.3)	84.4(1.5)	81.6(2.1)	85.4(1.2)	85.8(1.7)	89.0(4.4)	**90.3(1.1)**	90.1(1.0)
DD	>24h	78.6(0.4)	78.5(0.3)	–	79.4(0.9)	76.2(2.6)	**81.6(0.3)**	81.0(0.5)
NCI1	74.8(0.4)	**85.4(0.3)**	62.3(0.3)	–	74.4(0.5)	76.3(1.7)	84.5(0.2)	83.5(0.2)
PTC-FM	60.5(1.7)	55.2(2.3)	–	63.8(1.0)	–	–	62.3(1.0)	**63.9(1.3)**
PTC-FR	61.6(1.0)	63.9(1.4)	–	65.5(1.4)	–	–	66.7(1.4)	**67.8(1.1)**
PTC-MM	62.9(1.4)	60.6(1.1)	–	63.3(1.7)	–	–	65.6(1.1)	**67.9(1.4)**
PTC-MR	57.8(2.1)	55.4(1.5)	57.3(1.1)	58.1(1.6)	58.6(2.5)	62.3(5.7)	**62.5(1.6)**	62.1(1.5)
Total time	>24h	2m27s	–	–	–	–	38m4s	**49.9s**

3.6.4 Sensitivity Analysis

Here, we conduct a parameter sensitivity analysis of RetGK_{II} on the datasets REDDIT-BINARY, NCI1, SYNTHETIC, Synthie, ENZYMES, and PROTEINS. We test the stability

Table 3.3: Classification results (in %) for graph datasets with continuous attributes

Datasets	HGK(WL)	RetGK_{I}	RetGK_{II}
ENZYMES	63.9(1.1)	70.0(0.9)	**70.7(0.9)**
PROTEINS	74.9(0.6)	**76.2(0.5)**	75.9(0.4)
FRANK	73.2(0.3)	76.4(0.3)	**76.7(0.4)**
SYNTHETIC	97.6(0.4)	97.9(0.3)	**98.9(0.4)**
Synthie	80.3(1.4)	**97.1(0.3)**	96.2(0.3)
Total time	–	45m30s	**40.8s**

Table 3.4: Classification results (in %) for graph datasets with both discrete and continuous attributes

Datasets	GIK	CSM	RetGK_{I}	RetGK_{II}
ENZYMES	71.7(0.8)	69.8(0.7)	**72.2(0.8)**	70.6(0.7)
PROTEINS	76.1(0.3)	–	**78.0(0.3)**	77.3(0.5)
BZR	–	79.4(1.2)	86.4(1.2)	**87.1(0.7)**
COX2	–	74.4(1.7)	80.1(0.9)	**81.4(0.6)**
DHFR	–	79.9(1.1)	81.5(0.9)	**82.5(0.8)**
Total time	–	–	4m17s	**2m51s**

of RetGK_{II} by varying the values of the random walk steps S, the dimension D_0 of the approximate explicit feature map on $\mathcal{A}_0$, and the dimension D_c of the feature map on $\mathcal{A}_c$. We plot the average classification accuracy of ten repetitions of 10-fold cross-validations with respect to S, D_0, and D_c in Fig. 3.5. It can be concluded that RetGK_{II} performs consistently across a wide range of parameter values.

3.7 Chapter Summary

In this chapter, we introduced the return probability feature for characterizing and comparing the structural role of nodes across graphs. Based on the RPF, we embedded graphs in an RKHS and derived the corresponding graph kernels RetGK_{I}. Then, making use of approximate explicit feature maps, we represented each graph with a multi-dimensional tensor, and

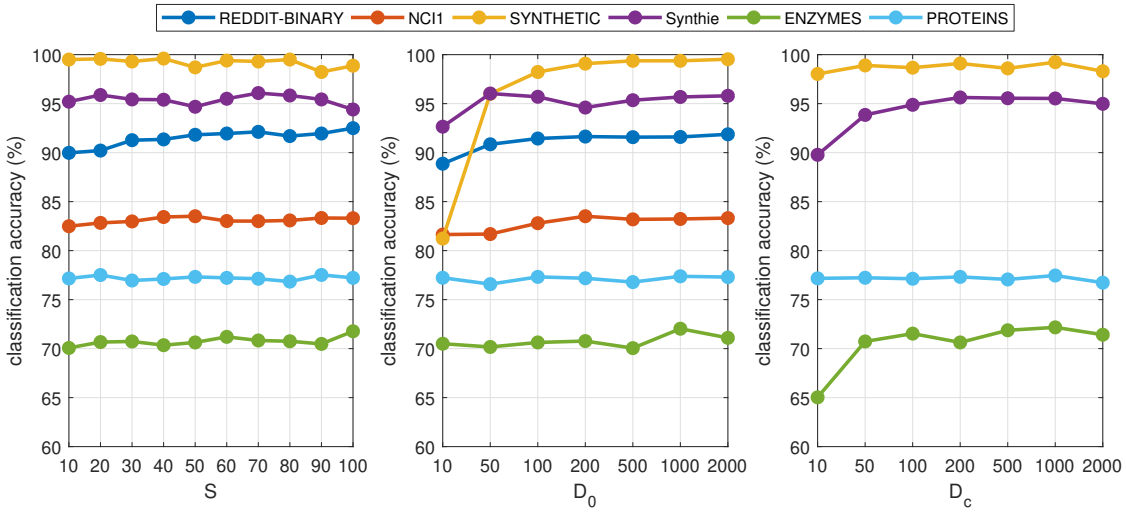


Figure 3.5: Parameter sensitivity study for RetGK_{II} on six benchmark datasets

then obtained the computationally efficient graph kernels RetGK_{II}. We applied RetGK_{I} and RetGK_{II} to classify graphs, and achieved promising results on many benchmark datasets. Given the prevalence of structured data, we believe that our work can be potentially useful in many applications.

Chapter 4

SAGE: Scalable Attributed Graph Embeddings

4.1 Introduction

In the last chapter, we developed RetGK, an effective graph kernel for quantitatively measuring the similarity between any two graphs. However, graph kernels can only feed the kernel-based learning algorithms, which requires to compute all the pairwise kernel values. The computational complexity scales quadratically to the number of samples, limiting its applications to the dataset of small size. There are many other types of machine learning methods, such as, logistic regression, classification/regression tree, and neural networks, which does not have the above issue. Therefore, we want to represent graphs with Euclidean vectors so that we can apply these methods to the graph-structured data. The other concern is that RetGK considers only node attributes while ignoring edge attributes which are usually very important for characterizing the interaction between objects (nodes).

In this chapter, we develop scalable attributed graph embeddings (SAGE) [17] for graphs with both node and edge attributes. We still use the return probability features (see (3.5)) for representing node. However, instead of using kernel mean embedding (see section 3.4), we employ the recently developed "D2KE" (which is short for "from **d**istance to **k**ernel and **e**mbeddings") framework [67] to represent graphs with Euclidean vectors. We introduce a novel strategy of converting edge-attributed graphs to node-attributed graphs, by the virtue of the adjoint graph. Therefore, edge attributes information can be involved in the exactly same way as node attributes information.

4.2 Graph Dissimilarity Measure

We recall a node-attributed graph G of n nodes can be represented by a set (see 3.10)

$$\mathcal{S}(G) = \{(\vec{\boldsymbol{p}}_i, \vec{\boldsymbol{a}}_i^{(d)}, \vec{\boldsymbol{a}}_i^{(c)})\}_{i=1}^{n} \subseteq \mathbb{R}^S \times \mathcal{A}_d \times \mathcal{A}_c, \tag{4.1}$$

where $\vec{\boldsymbol{p}}_i, \vec{\boldsymbol{a}}_i^{(d)}, \vec{\boldsymbol{a}}_i^{(c)}$ are return probability features, discrete node attributes, and continuous node attributes[7], respectively, and $\vec{\boldsymbol{a}}_i^{(d)}$ is an one-hot vector. The problem of measuring the difference between node-attributed graphs is converted to that of measuring the difference between two corresponding representation sets.

In this chapter, we use the energy distance [68]. Let $\mathcal{S}_1 = \{x_1, x_2, ..., x_{n_1}\} \subseteq \mathbb{R}^S \times \mathcal{A}_d \times \mathcal{A}_c$ and $\mathcal{S}_2 = \{y_1, y_2, ..., y_{n_2}\} \subseteq \mathbb{R}^S \times \mathcal{A}_d \times \mathcal{A}_c$ be two sample sets. Let d be a metric (here called the "ground distance") on $\mathbb{R}^S \times \mathcal{A}_d \times \mathcal{A}_c$. The energy distance, D_E, between $\mathcal{S}_1$ and $\mathcal{S}_2$ is

[7]Note that without of loss of generality, we assume that $\mathcal{G}$ has both discrete node attributes domain, $\mathcal{A}_c$, and continuous node attributes domain, $\mathcal{A}_d$.

defined as

$$D_E^2 = 2A - B - C, \tag{4.2}$$

where the above terms $A = \frac{1}{n_1 n_2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} d(x_i, y_j)$, $B = \frac{1}{n_1^2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_1} d(x_i, x_j)$, and $C = \frac{1}{n_2^2} \sum_{i=1}^{n_2} \sum_{j=1}^{n_2} d(y_i, y_j)$, respectively.

4.2.1 Constructing the Ground Distance d

Now we construct the ground distance d such that it can well encode the difference of both the structure and node attributes information.

A straightforward way to integrate them is simply concatenating $\vec{\boldsymbol{p}}_i$, $\vec{\boldsymbol{a}}_i^{(d)}$, and $\vec{\boldsymbol{a}}_i^{(c)}$ to make a long vector $\vec{\boldsymbol{x}}_i$, $i = 1, 2, ..., n$. The ground distances between node representations are just the Euclidean distances between these long vectors. However, this method does not consider the heterogeneity of these multi-modal vectors.

Instead, we provide a novel ground distance by leveraging the power of tensors. That is, we represent each node v_i as a rank-one tensor, i.e., $x_i = \vec{\boldsymbol{p}}_i \otimes \vec{\boldsymbol{a}}_i^{(d)} \otimes \vec{\boldsymbol{a}}_i^{(c)}$. Applying the fact that $\langle \vec{\boldsymbol{u}}_1 \otimes \vec{\boldsymbol{v}}_1 \otimes \vec{\boldsymbol{w}}_1, \vec{\boldsymbol{u}}_2 \otimes \vec{\boldsymbol{v}}_2 \otimes \vec{\boldsymbol{w}}_2 \rangle_{\mathcal{T}} = \langle \vec{\boldsymbol{u}}_1, \vec{\boldsymbol{u}}_2 \rangle \cdot \langle \vec{\boldsymbol{v}}_1, \vec{\boldsymbol{v}}_2 \rangle \cdot \langle \vec{\boldsymbol{w}}_1, \vec{\boldsymbol{w}}_2 \rangle$, we can obtain d by computing the distance between two rank-one tensors x and y i.e.,

$$d^2(x, y) = \|x - y\|_{\mathcal{T}}^2 = \langle x, x \rangle_{\mathcal{T}} + \langle y, y \rangle_{\mathcal{T}} - 2\langle x, y \rangle_{\mathcal{T}}. \tag{4.3}$$

4.3 Scalable Attributed Graph Embeddings

We aim to construct graph embeddings from the energy distance. Let $\mathcal{X}_G$ be the space of all graphs. We define the function k on $\mathcal{X}_G \times \mathcal{X}_G$,

$$k(G_x, G_y) = \int_{\mathcal{X}_G} p(G_\omega)\zeta_{G_\omega}(G_x)\zeta_{G_\omega}(G_y)\mathrm{d}G_\omega, \tag{4.4}$$

where $\zeta_{G_\omega}(G) = \exp(-\gamma D_E(G_x, G_y))$ and G_ω is a random graph, and $p(G_\omega)$ is a probability distribution over all random graphs in $\mathcal{X}_G$. As shown in [67], we have

$$k(G_x, G_y) \to \exp(-\gamma D_E(G_x, G_y)) \quad \text{as } \gamma \to \infty, \tag{4.5}$$

which implies that k is a similarity measurement. Moreover, it can be checked that k is positive definite. Consequently, k is a graph kernel.

4.3.1 Node-attributed Graph Embeddings

The exact computation of k is intractable, since the integral over $\mathcal{X}_G$ has no analytic solutions. Inspired by randomized kernel approximation introduced in [34], we approximate the graph kernel k by using a finite number of random graphs. Let G_{ω_i}, $i = 1, 2, ..., R$ be random graphs sampled from a probability distribution p on $\mathcal{X}_G$. Then, as $R \to \infty$, we have

$$\tilde{k}(G_x, G_y) = \frac{1}{R}\sum_{i=1}^{R} \zeta_{G_{\omega_i}}(G_x)\zeta_{G_{\omega_i}}(G_y) \to k(G_x, G_y),$$

which explicitly yields the Euclidean space embedding, $\vec{\boldsymbol{Z}}^N(G)$, for node-attributed graph G, i.e.,

$$\vec{\boldsymbol{Z}}^N(G) = \frac{1}{\sqrt{R}}\big[\zeta_{G_{\omega_1}}(G), \zeta_{G_{\omega_2}}(G), ..., \zeta_{G_{\omega_R}}(G)\big]^T \in \mathbb{R}^R, \tag{4.6}$$

and obviously, we have $\langle \vec{\boldsymbol{Z}}^N(G_x), \vec{\boldsymbol{Z}}^N(G_y)\rangle = \tilde{k}(G_x, G_y)$.

Sampling random attributed graphs Here we provide a simple but effective approach for sampling random attributed graphs, which is similar with the data-dependent random features sampling strategy proposed in [69]. The "data-dependent" strategy means that we sample graphs from the training dataset $\{G_i\}_{i=1}^{N_{\text{tr}}}$. However, unlike the existing methods that select a representative set of a whole graph, we propose to sample parts of graphs as random graphs. This sampling method will generate random graphs with various topological connectivities and can help to identify hidden global structures. We also note that in order to compute $\zeta_{G_\omega}(G)$, it is sufficient to directly use the node embeddings set sampled from the node embedding space. Algorithm 2 gives our random graph sampling procedure.

Algorithm 2 Random Attributed Graphs Generation

Input: The node embedding sets $\{\mathcal{S}(G_i)\}_{i=1}^{N_{\text{train}}}$ of the training graph dataset, maximal size of the random graph $D_{\max}$.

Output: The node embeddings $\mathcal{S}(G_\omega)$ of a random graph G_ω.

1: Uniformly draw a number k in $\{1, 2, ..., N_{\text{train}}\}$.

2: Uniformly draw a number D_k in $\{1, 2, ..., D_{\max}\}$.

3: Randomly draw D_k nodes, i.e., $\{v_{i_1}, v_{i_2}, ..., v_{i_{D_k}}\}$ in the graph G_k.

4: Return node embeddings $\mathcal{S}(G_\omega) = \{(\vec{\boldsymbol{p}}_{i_m}, \vec{\boldsymbol{a}}_{i_m}^{(d)}, \vec{\boldsymbol{a}}_{i_m}^{(c)})\}_{m=1}^{D_k}$, where $\{(\vec{\boldsymbol{p}}_i, \vec{\boldsymbol{a}}_i^{(d)}, \vec{\boldsymbol{a}}_i^{(c)})\}_{i=1}^{n_k} = \mathcal{S}(G_k)$ is the node embedding set of the graph G_k of n_k nodes.

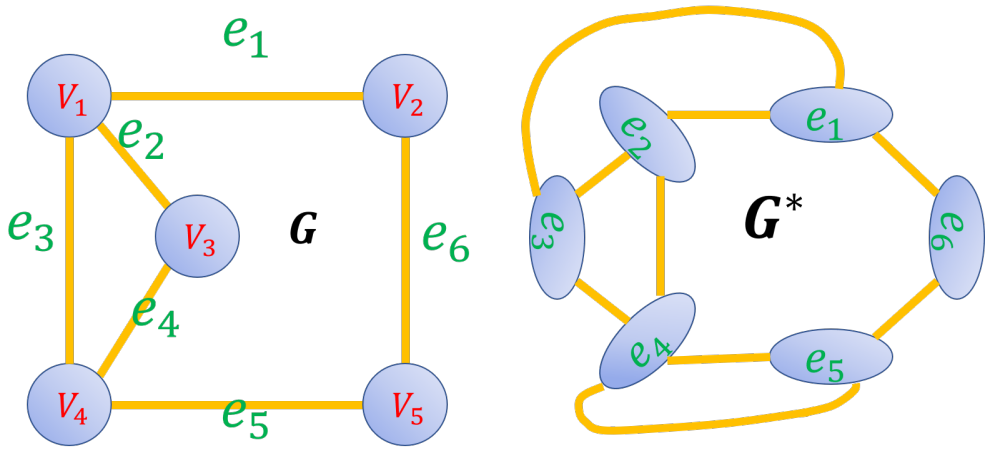


Figure 4.1: Left: A graph G. Right: The adjoint graph G^* converted from G. For example, in G, the edges e_1, e_2, and e_3 share a common node V_1. So in G^*, the nodes e_1, e_2, and e_3 are connected.

4.3.2 Edge-attributed Graph Embeddings

So far we consider only the structure and node attributes information, while losing the edge attributes. Because our method is built on the "node embedding of graphs", the edge attributes cannot be coupled with the node representations. To overcome this drawback, we employ the adjoint graph [70] (also called the "line graph" or the "edge graph"), which translates the property of the original graph from edges to nodes. Consequently, we can consider the problem of embedding edge-attributed graphs as that of embedding node-attributed graphs.

Converting edge attributes to node attributes Given a graph G, we can construct its adjoint graph G^* such that i) the nodes in G^* represent the edges in G; ii) two nodes in G^* are adjacent if the corresponding edges in G have the same ending nodes. A toy graph is shown in Fig. 4.1. We can see that the edges in G are transformed to the nodes in

G^*. Simultaneously, the edge attributes are converted into the node attributes. Therefore, the node embeddings of G^* are equivalent to the edge embeddings of G. The following theorem demonstrate that in almost every case, the adjoint graph preserves all the structure information of the original graph.

Theorem 4.1. *[70] Two graphs are isomorphic if and only if their adjoint graphs are isomorphic, with the exception of the triangle graph K_3 and the claw $K_{1,3}$.*

In addition, it is very convenient to obtain the adjacent matrix of G^*, which can be computed by

$$\boldsymbol{A}_{G^*} = \boldsymbol{C}_G^T \boldsymbol{C}_G - 2\boldsymbol{I}, \tag{4.7}$$

where $\boldsymbol{C}_G$ is the incidence matrix of G (see (2.2)).

Edge-attributed graph embeddings Now we can operate the return probability feature extraction on G^*. Let $\boldsymbol{q}_j$ and $\vec{\boldsymbol{b}}_j$ respectively be the edge structural representation and edge attributes of edge e_i in G, we can then characterize the graph with the set (similar with (4.1))

$$\mathcal{S}(G^*) = \{(\vec{\boldsymbol{q}}_j, \vec{\boldsymbol{b}}_j)\}_{j=1}^m \subseteq \mathbb{R}^S \times \mathcal{A}_E. \tag{4.8}$$

With $\mathcal{S}(G^*)$ (4.8), we can get the edge-attributed graph representation,

$$\vec{\boldsymbol{Z}}^E(G) = \vec{\boldsymbol{Z}}^N(G^*) \in \mathbb{R}^R, \tag{4.9}$$

based on the method in Section 4.3.1.

4.3.3 Two Types of Graph Embedding Fusion

In some cases, the graph has both node and edge attributes. Following the graph embedding procedure in Section 3.1 and Section 3.2, we will have both the node-attributed embedding $\vec{\boldsymbol{Z}}^N(G)$ and the edge-attributed embedding $\vec{\boldsymbol{Z}}^E(G)$. There are many fusion strategies. In our paper, we fuse $\vec{\boldsymbol{Z}}^N(G)$ and $\vec{\boldsymbol{Z}}^E(G)$ in the following way.

To obtain the final embedding vector, $\vec{\boldsymbol{Z}}(G)$, of G, we first compute the Hadamard product and the absolute difference of $\vec{\boldsymbol{Z}}^N(G)$ and $\vec{\boldsymbol{Z}}^E(G)$, and then concatenate them with $\vec{\boldsymbol{Z}}^N(G)$ together, i.e.,

$$\vec{\boldsymbol{Z}}(G) = \left[\vec{\boldsymbol{Z}}_0^T(G), \vec{\boldsymbol{Z}}_1^T(G), \vec{\boldsymbol{Z}}_2^T(G)\right]^T \in \mathbb{R}^{3R}, \tag{4.10}$$

where $\vec{\boldsymbol{Z}}_0(G) = \vec{\boldsymbol{Z}}^N(G)$, $\vec{\boldsymbol{Z}}_1(G) = \vec{\boldsymbol{Z}}^N(G) \circ \vec{\boldsymbol{Z}}^E(G)$, and $\vec{\boldsymbol{Z}}_2(G) = |\vec{\boldsymbol{Z}}^N(G) - \vec{\boldsymbol{Z}}^E(G)|$. Note the above fusion strategy has widely been used in natural language processing, and has achieved promising peformance [71].

4.3.4 Summary of the SAGE Algorithm

In algorithm 3, we summarize the procedure of obtaining the vector embedding of graph $G = \{\mathcal{V}, \mathcal{E}, \mathcal{A}_N, \mathcal{A}_E\}$, which has both node attributes and edge attributes. These attributes can be discrete or continuous.

Algorithm 3 Scalable Attributed Graph Embeddings

Input: The attributed graph sets $\{G_i\}_{i=1}^N$, the dimension of RPF S, the number of random graphs R.

Output: The graph embeddings $\{\vec{\boldsymbol{Z}}(G_i)\}_{i=1}^N$.

1: Compute the node embeddings $\{\mathcal{S}(G_i)\}_{i=1}^N$ (see (4.1)) and the edge embeddings $\{\mathcal{S}(G_i^*)\}_{i=1}^N$ (see (4.8)).
2: Generate random graph representations $\{\mathcal{S}(G_{\omega_i})\}_{i=1}^R$ and $\{\mathcal{S}(G_{\omega_i}^*)\}_{i=1}^R$ using Algorithm 2.
3: Compute $\vec{\boldsymbol{Z}}^N(G_i)$ (see (4.6)) and $\vec{\boldsymbol{Z}}^E(G_i)$ (see (4.9)), $i = 1, 2, ..., N$.
4: Compute $\vec{\boldsymbol{Z}}(G_i)$ (see (4.10)), $i = 1, 2, ..., N$.

4.4 Experiments

We carry out experiments to evaluate the effectiveness and efficiency of SAGE, with the goal of answering the following two questions:

- **Q1**: How does SAGE scale with respect to the number of graphs, N?
- **Q2**: How much improvement does SAGE achieve by considering the node and edge attributes?

Datasets We consider graphs with both node and edge attributes information [54]. BZR_MD, COX2_MD, DHFR_MD, ER_MD, and AIDS.

Setup In our experiments, we directly employ a linear SVM implemented in LIBLINEAR [72] since SAGE is a graph-level embedding. The cost parameter C is selected from $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10^1, 10^2, 10^3\}$. We set the node embedding size $S = 50$ and select R in the range of $[4, \min(2^k \leq N, 2048)]$, where N the size of the graph dataset. The ranges of hyperparameters γ and $D_{\max}$ are $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10^1\}$ and [3:3:30], respectively. All parameters of the SVM and hyperparameters of our method were optimized only on the training dataset. For all these datasets, we perform 10-fold cross-validation to evaluate the

performance of SAGE, using 9 folds for training and 1 for testing. To eliminate the random effects, we repeat the experiments ten times (thus 100 runs per dataset) and report averaged prediction accuracies and standard deviations.

4.4.1 Empirical Impact of the Number of Graphs on Running Time

We investigate the scalability of SAGE on synthetic graphs when varying the number of graphs in the range of $N = [8,\ 32768]$ and the size of graph in the range of $n = [16,\ 1024]$, respectively. Figure 4.2 shows the computational time for computing node embeddings, generating SAGE graph embeddings, and the total computation of graph classification, accordingly. As shown in Fig. 4.2, these results yields encouraging answers to our motivating question **Q1**: SAGE shows the linear scalability in terms of the number of graphs. This is a highly desired property of our SAGE embeddings, since most graph kernels have quadratic complexity in the number of graphs, rendering them hard to scale well.

4.4.2 Ablation Study of SAGE

To pursue the answer for question **Q2**, we perform ablation study of SAGE on graph datasets that have both node attributes and edge attributes. Table 4.1 shows the classification accuracies of three variants of SAGE: i) SAGE-plain without considering any attribute information, ii) SAGE-node considering only node attributes, and iii) SAGE-node-edge considering both node and edge information. The classification results of RetGK are presented as the baselines. We can observe that SAGE-node-edge consistently performs better than both SAGE-node

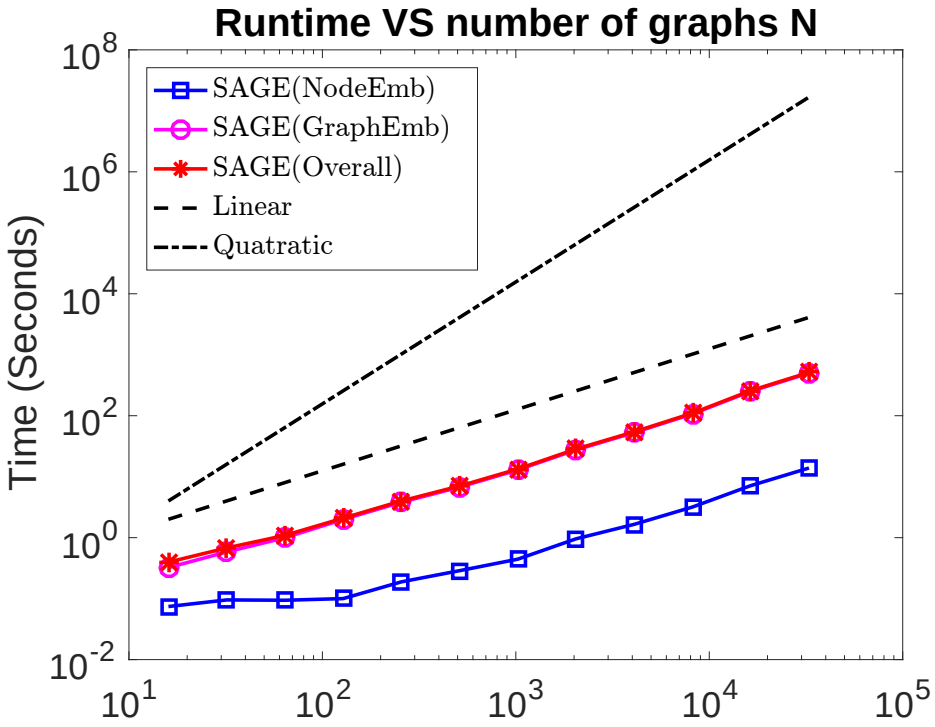


Figure 4.2: Varying number of graphs(N)

and SAGE-plain with significant improvement. Interestingly, even if we consider only the node attributes, the overall performance of SAGE is still better than that of RetGK.

Table 4.1: Ablation study of SAGE for classification accuracy (in %) on graphs with node and edge information

Datasets	RetGK_{I}	SAGE-plain	SAGE-node	SAGE-node-edge
BZR_MD	63.4(1.5)	62.0(1.2)	69.3(0.7)	70.7(0.7)
COX2_MD	63.3(2.2)	50.3(0.9)	63.3(1.0)	67.0(0.9)
DHFR_MD	66.2(1.6)	67.2(0.6)	69.2(0.7)	70.5(0.8)
ER_MD	72.8(1.4)	66.6(0.8)	73.6(0.7)	75.7(0.9)
Cuneiform	38.4(1.9)	7.7(0.4)	37.3(0.9)	45.8(1.1)
AIDS	99.4(0.1)	99.4(0.3)	99.5(0.3)	99.5(0.3)

4.5 Chapter Summary

In this work, we developed a scalable algorithm for representing attributed graphs, which may have node and edge attributes. We still follow the two-step embedding framework, i.e., the graph node-level embeddings and the graph-level embeddings. Leveraging the adjoint graphs, we translated the properties of graphs from edges to nodes, which provides a unifying view for embedding node-attributed and edge-attributed graphs. In the graph classification task, our graph embeddings achieve superior performance in both accuracy and scalability.

Chapter 5

KerGM: Kernelized Graph Matching

5.1 Introduction

In this chapter, we develop our kernelized graph matching agorithm. We start with the introduction to two quadratic assignment problems (QAPs) for graph matching. After that, we introduce the related work.

5.1.1 Quadratic Assignment Problems for Graph Matching

Let $\mathcal{G} = \{\boldsymbol{A}, \mathcal{V}, \boldsymbol{P}, \mathcal{E}, \boldsymbol{Q}\}$ be an undirected, attributed graph of n nodes and m edges, where $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is the adjacency matrix, $\mathcal{V} = \{v_i\}_{i=1}^n$ and $\boldsymbol{P} = [\vec{\boldsymbol{p}}_1, \vec{\boldsymbol{p}}_2, ..., \vec{\boldsymbol{p}}_n] \in \mathbb{R}^{d_N\times n}$ are the respective node set and node attributes matrix, and $\mathcal{E} = \{e_{ij}|v_i \text{ and } v_j \text{ are connected}\}$ and $\boldsymbol{Q} = [\vec{\boldsymbol{q}}_{ij}|e_{ij} \in \mathcal{E}] \in \mathbb{R}^{d_E\times m}$ are the respective edge set and edge attributes matrix. Given two graphs $\mathcal{G}_1 = \{\boldsymbol{A}_1, \mathcal{V}_1, \boldsymbol{P}_1, \mathcal{E}_1, \boldsymbol{Q}_1\}$ and $\mathcal{G}_2 = \{\boldsymbol{A}_2, \mathcal{V}_2, \boldsymbol{P}_2, \mathcal{E}_2, \boldsymbol{Q}_2\}$ of n nodes[8], the graph matching (GM) problem aims to find a correspondence between nodes in $\mathcal{V}_1$ and $\mathcal{V}_2$ which is optimal in some sense.

[8]We assume G_1 and G_2 have the same number of nodes. If not, we add dummy nodes.

For **Koopmans-Beckmann's QAP** [26], the optimality refers to the Frobenius inner product maximization between two adjacency matrices after permutation, i.e.,

$$\max \langle \boldsymbol{A}_1 \boldsymbol{X}, \boldsymbol{X} \boldsymbol{A}_2 \rangle_{\mathrm{F}} \quad \text{s.t. } \boldsymbol{X} \in \mathcal{P} = \{ \boldsymbol{X} \in \{0,1\}^{n \times n} | \boldsymbol{X} \vec{\mathbf{1}} = \vec{\mathbf{1}}, \boldsymbol{X}^T \vec{\mathbf{1}} = \vec{\mathbf{1}} \}, \tag{5.1}$$

where $\langle \boldsymbol{A}, \boldsymbol{B} \rangle_{\mathrm{F}} = \mathrm{tr}(\boldsymbol{A}^T \boldsymbol{B})$ is the Frobenius inner product. The issue with (5.1) is that it ignores the complex edge attributes, which are usually of particular importance in characterizing graphs.

For **Lawler's QAP** [27], the optimality refers to the similarity maximization between the node attribute sets and between the edge attribute sets, i.e.,

$$\max \sum_{v_i^1 \in \mathcal{V}_1, v_a^2 \in \mathcal{V}_2} k^N(\vec{\boldsymbol{p}}_i^1, \vec{\boldsymbol{p}}_a^2) \boldsymbol{X}_{ia} + \sum_{e_{ij}^1 \in \mathcal{E}_1, e_{ab}^2 \in \mathcal{E}_2} k^E(\vec{\boldsymbol{q}}_{ij}^1, \vec{\boldsymbol{q}}_{ab}^2) \boldsymbol{X}_{ia} \boldsymbol{X}_{jb} \quad \text{s.t. } \boldsymbol{X} \in \mathcal{P}, \tag{5.2}$$

where k^N and k^E are the node and edge similarity measurements, respectively. Furthermore, (5.2) can be rewritten in compact form:

$$\max \langle \boldsymbol{K}^N, \boldsymbol{X} \rangle_{\mathrm{F}} + \mathrm{vec}(\boldsymbol{X})^T \boldsymbol{K} \mathrm{vec}(\boldsymbol{X}) \quad \text{s.t. } \boldsymbol{X} \in \mathcal{P}, \tag{5.3}$$

where $\boldsymbol{K}^N \in \mathbb{R}^{n \times n}$ is the node affinity matrix, $\boldsymbol{K}$ is an $n^2 \times n^2$ matrix, defined such that

$$\boldsymbol{K}_{ia,jb} = \begin{cases} k^E(\vec{\boldsymbol{q}}_{ij}^1, \vec{\boldsymbol{q}}_{ab}^2), & \text{if } i \neq j, a \neq b, e_{ij}^1 \in \mathcal{E}_1, \text{and } e_{ab}^2 \in \mathcal{E}_2, \\ 0, & \text{otherwise} \end{cases}. \tag{5.4}$$

It is well known that Koopmans-Beckmann's QAP is a special case of Lawler's QAP if we set $\boldsymbol{K} = \boldsymbol{A}_2 \otimes \boldsymbol{A}_1$ and $\boldsymbol{K}^N = \boldsymbol{0}_{n \times n}$. The issue of (5.3) is that the size of $\boldsymbol{K}$ scales quadruply with respect to n, which precludes its applications to large graphs. In our work, we will show

that Lawler's QAP can be written in the Koopmans-Beckmann's form, which can avoid computing $\boldsymbol{K}$.

5.1.2 Related Works

In the past forty years, a myriad of graph matching algorithms have been proposed [24], most of which focused on solving QAPs. Previous work [73, 74, 75] approximated the quadratic term with a linear one, which consequently can be solved by standard linear programming solvers. In [76], several convex relaxation methods are proposed and compared. It is known that convex relaxations can achieve global convergence, but usually perform poorly because the final projection step separates the solution from the original QAP. Concave relaxations [77, 78] can avoid this problem since their outputs are just permutation matrices. However, concave programming [79] is NP-hard, which limits its applications. In [80], a seminal work termed the "path-following algorithm" was proposed, which leverages both the above relaxations via iteratively solving a series of optimization problems that gradually changed from convex to concave. In [81, 82, 83, 84], the path following strategy was further extended and improved. However, all the above algorithms, when applied to Lawler's QAP, need to compute the $n^2 \times n^2$ affinity matrix. To tackle the challenge, in [3], the authors elegantly factorized the affinity matrix into the Kronecker product of smaller matrices. However, it still cannot be well applied to large dense graphs, since it scales cubically with the number of edges. Beyond solving the QAP, there are interesting works on doing graph matching from other perspectives, such as probabilistic matching[85], hypergraph matching [86], and multigraph matching [87]. We refer to [88] for a survey of recent advances.

Organization: In Section 2, we present the proposed rules for array operations in Hilbert space. Section 3, Section 4 and Section 5 form the core of our work, where we develop the kernelized graph matching and the corresponding approximation. In Section 6, we develop the entropy-regularized Frank-Wolfe optimizaton algorithm and prove its convergence rate. In Section 7, we report the experimental results.

5.2 $\mathcal{H}$-operations for Arrays in Hilbert Spaces

Let $\mathcal{H}$ be any Hilbert space, coupled with the inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ taking values in $\mathbb{R}$. Let $\mathcal{H}^{n\times n}$ be the set of all $n \times n$ arrays in $\mathcal{H}$, and let $\boldsymbol{\Psi}$, $\boldsymbol{\Xi} \in \mathcal{H}^{n\times n}$, i.e., $\boldsymbol{\Psi}_{ij}$, $\boldsymbol{\Xi}_{ij} \in \mathcal{H}$, $\forall i, j = 1, 2, ..., n$. Analogous to matrix operations in Euclidean spaces, we make the following addition, transpose, and multiplication rules ($\mathcal{H}$-operations), i.e.,$\forall \boldsymbol{X} \in \mathbb{R}^{n\times n}$, and we have

1. $\boldsymbol{\Psi} + \boldsymbol{\Xi}$, $\boldsymbol{\Psi}^T \in \mathcal{H}^{n\times n}$, where $[\boldsymbol{\Psi} + \boldsymbol{\Xi}]_{ij} \triangleq \boldsymbol{\Psi}_{ij} + \boldsymbol{\Xi}_{ij} \in \mathcal{H}$ and $[\boldsymbol{\Psi}^T]_{ij} \triangleq \boldsymbol{\Psi}_{ji} \in \mathcal{H}$.
2. $\boldsymbol{\Psi} * \boldsymbol{\Xi} \in \mathbb{R}^{n\times n}$, where $[\boldsymbol{\Psi} * \boldsymbol{\Xi}]_{ij} \triangleq \sum_{k=1}^{n} \langle \boldsymbol{\Psi}_{ik}, \boldsymbol{\Xi}_{kj} \rangle_{\mathcal{H}} \in \mathbb{R}$.
3. $\boldsymbol{\Psi} \odot \boldsymbol{X}$, $\boldsymbol{X} \odot \boldsymbol{\Psi} \in \mathcal{H}^{n\times n}$, where $[\boldsymbol{\Psi} \odot \boldsymbol{X}]_{ij} \triangleq \sum_{k=1}^{n} \boldsymbol{\Psi}_{ik} \boldsymbol{X}_{kj} = \sum_{k=1}^{n} \boldsymbol{X}_{kj} \boldsymbol{\Psi}_{ik} \in \mathcal{H}$ and $[\boldsymbol{X} \odot \boldsymbol{\Psi}]_{ij} \triangleq \sum_{k=1}^{n} \boldsymbol{X}_{ik} \boldsymbol{\Psi}_{kj} \in \mathcal{H}$.

Note that if $\mathcal{H} = \mathbb{R}$, all the above degenerate to the common operations for matrices in Euclidean spaces. In Fig. 5.1, we visualize the operation $\boldsymbol{\Psi} \odot \boldsymbol{X}$, where we let $\mathcal{H} = \mathbb{R}^d$, let $\boldsymbol{\Psi}$ be a 3×3 array in $\mathbb{R}^d$, and let $\boldsymbol{X}$ be a 3×3 permutation matrix. It is easy to see that $\boldsymbol{\Psi} \odot \boldsymbol{X}$ is just $\boldsymbol{\Psi}$ after column-permutation.

As presented in the following corollary, the multiplication $\odot$ satisfy the combination law.

Corollary 5.1. $\forall \boldsymbol{X}, \boldsymbol{Y} \in \mathbb{R}^{n\times n}$, $\boldsymbol{\Psi} \odot \boldsymbol{X} \odot \boldsymbol{Y} = \boldsymbol{\Psi} \odot (\boldsymbol{X}\boldsymbol{Y})$, and $\boldsymbol{Y} \odot (\boldsymbol{X} \odot \boldsymbol{\Psi}) = (\boldsymbol{Y}\boldsymbol{X}) \odot \boldsymbol{\Psi}$.

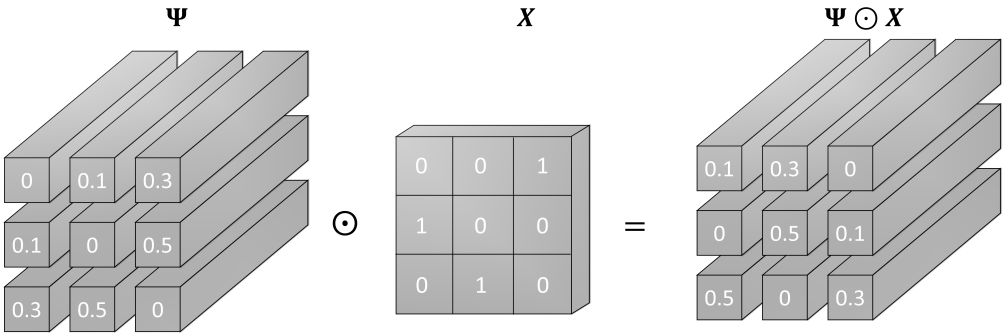


Figure 5.1: Visualization of the operation $\boldsymbol{\Psi} \odot \boldsymbol{X}$.

Proof. **(1).** $\forall i, j = 1, 2, ..., n$, the (i, j) element of $\boldsymbol{\Psi} \odot \boldsymbol{X} \odot \boldsymbol{Y}$ is

$$\begin{aligned}[\boldsymbol{\Psi} \odot \boldsymbol{X} \odot \boldsymbol{Y}]_{ij} &= \sum_{k=1}^{n} [\boldsymbol{\Psi}\boldsymbol{X}]_{ik} \boldsymbol{Y}_{kj} = \sum_{k=1}^{n} \boldsymbol{Y}_{kj} \sum_{\alpha=1}^{n} \boldsymbol{\Psi}_{i\alpha} \boldsymbol{X}_{\alpha k} \\ &= \sum_{\alpha=1}^{n} \boldsymbol{\Psi}_{i\alpha} \big(\sum_{k=1}^{n} \boldsymbol{X}_{\alpha k} \boldsymbol{Y}_{kj} \big) = \sum_{\alpha=1}^{n} \boldsymbol{\Psi}_{i\alpha} [\boldsymbol{X}\boldsymbol{Y}]_{\alpha j} = [\boldsymbol{\Psi} \odot (\boldsymbol{X}\boldsymbol{Y})]_{ij}. \end{aligned} \tag{5.5}$$

Therefore, $\boldsymbol{\Psi} \odot \boldsymbol{X} \odot \boldsymbol{Y} = \boldsymbol{\Psi} \odot (\boldsymbol{X}\boldsymbol{Y})$.

(2). $\forall i, j = 1, 2, ..., n$, the (i, j) element of $\boldsymbol{Y} \odot (\boldsymbol{X} \odot \boldsymbol{\Psi})$ is

$$[\boldsymbol{Y} \odot (\boldsymbol{X} \odot \boldsymbol{\Psi})]_{ij} = \sum_{k=1}^{n} \boldsymbol{Y}_{ik} [\boldsymbol{X}\boldsymbol{\Psi}]_{kj} = \sum_{k=1}^{n} \boldsymbol{Y}_{ik} \sum_{\alpha=1}^{n} \boldsymbol{X}_{k\alpha} \boldsymbol{\Psi}_{\alpha j} \tag{5.6}$$

$$= \sum_{\alpha=1}^{n} \big(\sum_{k=1}^{n} \boldsymbol{Y}_{ik} \boldsymbol{X}_{k\alpha} \big) \boldsymbol{\Psi}_{\alpha j} = \sum_{\alpha=1}^{n} [\boldsymbol{Y}\boldsymbol{X}]_{i\alpha} \boldsymbol{\Psi}_{\alpha j} = [(\boldsymbol{Y}\boldsymbol{X}) \odot \boldsymbol{\Psi}]_{ij}. \tag{5.7}$$

Therefore, $\boldsymbol{Y} \odot (\boldsymbol{X} \odot \boldsymbol{\Psi}) = (\boldsymbol{Y}\boldsymbol{X}) \odot \boldsymbol{\Psi}$. □

Based on the $\mathcal{H}$-operations, we can construct the Frobenius inner product on $\mathcal{H}^{n \times n}$.

Proposition 5.1. *Define the function* $\langle \cdot, \cdot \rangle_{\mathrm{F}_{\mathcal{H}}} : \mathcal{H}^{n \times n} \times \mathcal{H}^{n \times n} \to \mathbb{R}$ *such that* $\langle \boldsymbol{\Psi}, \boldsymbol{\Xi} \rangle_{\mathrm{F}_{\mathcal{H}}} \triangleq \mathrm{tr}(\boldsymbol{\Psi}^T * \boldsymbol{\Xi}) = \sum_{i,j=1}^{n} \langle \boldsymbol{\Psi}_{ij}, \boldsymbol{\Xi}_{ij} \rangle_{\mathcal{H}}$, $\forall \boldsymbol{\Psi}, \boldsymbol{\Xi} \in \mathcal{H}^{n \times n}$. *Then* $\langle \cdot, \cdot \rangle_{\mathrm{F}_{\mathcal{H}}}$ *is an inner product on* $\mathcal{H}^{n \times n}$.

Proof. It is sufficient to show that the function $\langle \cdot, \cdot \rangle_{\mathrm{F}_{\mathcal{H}}}$ satisfies the following properties.

1. [Conjugate symmetry]:

$$\overline{\langle \boldsymbol{\Psi}, \boldsymbol{\Xi} \rangle_{\mathrm{F}_{\mathcal{H}}}} = \sum_{i,j=1}^{n} \overline{\langle \boldsymbol{\Psi}_{ij}, \boldsymbol{\Xi}_{ij} \rangle_{\mathcal{H}}} = \sum_{i,j=1}^{n} \langle \boldsymbol{\Xi}_{ij}, \boldsymbol{\Psi}_{ij} \rangle_{\mathcal{H}} = \langle \boldsymbol{\Xi}, \boldsymbol{\Psi} \rangle_{\mathrm{F}_{\mathcal{H}}}, \tag{5.8}$$

2. [Linearity in the first argument]:

$$\langle a\boldsymbol{\Psi}, \boldsymbol{\Xi} \rangle_{\mathrm{F}_{\mathcal{H}}} = \sum_{i,j=1}^{n} \langle a\boldsymbol{\Psi}_{ij}, \boldsymbol{\Xi}_{ij} \rangle_{\mathcal{H}} = \sum_{i,j=1}^{n} a\langle \boldsymbol{\Psi}_{ij}, \boldsymbol{\Xi}_{ij} \rangle_{\mathcal{H}} = a\langle \boldsymbol{\Psi}, \boldsymbol{\Xi} \rangle_{\mathrm{F}_{\mathcal{H}}} \tag{5.9}$$

$$\langle \boldsymbol{\Psi}^{(1)} + \boldsymbol{\Psi}^{(2)}, \boldsymbol{\Xi} \rangle_{\mathrm{F}_{\mathcal{H}}} = \sum_{i,j=1}^{n} \langle \boldsymbol{\Psi}_{ij}^{(1)} + \boldsymbol{\Psi}_{ij}^{(2)}, \boldsymbol{\Xi}_{ij} \rangle_{\mathcal{H}} = \sum_{i,j=1}^{n} \langle \boldsymbol{\Psi}_{ij}^{(1)}, \boldsymbol{\Xi}_{ij} \rangle_{\mathcal{H}} + \sum_{i,j=1}^{n} \langle \boldsymbol{\Psi}_{ij}^{(2)}, \boldsymbol{\Xi}_{ij} \rangle_{\mathcal{H}}. \tag{5.10}$$

$$= \langle \boldsymbol{\Psi}^{(1)}, \boldsymbol{\Xi} \rangle_{\mathrm{F}_{\mathcal{H}}} + \langle \boldsymbol{\Psi}^{(2)}, \boldsymbol{\Xi} \rangle_{\mathrm{F}_{\mathcal{H}}} \tag{5.11}$$

3. [Positive-definiteness]:

$$\langle \boldsymbol{\Psi}, \boldsymbol{\Psi} \rangle_{\mathrm{F}_{\mathcal{H}}} = \sum_{i,j=1}^{n} \langle \boldsymbol{\Psi}_{ij}, \boldsymbol{\Psi}_{ij} \rangle_{\mathcal{H}} \geq 0. \tag{5.12}$$

$$\langle \boldsymbol{\Psi}, \boldsymbol{\Psi} \rangle_{\mathrm{F}_{\mathcal{H}}} = 0 \iff \forall i, j = 1, 2, ..., n, \boldsymbol{\Psi}_{ij} = \mathbf{0} \iff \boldsymbol{\Psi} = \boldsymbol{O}. \tag{5.13}$$

□

As an immediate result, the function $\| \cdot \|_{\mathrm{F}_{\mathcal{H}}} : \mathcal{H}^{n \times n} \to \mathbb{R}$, defined such that $\|\boldsymbol{\Psi}\|_{\mathrm{F}_{\mathcal{H}}} = \sqrt{\langle \boldsymbol{\Psi}, \boldsymbol{\Psi} \rangle_{\mathrm{F}_{\mathcal{H}}}}$, is the Frobenius norm on $\mathcal{H}^{n \times n}$. Next, we introduce two properties of $\langle \cdot, \cdot \rangle_{\mathrm{F}_{\mathcal{H}}}$, which play important roles for developing the convex-concave relaxation of the Lawler's graph matching problem.

Corollary 5.2. $\langle \boldsymbol{\Psi} \odot \boldsymbol{X}, \boldsymbol{\Xi} \rangle_{\mathrm{F}_{\mathcal{H}}} = \langle \boldsymbol{\Psi}, \boldsymbol{\Xi} \odot \boldsymbol{X}^T \rangle_{\mathrm{F}_{\mathcal{H}}}$ *and* $\langle \boldsymbol{X} \odot \boldsymbol{\Psi}, \boldsymbol{\Xi} \rangle_{\mathrm{F}_{\mathcal{H}}} = \langle \boldsymbol{\Psi}, \boldsymbol{X}^T \odot \boldsymbol{\Xi} \rangle_{\mathrm{F}_{\mathcal{H}}}$.

Proof. **(1).**

$$\begin{aligned}\langle \boldsymbol{\Psi} \odot \boldsymbol{X}, \boldsymbol{\Xi} \rangle_{\mathrm{F}_{\mathcal{H}}} &= \sum_{i=1}^{n}\sum_{j=1}^{n}\langle [\boldsymbol{\Psi} \odot \boldsymbol{X}]_{ij}, \boldsymbol{\Xi}_{ij} \rangle_{\mathcal{H}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\langle \sum_{k=1}^{n} \boldsymbol{\Psi}_{ik}\boldsymbol{X}_{kj}, \boldsymbol{\Xi}_{ij} \rangle_{\mathcal{H}} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\langle \boldsymbol{\Psi}_{ik}, \boldsymbol{X}_{kj}\boldsymbol{\Xi}_{ij} \rangle_{\mathcal{H}} = \sum_{i=1}^{n}\sum_{k=1}^{n}\langle \boldsymbol{\Psi}_{ik}, \sum_{j=1}^{n} \boldsymbol{X}_{kj}\boldsymbol{\Xi}_{ij} \rangle_{\mathcal{H}} \\ &= \sum_{i=1}^{n}\sum_{k=1}^{n}\langle \boldsymbol{\Psi}_{ik}, [\boldsymbol{\Xi} \odot \boldsymbol{X}^T]_{ik} \rangle_{\mathcal{H}} = \langle \boldsymbol{\Psi}, \boldsymbol{\Xi} \odot \boldsymbol{X}^T \rangle_{\mathrm{F}_{\mathcal{H}}}. \end{aligned} \tag{5.14}$$

(2).

$$\begin{aligned}\langle \boldsymbol{X} \odot \boldsymbol{\Psi}, \boldsymbol{\Xi} \rangle_{\mathrm{F}_{\mathcal{H}}} &= \sum_{i=1}^{n}\sum_{j=1}^{n}\langle [\boldsymbol{X} \odot \boldsymbol{\Psi}]_{ij}, \boldsymbol{\Xi}_{ij} \rangle_{\mathcal{H}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\langle \sum_{k=1}^{n} \boldsymbol{X}_{ik}\boldsymbol{\Psi}_{kj}, \boldsymbol{\Xi}_{ij} \rangle_{\mathcal{H}} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\langle \boldsymbol{\Psi}_{kj}, \boldsymbol{X}_{ik}\boldsymbol{\Xi}_{ij} \rangle_{\mathcal{H}} = \sum_{j=1}^{n}\sum_{k=1}^{n}\langle \boldsymbol{\Psi}_{kj}, \sum_{i=1}^{n} \boldsymbol{X}_{ik}\boldsymbol{\Xi}_{ij} \rangle_{\mathcal{H}} \\ &= \sum_{j=1}^{n}\sum_{k=1}^{n}\langle \boldsymbol{\Psi}_{kj}, [\boldsymbol{X}^T \odot \boldsymbol{\Xi}]_{kj} \rangle_{\mathcal{H}} = \langle \boldsymbol{\Psi}, \boldsymbol{X}^T \odot \boldsymbol{\Xi} \rangle_{\mathrm{F}_{\mathcal{H}}}. \end{aligned} \tag{5.15}$$

□

5.3 Kernelized Graph Matching

Before deriving kernelized graph matching, we first present an assumption.

Assumption 5.1. *We assume that the edge affinity function $k^E : \mathbb{R}^{d_E} \times \mathbb{R}^{d_E} \to \mathbb{R}$ is a kernel. That is, there exist both an RKHS, $\mathcal{H}$, and an (implicit) feature map, $\psi : \mathbb{R}^{d_E} \to \mathcal{H}$, such that $k^E(\vec{\boldsymbol{q}}^1, \vec{\boldsymbol{q}}^2) = \langle \psi(\vec{\boldsymbol{q}}^1), \psi(\vec{\boldsymbol{q}}^2) \rangle_{\mathcal{H}}$, $\forall \vec{\boldsymbol{q}}^1, \vec{\boldsymbol{q}}^2 \in \mathbb{R}^{d_E}$.*

Note that Assumption 5.1 is rather mild, since kernel functions are powerful and popular in quantifying the similarity between attributes [28], [43].

For any graph $\mathcal{G} = \{\boldsymbol{A}, \mathcal{V}, \boldsymbol{P}, \mathcal{E}, \boldsymbol{Q}\}$, we can construct an array, $\boldsymbol{\Psi} \in \mathcal{H}^{n\times n}$:

$$\boldsymbol{\Psi}_{ij} = \begin{cases} \psi(\vec{\boldsymbol{q}}_{ij}) \in \mathcal{H}, & \text{if } (v_i, v_j) \in \mathcal{E} \\ 0_{\mathcal{H}} \quad \in \mathcal{H}, & \text{otherwise} \end{cases}, \text{where } 0_{\mathcal{H}} \text{ is the zero vector in } \mathcal{H}. \tag{5.16}$$

where $0_{\mathcal{H}}$ is the zero vector in $\mathcal{H}$, satisfying $\langle 0_{\mathcal{H}}, \varphi\rangle_{\mathcal{H}}$, $\forall \varphi \in \mathcal{H}^{n\times n}$. Given two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, let $\boldsymbol{\Psi}^{(1)}$ and $\boldsymbol{\Psi}^{(2)}$ be the corresponding Hilbert arrays of $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively. Then the edge similarity term in Lawler's QAP (see (5.2)) can be written as

$$\begin{aligned} &\sum_{e^1_{ij}\in\mathcal{E}_1, e^2_{ab}\in\mathcal{E}_2} k^E(\vec{\boldsymbol{q}}^1_{ij}, \vec{\boldsymbol{q}}^2_{ab})\boldsymbol{X}_{ia}\boldsymbol{X}_{jb} \\ =&\sum_{e^1_{ij}\in\mathcal{E}_1, e^2_{ab}\in\mathcal{E}_2} \langle\psi(\vec{\boldsymbol{q}}^1_{ij}), \psi(\vec{\boldsymbol{q}}^2_{ab}))\rangle_{\mathcal{H}}\boldsymbol{X}_{ia}\boldsymbol{X}_{jb} \\ =&\sum_{i,j=1}^{n}\sum_{a,b=1}^{n}\langle\boldsymbol{\Psi}^{(1)}_{ij}, \boldsymbol{\Psi}^{(2)}_{ab}\rangle_{\mathcal{H}}\boldsymbol{X}_{ia}\boldsymbol{X}_{jb} \\ =&\sum_{i,b=1}^{n}\langle\sum_{j=1}^{n}\boldsymbol{\Psi}^{(1)}_{ij}\boldsymbol{X}_{jb}, \sum_{a=1}^{n}\boldsymbol{X}_{ia}\boldsymbol{\Psi}^{(2)}_{ab}\rangle_{\mathcal{H}_{\mathcal{K}}} \\ =&\langle\boldsymbol{\Psi}^{(1)}\odot\boldsymbol{X}, \boldsymbol{X}\odot\boldsymbol{\Psi}^{(2)}\rangle_{\mathrm{F}_{\mathcal{H}}}, \end{aligned} \tag{5.17}$$

which shares a similar form with (5.1), and can be considered as the Koopmans-Beckmann's alignment between the Hilbert arrays $\boldsymbol{\Psi}^{(1)}$ and $\boldsymbol{\Psi}^{(2)}$. The last term in (5.17) is just the Frobenius inner product between two Hilbert arrays after permutation. Adding the node

affinity term, we write Laweler's QAP as[9]:

$$\min \; J_{\mathrm{gm}}(\boldsymbol{X}) = -\langle \boldsymbol{K}^N, \boldsymbol{X} \rangle_{\mathrm{F}} - \langle \boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X}, \boldsymbol{X} \odot \boldsymbol{\Psi}^{(2)} \rangle_{\mathrm{F}_{\mathcal{H}}} \quad \text{s.t. } \boldsymbol{X} \in \mathcal{P}. \tag{5.18}$$

5.3.1 Convex and Concave Relaxations

The form (5.18) inspires an intuitive way to develop convex and concave relaxations. To do this, we first introduce an auxiliary function

$$J_{\mathrm{aux}}(\boldsymbol{X}) = \frac{1}{2}\langle \boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X}, \boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X} \rangle_{\mathrm{F}_{\mathcal{H}}} + \frac{1}{2}\langle \boldsymbol{X} \odot \boldsymbol{\Psi}^{(2)}, \boldsymbol{X} \odot \boldsymbol{\Psi}^{(2)} \rangle_{\mathrm{F}_{\mathcal{H}}}.$$

Applying Corollary 5.1 and 5.2, for any $\boldsymbol{X} \in \mathcal{P}$, which satisfies $\boldsymbol{X}\boldsymbol{X}^T = \boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{I}$, we have

$$\begin{aligned} J_{\mathrm{aux}}(\boldsymbol{X}) =& \frac{1}{2}\langle \boldsymbol{\Psi}^{(1)}, \boldsymbol{\Psi}^{(1)} \odot (\boldsymbol{X}\boldsymbol{X}^T) \rangle_{\mathrm{F}_{\mathcal{H}}} + \frac{1}{2}\langle \boldsymbol{\Psi}^{(2)}, (\boldsymbol{X}^T\boldsymbol{X}) \odot \boldsymbol{\Psi}^{(2)} \rangle_{\mathrm{F}_{\mathcal{H}}} \\ =& \frac{1}{2}\|\boldsymbol{\Psi}^{(1)}\|^2_{\mathrm{F}_{\mathcal{H}}} + \frac{1}{2}\|\boldsymbol{\Psi}^{(2)}\|^2_{\mathrm{F}_{\mathcal{H}}}, \end{aligned} \tag{5.19}$$

which is always a constant. Introducing $J_{\mathrm{aux}}(\boldsymbol{X})$ to (5.18), we obtain convex and concave relaxations:

$$J_{\mathrm{vex}}(\boldsymbol{X}) = J_{\mathrm{gm}}(\boldsymbol{X}) + J_{\mathrm{aux}}(\boldsymbol{X}) = -\langle \boldsymbol{K}^N, \boldsymbol{X} \rangle_{\mathrm{F}} + \frac{1}{2}\|\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X} - \boldsymbol{X} \odot \boldsymbol{\Psi}^{(2)}\|^2_{\mathrm{F}_{\mathcal{H}}}, \tag{5.20}$$

$$J_{\mathrm{cav}}(\boldsymbol{X}) = J_{\mathrm{gm}}(\boldsymbol{X}) - J_{\mathrm{aux}}(\boldsymbol{X}) = -\langle \boldsymbol{K}^N, \boldsymbol{X} \rangle_{\mathrm{F}} - \frac{1}{2}\|\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X} + \boldsymbol{X} \odot \boldsymbol{\Psi}^{(2)}\|^2_{\mathrm{F}_{\mathcal{H}}}. \tag{5.21}$$

The convexity of $J_{\mathrm{vex}}(\boldsymbol{X})$ is easy to conclude, because the composite function of the squared norm, $\|\cdot\|^2_{\mathrm{F}_{\mathcal{H}}}$, and the linear transformation, $\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X} - \boldsymbol{X} \odot \boldsymbol{\Psi}^{(2)}$, is convex. We have similarity interpretation for the concavity of $J_{\mathrm{cav}}(\boldsymbol{X})$.

[9]For convenience in developing the path-following strategy, we write it in the minimization form.

It is interesting to see that the term $\frac{1}{2}\|\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X} - \boldsymbol{X} \odot \boldsymbol{\Psi}^{(2)}\|_{\mathrm{F}_{\mathcal{H}}}$ in (5.20) is just the distance between Hilbert arrays. If we set the map $\psi(x) = x$, then the convex relaxation of (5.1) is recovered (see [89]).

5.3.2 Path-following Strategy

Leveraging these two relaxations [80], we minimize J_{gm} by successively optimizing a series of sub-problems parameterized by $\alpha \in [0, 1]$:

$$\min J_\alpha(\boldsymbol{X}) = (1-\alpha)J_{\mathrm{vex}}(\boldsymbol{X}) + \alpha J_{\mathrm{cav}}(\boldsymbol{X}) \quad \text{s.t. } \boldsymbol{X} \in \mathcal{D} = \{\boldsymbol{X} \in \mathbb{R}_+^{n\times n} | \boldsymbol{X}\mathbf{1} = \mathbf{1}, \boldsymbol{X}^T\mathbf{1} = \mathbf{1}\}, \tag{5.22}$$

where $\mathcal{D}$ is the double stochastic relaxation of the permutation matrix set, $\mathcal{P}$. We start at $\alpha = 0$ and find the unique minimum. Then we gradually increase α until $\alpha = 1$. That is, we optimize $J_{\alpha+\triangle\alpha}$ with the local minimizer of J_α as the initial point. Finally, we output the local minimizer of $J_{\alpha=1}$. We refer to [80], [3], and [83] for detailed descriptions and improvements.

5.4 Gradient Computations

If we use the first-order optimization methods, we need only the gradients.

Proposition 5.2. *The gradient of $J_\alpha(\boldsymbol{X})$ is*

$$\triangledown J_\alpha(\boldsymbol{X}) = (1-2\alpha)\big[(\boldsymbol{\Psi}^{(1)} * \boldsymbol{\Psi}^{(1)})\boldsymbol{X} + \boldsymbol{X}(\boldsymbol{\Psi}^{(2)} * \boldsymbol{\Psi}^{(2)})\big] - 2(\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X}) * \boldsymbol{\Psi}^{(2)} - \boldsymbol{K}^N, \tag{5.23}$$

where

$$\forall i,j=1,2,...,n,\quad [\boldsymbol{\Psi}^{(1)}*\boldsymbol{\Psi}^{(1)}]_{ij}=\sum_{e^1_{ik},e^1_{kj}\in\mathcal{E}_1} k^E(\vec{\boldsymbol{q}}^1_{ik},\vec{\boldsymbol{q}}^1_{kj}), \tag{5.24}$$

$$\forall a,b=1,2,...,n,\quad [\boldsymbol{\Psi}^{(2)}*\boldsymbol{\Psi}^{(2)}]_{ab}=\sum_{e^2_{ac},e^2_{cb}\in\mathcal{E}_2} k^E(\vec{\boldsymbol{q}}^2_{ac},\vec{\boldsymbol{q}}^2_{cb}), \tag{5.25}$$

$$\text{and } \forall i,a=1,2,...,n,\quad [(\boldsymbol{\Psi}^{(1)}\odot\boldsymbol{X})*\boldsymbol{\Psi}^{(2)}]_{ia}=\sum_{e^1_{ik}\in\mathcal{E}_1,e^2_{ca}\in\mathcal{E}_2} \boldsymbol{X}_{kc}k^E(\vec{\boldsymbol{q}}^1_{ik},\vec{\boldsymbol{q}}^2_{ca}). \tag{5.26}$$

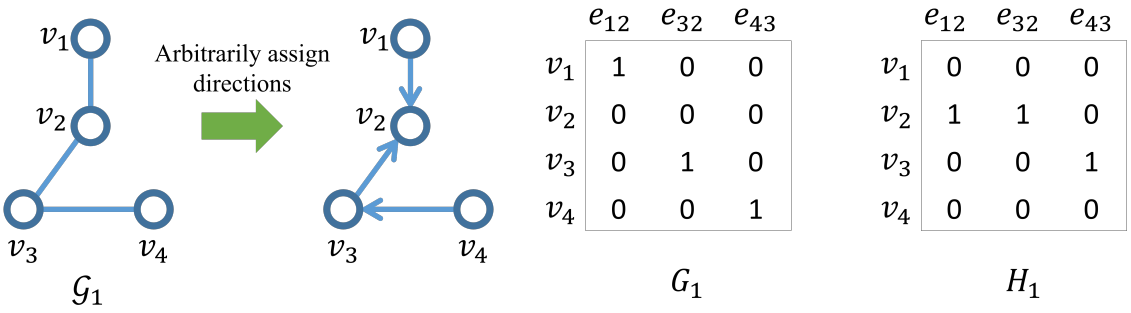
Proof. See Appendix C. □

The expressions in Proposition 5.2 show how to compute the gradient in a componentwise way. The gradient can also be expressed in a compact matrix multiplication form.

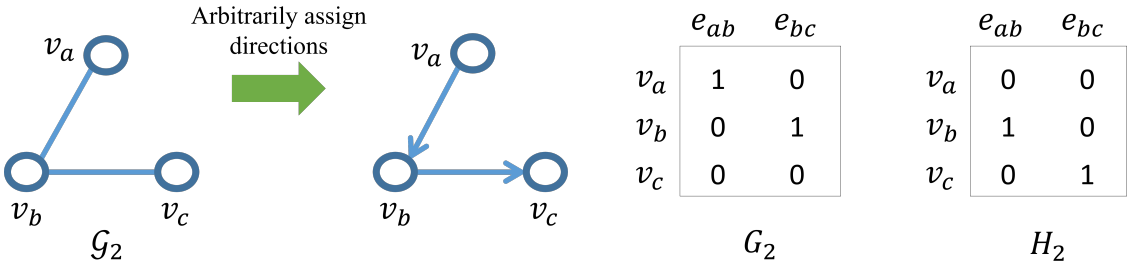
5.4.1 Gradients in Compact Matrix Multiplication Forms

In this section, we rewrite the terms of (5.23) in compact matrix multiplication forms, providing a convenient way to compute gradients. We first give some necessary definitions.

1. Given a graph $\mathcal{G}=\{\boldsymbol{A},\mathcal{V},\boldsymbol{P},\mathcal{E},\boldsymbol{Q}\}$ of n nodes and m edges. We define the Head-incidence matrix $\boldsymbol{G}\in\{0,1\}^{n\times m}$ and the Tail-incidence matrix $\boldsymbol{H}\in\{0,1\}^{n\times m}$. For any edge $e_{ij}\in\mathcal{E}$, we arbitrarily assign a direction on e_{ij}, e.g., $v_i\to v_j$ or $v_j\to v_i$. Suppose that the artifically assigned direction of e_{ij} is $v_j\to v_i$, then the items $\boldsymbol{G}(v_j,e_{ij})=1$ and $\boldsymbol{H}(v_i,e_{ij})=1$. A toy example is shown in Fig. 5.2.
2. Given two graphs $\mathcal{G}_1=\{\boldsymbol{A}_1,\mathcal{V}_1,\boldsymbol{P}_1,\mathcal{E}_1,\boldsymbol{Q}_1\}$ of n_1 nodes and m_1 edges, and $\mathcal{G}_2=\{\boldsymbol{A}_2,\mathcal{V}_2,\boldsymbol{P}_2,\mathcal{E}_2,\boldsymbol{Q}_2\}$ of n_2 nodes and m_2 edges, let $\boldsymbol{K}^E_{11}\in\mathbb{R}^{m_1\times m_1}$, $\boldsymbol{K}^E_{22}\in\mathbb{R}^{m_2\times m_2}$, and $\boldsymbol{K}^E_{12}\in\mathbb{R}^{m_1\times m_2}$ be three kernel matrices induced by the kernel k^E (the edge affinity



(a)



(b)

Figure 5.2: (a) A toy Graph $\mathcal{G}_1$, and its Head-incidence matrix $\boldsymbol{G}_1$ and Tail-incidence matrix $\boldsymbol{H}_1$; (b) A toy Graph $\mathcal{G}_2$, and its Head-incidence matrix $\boldsymbol{G}_2$ and Tail-incidence matrix $\boldsymbol{H}_2$.

function). They are defined such that

$$[\boldsymbol{K}_{11}^E](e_{i_1j_1}^1, e_{i_2j_2}^1) = k^E(\vec{\boldsymbol{q}}_{i_1j_1}^1, \vec{\boldsymbol{q}}_{i_2j_2}^1), \quad \text{if } e_{i_1j_1}^1, e_{i_2j_2}^1 \in \mathcal{E}_1, \tag{5.27}$$

$$[\boldsymbol{K}_{22}^E](e_{a_1b_1}^2, e_{a_2b_2}^2) = k^E(\vec{\boldsymbol{q}}_{a_1b_1}^2, \vec{\boldsymbol{q}}_{a_2b_2}^2), \quad \text{if } e_{a_1b_1}^2, e_{a_2b_2}^2 \in \mathcal{E}_2, \tag{5.28}$$

$$[\boldsymbol{K}_{12}^E](e_{ij}^1, e_{ab}^2) = k^E(\vec{\boldsymbol{q}}_{ij}^1, \vec{\boldsymbol{q}}_{ab}^2), \quad \text{if } e_{ij}^1 \in \mathcal{E}_1, e_{ab}^2 \in \mathcal{E}_2, \tag{5.29}$$

Proposition 5.3. *Let $\boldsymbol{G}_1$ and $\boldsymbol{H}_1$, and $\boldsymbol{G}_2$ and $\boldsymbol{H}_2$ be the Head-incidence matrix and the Tail-incidence matrix of graph $\mathcal{G}_1$ and $\mathcal{G}_2$ (see Fig. 5.2), respectively. Then the terms in* (5.23) *can be written as*

$$\begin{aligned}
\boldsymbol{\Psi}^{(1)} * \boldsymbol{\Psi}^{(1)} &= \boldsymbol{H}_1(\boldsymbol{G}_1^T\boldsymbol{G}_2 \circ \boldsymbol{K}_{11}^E)\boldsymbol{H}_2^T + \boldsymbol{H}_1(\boldsymbol{G}_1^T\boldsymbol{H}_2 \circ \boldsymbol{K}_{11}^E)\boldsymbol{G}_2^T \\
&\quad + \boldsymbol{G}_1(\boldsymbol{H}_1^T\boldsymbol{G}_2 \circ \boldsymbol{K}_{11}^E)\boldsymbol{H}_2^T + \boldsymbol{G}_1(\boldsymbol{H}_1^T\boldsymbol{H}_2 \circ \boldsymbol{K}_{11}^E)\boldsymbol{G}_2^T, \qquad (5.30) \\
\boldsymbol{\Psi}^{(2)} * \boldsymbol{\Psi}^{(2)} &= \boldsymbol{H}_1(\boldsymbol{G}_1^T\boldsymbol{G}_2 \circ \boldsymbol{K}_{22}^E)\boldsymbol{H}_2^T + \boldsymbol{H}_1(\boldsymbol{G}_1^T\boldsymbol{H}_2 \circ \boldsymbol{K}_{22}^E)\boldsymbol{G}_2^T \\
&\quad + \boldsymbol{G}_1(\boldsymbol{H}_1^T\boldsymbol{G}_2 \circ \boldsymbol{K}_{22}^E)\boldsymbol{H}_2^T + \boldsymbol{G}_1(\boldsymbol{H}_1^T\boldsymbol{H}_2 \circ \boldsymbol{K}_{22}^E)\boldsymbol{G}_2^T, \qquad (5.31) \\
\text{and } (\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X}) * \boldsymbol{\Psi}^{(2)} &= \boldsymbol{H}_1(\boldsymbol{G}_1^T\boldsymbol{X}\boldsymbol{G}_2 \circ \boldsymbol{K}_{12}^E)\boldsymbol{H}_2^T + \boldsymbol{H}_1(\boldsymbol{G}_1^T\boldsymbol{X}\boldsymbol{H}_2 \circ \boldsymbol{K}_{12}^E)\boldsymbol{G}_2^T \\
&\quad + \boldsymbol{G}_1(\boldsymbol{H}_1^T\boldsymbol{X}\boldsymbol{G}_2 \circ \boldsymbol{K}_{12}^E)\boldsymbol{H}_2^T + \boldsymbol{G}_1(\boldsymbol{H}_1^T\boldsymbol{X}\boldsymbol{H}_2 \circ \boldsymbol{K}_{12}^E)\boldsymbol{G}_2^T. \qquad (5.32)
\end{aligned}$$

Proof. See Appendix C. □

5.5 Approximate Kernelized Graph Matching

Based on the above discussion, we significantly reduce the space cost of Lawler's QAP by avoiding computing the affinity matrix $\boldsymbol{K} \in \mathbb{R}^{n^2 \times n^2}$. However, the time cost of computing gradient with (5.23) is $O(n^4)$, which can be further reduced by employing the approximate explicit feature maps (see Chapter 2.2.3).

For the kernel $k^E : \mathbb{R}^{d_E} \times \mathbb{R}^{d_E} \to \mathbb{R}$, we may find an explicit feature map $\hat{\psi} : \mathbb{R}^{d_E} \to \mathbb{R}^D$, such that

$$\forall\ \vec{\boldsymbol{q}}^1, \vec{\boldsymbol{q}}^2 \in \mathbb{R}^{d_E},\ \langle \hat{\psi}(\vec{\boldsymbol{q}}^1), \hat{\psi}(\vec{\boldsymbol{q}}^2) \rangle = \hat{k}^E(\vec{\boldsymbol{q}}^1, \vec{\boldsymbol{q}}^2) \approx k^E(\vec{\boldsymbol{q}}^1, \vec{\boldsymbol{q}}^2). \tag{5.33}$$

For example, if $k^E(\vec{\boldsymbol{q}}^1, \vec{\boldsymbol{q}}^2) = \exp(-\gamma \|\vec{\boldsymbol{q}}^1 - \vec{\boldsymbol{q}}^2\|_2^2)$, then $\hat{\psi}$ is the Fourier random feature map [34]:

$$\hat{\psi}(\vec{\boldsymbol{q}}) = \sqrt{\frac{2}{D}} \big[\cos(\omega_1^T \vec{\boldsymbol{q}} + b_1), ..., \cos(\omega_D^T \vec{\boldsymbol{q}} + b_D) \big]^T,\ \text{where } \omega_i \sim N(\vec{\boldsymbol{0}}, \gamma^2 \boldsymbol{I}) \text{ and } b_i \sim U[0, 1]. \tag{5.34}$$

Note that in practice, the performance of $\hat{\psi}$ is good enough for relatively small values of D [28]. By the virtue of explicit feature maps, we obtain a new graph representation $\hat{\boldsymbol{\Psi}} \in (\mathbb{R}^D)^{n \times n}$:

$$\hat{\boldsymbol{\Psi}}_{ij} = \begin{cases} \hat{\psi}(\vec{\boldsymbol{q}}_{ij}) \in \mathbb{R}^D, & \text{if } (v_i, v_j) \in \mathcal{E} \\ \vec{\boldsymbol{0}} \quad\ \in \mathbb{R}^D, & \text{otherwise} \end{cases}, \text{where } \vec{\boldsymbol{0}} \text{ is the zero vector in } \mathbb{R}^D. \tag{5.35}$$

Its space cost is $O(Dn^2)$, where $D \ll n$.

5.5.1 Approximated Lawler's Formulation

The Lawler's formulation (5.18) can be approximated by

$$\min\ \hat{J}_{\text{gm}}(\boldsymbol{X}) = -\langle \boldsymbol{K}^N, \boldsymbol{X} \rangle_{\text{F}} - \langle \hat{\boldsymbol{\Psi}}^{(1)} \odot \boldsymbol{X}, \boldsymbol{X} \odot \hat{\boldsymbol{\Psi}}^{(2)} \rangle_{\text{F}_{\mathcal{H}}} \quad \text{s.t. } \boldsymbol{X} \in \mathcal{P}. \tag{5.36}$$

The convex and concave relaxations become

$$\hat{J}_{\text{vex}}(\boldsymbol{X}) = -\langle \boldsymbol{K}^N, \boldsymbol{X} \rangle_{\text{F}} + \frac{1}{2} \| \hat{\boldsymbol{\Psi}}^{(1)} \odot \boldsymbol{X} - \boldsymbol{X} \odot \hat{\boldsymbol{\Psi}}^{(2)} \|^2_{\text{F}_{\mathcal{H}}}, \tag{5.37}$$

$$\hat{J}_{\text{cav}}(\boldsymbol{X}) = -\langle \boldsymbol{K}^N, \boldsymbol{X} \rangle_{\text{F}} - \frac{1}{2} \| \hat{\boldsymbol{\Psi}}^{(1)} \odot \boldsymbol{X} + \boldsymbol{X} \odot \hat{\boldsymbol{\Psi}}^{(2)} \|^2_{\text{F}_{\mathcal{H}}}. \tag{5.38}$$

The path-following optimization problem become

$$\min \hat{J}_\alpha(\boldsymbol{X}) = (1-\alpha)\hat{J}_{\text{vex}}(\boldsymbol{X}) + \alpha \hat{J}_{\text{cav}}(\boldsymbol{X}) \quad \text{s.t. } \boldsymbol{X} \in \mathcal{D} = \{\boldsymbol{X} \in \mathbb{R}_+^{n \times n} | \boldsymbol{X}\mathbf{1} = \mathbf{1}, \boldsymbol{X}^T\mathbf{1} = \mathbf{1}\}, \tag{5.39}$$

The first-order gradient of (5.39) become

$$\nabla \hat{J}_\alpha(\boldsymbol{X}) = (1-2\alpha)\big[(\hat{\boldsymbol{\Psi}}^{(1)} * \hat{\boldsymbol{\Psi}}^{(1)})\boldsymbol{X} + \boldsymbol{X}(\hat{\boldsymbol{\Psi}}^{(2)} * \hat{\boldsymbol{\Psi}}^{(2)})\big] - 2(\hat{\boldsymbol{\Psi}}^{(1)} \odot \boldsymbol{X}) * \hat{\boldsymbol{\Psi}}^{(2)} - \boldsymbol{K}^N, \tag{5.40}$$

Now computing the gradient-related terms $\hat{\boldsymbol{\Psi}}^{(1)} * \hat{\boldsymbol{\Psi}}^{(1)}$, $\hat{\boldsymbol{\Psi}}^{(2)} * \hat{\boldsymbol{\Psi}}^{(2)}$, and $(\hat{\boldsymbol{\Psi}}^{(1)} \odot \boldsymbol{X}) * \hat{\boldsymbol{\Psi}}^{(2)}$ in (5.23) becomes rather simple. We first slice $\hat{\boldsymbol{\Psi}}^{(1)}$ (and likewise $\hat{\boldsymbol{\Psi}}^{(2)}$) into D matrices $\hat{\boldsymbol{\Psi}}^{(1)}(:,:,i) \in \mathbb{R}^{n \times n}$, $i = 1, 2, ..., D$. Then it can be easily shown that

$$\hat{\boldsymbol{\Psi}}^{(1)} * \hat{\boldsymbol{\Psi}}^{(1)} = \sum_{i=1}^{D} \hat{\boldsymbol{\Psi}}^{(1)}(:,:,i)\hat{\boldsymbol{\Psi}}^{(1)}(:,:,i), \tag{5.41}$$

$$\hat{\boldsymbol{\Psi}}^{(2)} * \hat{\boldsymbol{\Psi}}^{(2)} = \sum_{i=1}^{D} \hat{\boldsymbol{\Psi}}^{(2)}(:,:,i)\hat{\boldsymbol{\Psi}}^{(2)}(:,:,i), \tag{5.42}$$

$$\text{and } (\hat{\boldsymbol{\Psi}}^{(1)} \odot \boldsymbol{X}) * \hat{\boldsymbol{\Psi}}^{(2)} = \sum_{i=1}^{D} \hat{\boldsymbol{\Psi}}^{(1)}(:,:,i)\boldsymbol{X}\hat{\boldsymbol{\Psi}}^{(2)}(:,:,i), \tag{5.43}$$

whose the first and second term respectively involves D and $2D$ matrix multiplications of the size $n \times n$. Hence, the time complexity is reduced to $O(Dn^3)$. Moreover, gradient computations with (5.41) are highly parallizable, which also contributes to scalability.

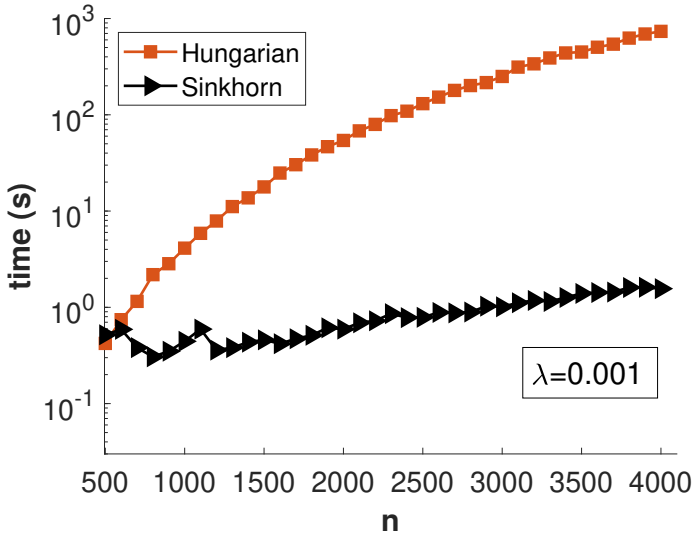


Figure 5.3: Hungarian vs Sinkhorn.

5.6 Entropy-regularized Frank-Wolfe Optimization Algorithm

In this part, we develop the Entropy-regularized Frank-Wolfe algorithm for optimizing (approximate) kernelized Lawler's problem (5.22) (or (5.39)). The only difference between the procedures of minimizing $J_\alpha(\boldsymbol{X})$ and $\hat{J}_\alpha(\boldsymbol{X})$ comes from their first-order gradients (see (5.23) and (5.40)). Therefore, without of loss of generality, we consider only optimizing $J_\alpha(\boldsymbol{X})$.

The state-of-the-art method for optimizing problem (5.22) is the Frank-Wolfe algorithm [77, 90, 91, 92], whose every iteration involves linear programming to obtain optimal direction $\boldsymbol{Y}^*$, i.e.,

$$\boldsymbol{Y}^* = \text{argmin}_{\boldsymbol{Y}\in\mathcal{D}} \langle \nabla J_\alpha(\boldsymbol{X}), \boldsymbol{Y}\rangle_{\text{F}}, \tag{5.44}$$

which is usually solved by the Hungarian algorithm [93]. Optimizing J_α may need to call the Hungarian algorithm many times, which is quite time-consuming for large graphs. In

this work, instead of minimizing $J_\alpha(\boldsymbol{X})$ in (5.22), we consider the following problem,

$$\min_{\boldsymbol{X}} \quad F_\alpha(\boldsymbol{X}) = J_\alpha(\boldsymbol{X}) + \lambda H(\boldsymbol{X}) \quad \text{s.t. } \boldsymbol{X} \in \mathcal{D}_n, \tag{5.45}$$

where $\mathcal{D}_n = \{\boldsymbol{X} \in \mathbb{R}_+^{n\times n} | \boldsymbol{X}\mathbf{1} = \frac{1}{n}\mathbf{1}, \boldsymbol{X}^T\mathbf{1} = \frac{1}{n}\mathbf{1}\}$, $H(\boldsymbol{X}) = \sum_{i,j=1}^n \boldsymbol{X}_{ij} \log \boldsymbol{X}_{ij}$ is the negative entropy, and the node affinity matrix $\boldsymbol{K}^N$ in $J_\alpha(\boldsymbol{X})$ (see (5.18) and (5.22)) is normalized as $\boldsymbol{K}^N \to \frac{1}{n}\boldsymbol{K}^N$ to balance the node and edge affinity terms. The observation is that if λ is set to be small enough, the solution of (5.45), after being multiplied by n, will approximate that of the original QAP (5.22) as much as possible. We design the entropy-regularized Frank-Wolfe algorithm ("EnFW" for short) for optimizing (5.45), in each outer iteration of which we solve the following nonlinear problem.

$$\min \langle \nabla J_\alpha(\boldsymbol{X}), \boldsymbol{Y} \rangle_{\mathrm{F}} + \lambda H(\boldsymbol{Y}) \quad \text{s.t. } \boldsymbol{Y} \in \mathcal{D}_n. \tag{5.46}$$

Note that (5.46) can be extremely efficiently solved by the Sinkhorn-Knopp algorithm [94]. Theoretically, the Sinkhorn-Knopp algorithm converges at the linear rate, i.e.,

$$0 < \limsup \frac{\|\boldsymbol{Y}_{k+1} - \boldsymbol{Y}^*\|}{\|\boldsymbol{Y}_k - \boldsymbol{Y}^*\|} < 1. \tag{5.47}$$

An empirical comparison between the runtimes of these two algorithms is shown in Fig. 5.3, where we can see that the Sinkhorn-Knopp algorithm for solving (5.46) is much faster than the Hungarian algorithm for solving (5.44).

5.6.1 Description of the EnFW Algorithm

We first give necessary definitions. Write the quadratic function

$$\begin{aligned} &J_\alpha(\boldsymbol{X}+s(\boldsymbol{Y}-\boldsymbol{X})) \\ =&J_\alpha(\boldsymbol{X})+s\langle\triangledown J_\alpha(\boldsymbol{X}),\boldsymbol{Y}-\boldsymbol{X}\rangle_{\mathrm{F}}+\frac{1}{2}\mathrm{vec}(\boldsymbol{Y}-\boldsymbol{X})^T\nabla^2 J_\alpha(\boldsymbol{X})\mathrm{vec}(\boldsymbol{Y}-\boldsymbol{X})s^2. \end{aligned} \tag{5.48}$$

Then, we define the coefficient of the quadratic term as

$$Q(\boldsymbol{X},\boldsymbol{Y}) \triangleq \frac{1}{2}\mathrm{vec}(\boldsymbol{Y}-\boldsymbol{X})^T\nabla^2 J_\alpha(\boldsymbol{X})\mathrm{vec}(\boldsymbol{Y}-\boldsymbol{X}) = \frac{1}{2}\langle\triangledown J_\alpha(\boldsymbol{Y}-\boldsymbol{X}),\boldsymbol{Y}-\boldsymbol{X}\rangle_{\mathrm{F}}, \tag{5.49}$$

where the second equality holds because J_α is a quadratic function. Next, similar to the original FW algorithm, we define the gap function $g(\boldsymbol{X})$ as

$$g(\boldsymbol{X}) \triangleq \langle\triangledown J_\alpha(\boldsymbol{X}),\boldsymbol{X}\rangle_{\mathrm{F}}+\lambda H(\boldsymbol{X})-\min_{\boldsymbol{Y}\in\mathcal{D}_n}\langle\triangledown J_\alpha(\boldsymbol{X}),\boldsymbol{Y}\rangle_{\mathrm{F}}+\lambda H(\boldsymbol{Y}). \tag{5.50}$$

Note that it is straightforward to conclude that $g(\boldsymbol{X})$ is nonnegative.

Proposition 5.4. *If $\boldsymbol{X}^*$ is an optimal solution of* (5.45)*, then $g(\boldsymbol{X}^*)=0$.*

Therefore, the gap function characterize the necessary condition for optimal solutions. Note that for any $\boldsymbol{X}\in\mathcal{D}_n$, if $g(\boldsymbol{X})=0$, then we say "$\boldsymbol{X}$ is a first-order stationary point". Now with the definitions of $Q(\boldsymbol{X},\boldsymbol{Y})$ and $g(\boldsymbol{X})$, we detail the EnFW procedure in Algorithm 4.

After obtaining the optimal solution path $\boldsymbol{X}_\alpha^*$, $\alpha=0:\triangle\alpha:1$, we discretize $n\boldsymbol{X}_1^*$ by the Hungarian [93] or the greedy discretization algorithm [95] to get the binary matching matrix. We next highlight the differences between the EnFW algorithm and the original FW algorithm: (i) We find the optimal direction by solving a nonlinear convex problem (5.46)

Algorithm 4 The EnFW optimization algorithm for minimizing F_α (5.45)

Initialize $\boldsymbol{X}_0 \in \mathcal{D}_n$ not converge
Compute the gradient $\nabla J_\alpha(\boldsymbol{X}_t)$ based on (5.23) or (5.41),
Obtain the optimal direction $\boldsymbol{Y}_t$ by solving (5.46), i.e.,
$\boldsymbol{Y}_t = \text{argmin}_{\boldsymbol{Y} \in \mathcal{D}_n} \langle \nabla J_\alpha(\boldsymbol{X}_t), \boldsymbol{Y} \rangle_{\text{F}} + \lambda H(\boldsymbol{Y})$,
Compute $G_t = g(\boldsymbol{X}_t)$ and $Q_t = Q(\boldsymbol{X}_t, \boldsymbol{Y}_t)$,
Determine the stepsize s_t: If $Q_t \leq 0$; $s_t = 1$, else $s_t = \min\{G_t/(2Q_t), 1\}$,
Update $\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + s_t(\boldsymbol{Y}_t - \boldsymbol{X}_t)$.
end
Output the solution $\boldsymbol{X}^*_\alpha$.

with the efficient Sinkhorn-Knopp algorithm, instead of solving the linear problem (5.44). (ii) We give an explicit formula for computing the stepsize s, instead of making a linear search on $[0, 1]$ for optimizing $F_\alpha(\boldsymbol{X} + s(\boldsymbol{Y} - \boldsymbol{X}))$ or estimating the Lipschitz constant of ∇F_α [96].

5.6.2 Convergence Analysis

In this part, we present the convergence properties of the proposed EnFW algorithm, including the sequentially decreasing property of the objective function and the convergence rates.

Theorem 5.1. *The generated objective function value sequence, $\{F_\alpha(\boldsymbol{X}_t)\}_{t=0}$, will decreasingly converge. The generated points sequence, $\{\boldsymbol{X}_t\}_{t=0} \subseteq \mathcal{D}_n \subseteq \mathbb{R}^{n \times n}$, will weakly converge to the first-order stationary point, at the rate $O(\frac{1}{\sqrt{t+1}})$, i.e,*

$$\min_{1 \leq t \leq T} g(\boldsymbol{X}_t) \leq \frac{2 \max\{\triangle_0, \sqrt{L\triangle_0/n}\}}{\sqrt{T+1}}, \tag{5.51}$$

where $\triangle_0 = F_\alpha(\boldsymbol{X}_0) - \min_{X \in \mathcal{D}_n} F_\alpha(\boldsymbol{X})$, and L is the largest absolute eigenvalue of $\nabla^2 J_\alpha(\boldsymbol{X})$.

Proof. See Appendix D.1. □

If $J_\alpha(\boldsymbol{X})$ is convex, which happens when α is small (see (5.22)), then we have a tighter bound $O(\frac{1}{T+1})$.

Theorem 5.2. *If $\boldsymbol{J}_\alpha(\boldsymbol{X})$ is convex, we have*

$$F_\alpha(\boldsymbol{X}_T) - F_\alpha(\boldsymbol{X}^*) \leq \frac{4L}{n(T+1)}. \tag{5.52}$$

Proof. See Appendix D.2. □

Note that in both cases, convex and non-convex, our EnFW achieves the same (up to a constant coefficient) convergence rate with the original FW algorithm (see [97] and [96]). Thanks to the efficiency of the Sinkhorn-Knopp algorithm, we need much less time to finish each iteration. Therefore, our optimization algorithm is more computationally efficient than the original FW algorithm.

5.7 Experiments

In this section, we conduct extensive experiments to demonstrate the matching performance and scalability of our kernelized graph matching framework. We implement all the algorithms using Matlab on an Intel i7-7820HQ, 2.90 GHz CPU with 64 GB RAM.

Notations: We use KerGM_I (see Section 5.3) to denote our algorithm when we use exact edge affinity kernels, and use KerGM_II (see Section 5.5) to denote it when we use approximate explicit feature maps.

Baseline methods: We compare our algorithm with many state-of-the-art graph (network) matching algorithms.

1. Integer projected fixed point method (IPFP) [90],
2. Spectral matching with affine constraints (SMAC) [98],
3. Probabilistic graph matching (PM) [85],
4. Re-weighted random walk matching (RRWM) [95],
5. Factorized graph matching (FGM) [3],
6. Branch path following for graph matching (BPFG) [83],
7. Graduated assignment graph matching (GAGM) [74],
8. Global network alignment using multiscale spectral signatures (GHOST) [99],
9. Triangle alignment (TAME) [100],
10. Maximizing accuracy in global network alignment (MAGNA) [101].

Note that GHOST, TAME, and MAGNA are popular protein-protein interaction (PPI) networks aligners.

Settings: For all the baseline methods, we used the parameters recommended in the public code. For our method, if not specified, we set the regularization parameter (see (5.45)) $\lambda = 0.005$ and the path following parameters $\alpha = 0 : 0.1 : 1$. We use the Hungarian algorithm for final discretization. We refer to Appendix E for other implementation details.

5.7.1 Synthetic Datasets

We evaluate algorithms on the synthetic Erdos–Rényi [102] random graphs, following the experimental protocol in [74, 3, 95]. For each trial, we generate two graphs: the reference

graph $\mathcal{G}_1$ and the perturbed graph $\mathcal{G}_2$, each of which has n_{in} inlier nodes and n_{out} outlier nodes. Each edge in $\mathcal{G}_1$ is randomly generated with probability $\rho \in [0, 1]$. The edges $e^1_{ij} \in \mathcal{E}_1$ are associated with the edge attributes $q^1_{ij} \sim \mathcal{U}[0, 1]$. The corresponding edge $e^2_{p(i)p(j)} \in \mathcal{E}_2$ has the attribute $q^2_{p(i)p(j)} = q^1_{ij} + \epsilon$, where p is a permutation map for inlier nodes, and $\epsilon \sim N(0, \sigma^2)$ is the Gaussian noise. For the baseline methods, the edge affinity value between q^1_{ij} and q^2_{ij} is computed as $k^E(q^1_{ij}, q^2_{ij}) = \exp(-(q^1_{ij} - q^2_{ij})^2/0.15)$. Note that for our method KerGM_{II}, we use the random Fourier features (5.34) to approximate the Gaussian kernel, and represent each graph by an $(n_{\text{in}} + n_{\text{out}}) \times (n_{\text{in}} + n_{\text{out}})$ array in $\mathbb{R}^D$. We set the parameter $\gamma = 5$ and the dimension $D = 20$.

Comparing matching accuracy We perform the comparison under three parameter settings, in all of which we set $n_{\text{in}} = 50$. Note that different from the standard protocol where $n_{\text{in}} = 20$ [3], we use relatively large graphs to highlight the advantage of our KerGM_{II}.

- We change the number of outlier nodes, n_{out}, from 0 to 50 while fixing the noise, $\sigma = 0$, and the edge density, $\rho = 1$.
- We change σ from 0 to 0.2 while fixing $n_{\text{out}} = 0$ and $\rho = 1$.
- We change ρ from 0.3 to 1 while fixing $n_{\text{out}} = 5$ and $\sigma = 0.1$.

For all cases in these settings, we repeat the experiments 100 times. In Fig. 5.4, we compare KerGM_{I} with state-of-the-arts. We report the averaged accuracies, the averaged objective function values, and their corresponding standard errors. In Fig. 5.5, we compare KerGM_{II} with state-of-the-arts. Clearly, both of our KerGM_{I} and KerGM_{II} outperform all the baseline methods with statistical significance.

Comparing scalability To fairly compare the scalability of different algorithms, we consider the exact matching between fully connected graphs, i.e., $n_{\text{out}} = 0$, $\sigma = 0$, and $\rho = 1$. We

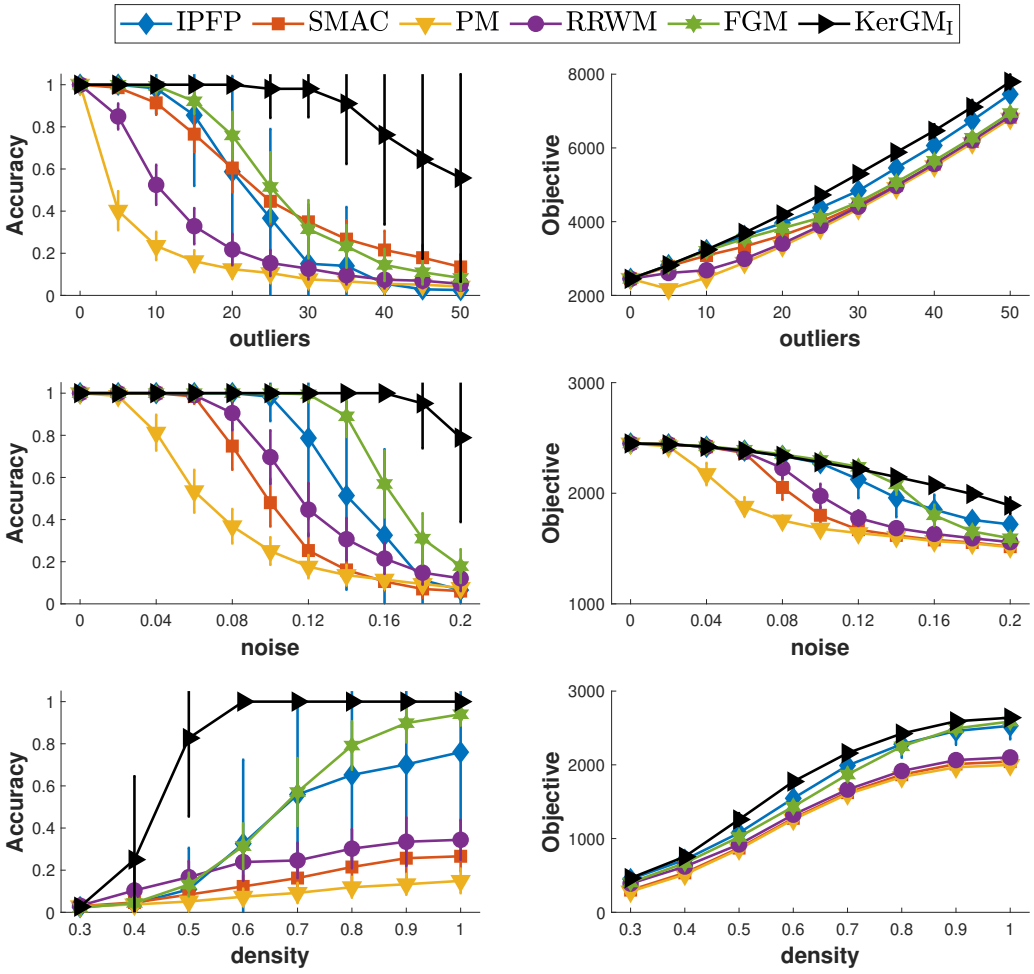


Figure 5.4: Matching results on synthetic graph dataset.

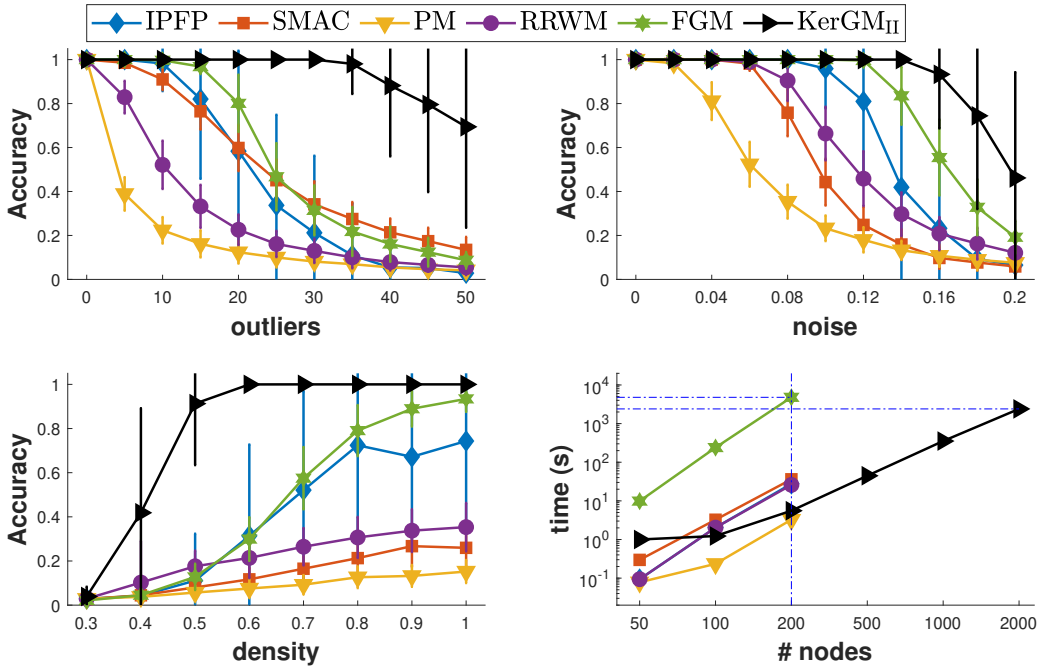
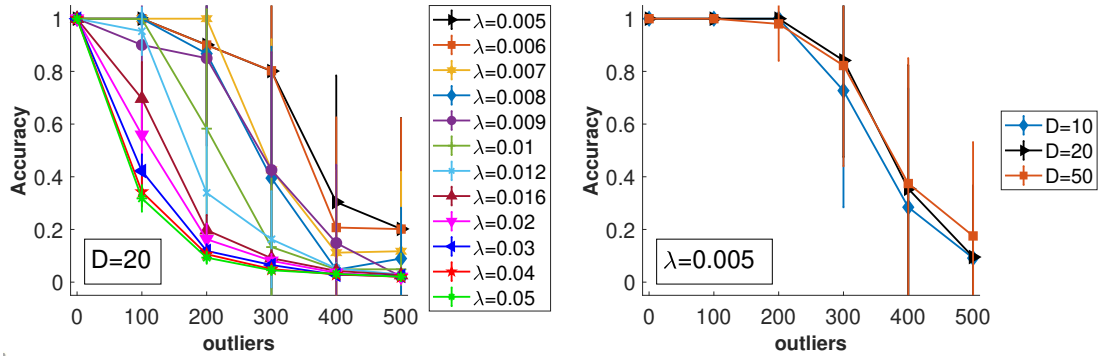


Figure 5.5: Matching results on synthetic graph dataset.

change the number of nodes, n ($= n_{\text{in}}$), from 50 to 2000, and report the CPU time of each algorithm in Fig. 5.5. We can see that all the baseline methods can handle only graphs with fewer than 200 nodes because of the expensive space cost of matrix $\boldsymbol{K}$ (see (5.3)). However, KerGM$_{\text{II}}$ can finish Lawler's graph matching problem with 2000 nodes in reasonable time.

Analyzing parameter sensitivity To analyze the parameter sensitivity of KerGM$_{\text{II}}$, we vary the regularization parameter, λ, and the dimension, D, of Fourier random features. We conduct large subgraph matching experiments by setting $n_{\text{in}} = 500$, $n_{\text{out}} = 0 : 100 : 500$, $\rho = 1$, and $\sigma = 0$. We repeat the experiments 50 times and report the average accuracies and standard errors. In Fig. 5.6, we show the results under different λ and different D. We can see that (i) smaller λ leads to better performance, which can be easily understood because the entropy regularizer will perturb the original optimal solution, and (ii) the dimension D



(a) (b)

Figure 5.6: (a) Parameter sensitivity study of the regularizer λ. (b) Parameter sensitivity study of the dimension, D, of the random Fourier feature.

does not much affect on KerGM_{II}, which implies that in practice, we can use relatively small D for reducing the time and space complexity.

5.7.2 Image Datasets

The **CMU House Sequence** dataset has 111 frames of a house, each of which has 30 labeled landmarks. We follow the experimental protocol in [3, 83]. We match all the image pairs, spaced by 0:10:90 frames. We consider two node settings: $(n_1, n_2) = (30, 30)$ and $(n_1, n_2) = (20, 30)$. We build graphs by using Delaunay triangulation [103] to connect landmarks. The edge attributes are the pairwise distances between nodes. For all methods, we compute the edge affinity as $k^E(q^1_{ij}, q^2_{ab}) = \exp(-(q^1_{ij} - q^2_{ab})^2/2500)$. In Fig. 5.7, we report the average matching accuracy and objective function (5.3) value ratio for every gap. It can be seen that on this dataset, KerGM_{I} and FGM achieve the best performance, and are slightly better than BPFG when outliers exist, i.e., $(n_1, n_2) = (20, 30)$.

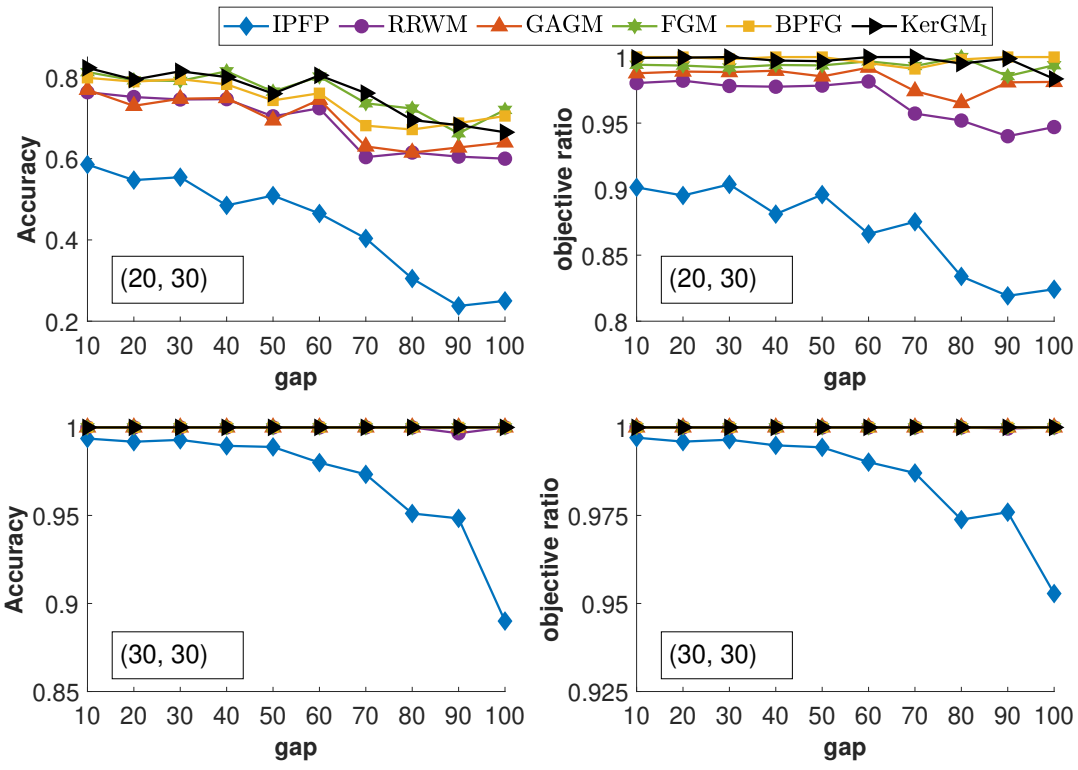


Figure 5.7: Comparison of graph matching on the CMU house dataset.

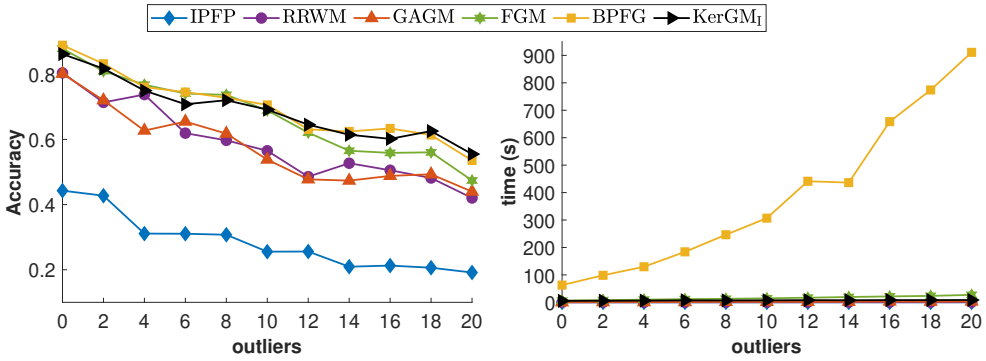
The **Pascal** dataset [104] has 20 pairs of motorbike images and 30 pairs of car images. For each pair, the detected feature points and manually labeled correspondences are provided. Following [3, 83], we randomly select 0:2:20 outliers from the background to compare different methods. For each node, v_i, its attribute, p_i, is assigned as its orientation of the normal vector at that point to the contour where the point was sampled. Nodes are connected by Delaunay triangulation [103]. For each edge, e_{ij}, its attribute $\vec{\boldsymbol{q}}_{ij}$ equals $[d_{ij}, \theta_{ij}]^T$, where d_{ij} is the distance between v_i and v_j, and θ_{ij} is the absolute angle between the edge and the horizontal line. For all methods, the node affinity is computed as $k^N(p_i, p_j) = \exp(-|p_i - p_j|)$. The edge affinity is computed as $k^E(\vec{\boldsymbol{q}}^1_{ij}, \vec{\boldsymbol{q}}^2_{ab}) = \exp(-|d^1_{ij} - d^2_{ab}|/2 - |\theta^1_{ij} - \theta^2_{ab}|/2)$. Fig. 5.8 (a) shows a matching result of KerGM_I. In Fig. 5.8 (b), we report the matching accuracies and CPU running time. From the perspective of matching accuracy, KerGM_I, BPFG, and FGM consistently outperforms other methods. When the number of outliers increases, KerGM_I and BPFG perform slightly better than FGM. However, from the perspective of running time, the time cost of BPFG is much higher than that of the others.

5.7.3 Protein-protein Interaction Network Dataset

The **S.cerevisiae (yeast) PPI network** [105] dataset is popularly used to evaluate PPI network aligners because it has known true node correspondences.

It consists of an unweighted high-confidence PPI network with 1004 proteins (nodes) and 8323 PPIs (edges), and five noisy PPI networks generated by adding 5%, 10%, 15%, 20%, 25% low-confidence PPIs. We do graph matching between the high-confidence network with every noisy network. To apply KerGM, we generate edge attributes by the heat diffusion

(a)



(b)

Figure 5.8: (a) A matching example for a pair of motorbike images generated by KerGM$_I$, where green and red lines respectively indicate correct and incorrect matches. (b) Comparison of graph matching on the Pascal dataset.

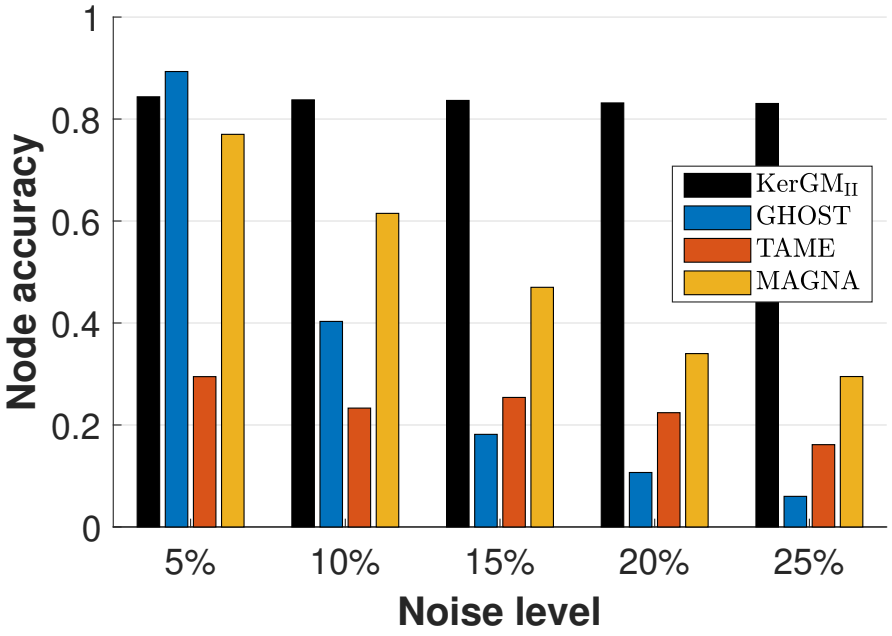


Figure 5.9: Results on PPI networks.

matrix [106, 50],

$$\boldsymbol{H}_t = \exp(-t\boldsymbol{L}) = \sum_{i=1}^{n} \exp(-\lambda_i t)\vec{\boldsymbol{u}}_i\vec{\boldsymbol{u}}_i^T \in \mathbb{R}^{n\times n}, \tag{5.53}$$

where $\boldsymbol{L}$ is the normalized Laplacian matrix [50], and $\{(\lambda_i, \vec{\boldsymbol{u}}_i)\}_{i=1}^n$ are eigenpairs of $\boldsymbol{L}$. The edge attributes vector $\vec{\boldsymbol{q}}_{ij}$ is assigned as

$$\vec{\boldsymbol{q}}_{ij} = [\boldsymbol{H}_5(i,j), \boldsymbol{H}_{10}(i,j), \boldsymbol{H}_{15}(i,j), \boldsymbol{H}_{20}(i,j)]^T \in \mathbb{R}^4. \tag{5.54}$$

We use the Fourier random features (5.34), and set $D = 50$ and $\gamma = 200$. We compare KerGM$_{\text{II}}$[10] with the state-of-the-art PPI aligners: TAME, GHOST, and MAGNA. In Fig. 5.9, we report the matching accuracies. Clearly, KerGM$_{\text{II}}$ significantly outperforms the baselines.

[10]To the best our knowledge, KerGM is the first one that uses Lawler's graph matching formulation to solve the PPI network alignment problem.

Especially when the noise level are 20% or 25%, KerGM_{II}'s accuracies are more than 50 percentages higher than those of other algorithms.

5.8 Chapter Summary

In this work, based on a mild assumption regarding edge affinity values, we provided KerGM, a unifying framework for Koopman-Beckmann's and Lawler's QAPs, within which both two QAPs can be considered as the alignment between arrays in RKHS. Then we derived convex and concave relaxations and the corresponding path-following strategy. To make KerGM more scalable to large graphs, we developed the computationally efficient entropy-regularized Frank-Wolfe optimization algorithm. KerGM achieved promising performance on both image and biology datasets. Thanks to its scalability, we believe KerGM can be potentially useful for many applications in the real world.

Chapter 6

Conclusions and Future Work

6.1 Summary and Conclusions

In this dissertation, we focused on developing algorithms for graph-structured data analysis based on the kernel theory. It consists of three parts: graph kernels, vector embedding of graphs, and graph matching.

We began by introducing the basic concept of attributed graphs and kernels and the corresponding reproducing kernel Hilbert spaces. We then proposed a new and effective graph kernel, RetGK, which is developed through two-step embeddings: the node embedding and the Hilbert space embedding of graphs. In the node embedding step, we proposed the node feature vector, RPF, which characterizes the "structural role" of nodes. The sth element of RPF is just the $s-$step return probabilities of random walks defined on the graph. We analytically proved that RPF is not only isomorphism-invariant but also informative about the topological structure of the graph. In the Hilbert space embedding step, we represented the graph in a reproducing kernel Hilbert space. To do this, we constructed the empirical distribution from the RPF vector set and employed the kernel mean embedding to represent the

distribution in RKHS. To reduce the time complexity, we used the random Fourier features to approximate the shift-invariant kernel, and as a result, we can represent the graph with a multi-dimenisonal tensor. We also conducted extensive graph classification experiments and made comprehensive comparisons to demonstrate the effectiveness of RetGK.

We next proposed an Euclidean vector embedding algorithm for graphs with both node and edge attributes. The motivations are that RetGK requires to compute the all pairs of kernel values between graphs, which is not scalable to the dataset with large number of graphs. On the other hand, RetGK can consider only the node attributes while ignoring the edge attributes. To tackle the above issues, we extended the work of "RetGK" and developed SAGE, a scalable attributed graph embedding algorithm. We employed the "D2KE" framework, which can naturally generate the scalable embedding of graphs when integrating with the return probabilities features. In order to consider the edge attribute information, we took advantage of the adjoint graph, which can convert the node attributes to the edge attributes. Moreover, such a strategy opens a door to involving edge attributes information for all the related algorithms that are based on the node attributes.

Finally, we proposed a very efficient algorithm, KerGM, for attributed graph matching. We considered the Lawler's quadratic assignment problem (QAP). We showed that if the edge affinity can be characterized by a kernel function, then the Lawler's QAP can be written as the Koopmans-Beckmann's alignment between Hilbert arrays, which gives an unifying view for the Koopmans-Beckmann's QAP and the Lawer's QAP. We did it by properly defining the $\mathcal{H}-$operations in Hilbert spaces. The new form of the Lawler's QAP has two advantages. (i). It allows us to avoid computing the huge affinity matrix of the size $n^2 \times n^2$. (ii). The new form inspires a natural way to develop the convex and concave relaxations and the path-following strategy. We also used the random Fourier features to further reduce the time

complexity. As a result, both of the space and the time complexity of Lawler's QAP have the same order of magnitude as those of the Koopmans-Beckmann's QAP. Furthermore, we proposed, EnFW, an entropy-regularized Frank-Wolfe algorithm for solving QAP. We analytically proved that EnFW has the same convergence rate as the original FW algorithm while dramatically reducing the computational burden for each outer iteration. We tested our method on various types of dataset, including synthetic random graphs, images, and protein-protein interaction networks. On these datasets, KerGM achieved promising performance in both the matching accuracy and scalability.

6.2 Future Directions

In the future, we can potential extend our research to the following directions:

Dynamic graph-structured analysis Throughout the dissertation, we assume that the graphs are static. However, it is common in real world that the topological connectivity and the attributes of graphs are changing with respect to the time. For example, during biological experiments, the connectivity of brain networks of subjects keep changing, because human attentions are intrinsically dynamic, with focus continuously shifting [107]. An interesting problem is how to formulate and solve the graph comparison, graph learning, and graph matching problem in the dynamic case. We need to consider not only the attribute and topology information at a specific time, but also the temporal correlation among graphs across different times.

Deep graph matching In the applications of matching landmarks between images, both of the edge attributes and the node attributes are manually generated. These attributes maybe

not representative/complex enough to describe the characteristics of the landmarks and their interactions. To solve the concern, one interesting idea is that we can automatically extract the features of node and edges with a convolutional neural network(CNN) [108]. We may design structure of CNN based on [109], which can generate the hypercolumn descriptors for landmarks. One existing problem is how to define the loss function for training the neural network. A straightforward idea is to use the minimized Lawler's quadratic loss function. However, in that case, the optimization process for graph matching will be nested in the procedure for training the neural network. Therefore, developing a feasible training strategy becomes the key challenging part of deep graph matching.

Bibliography

[1] K. M. Borgwardt, "Graph kernels," Ph.D. dissertation, IMU, 2007.

[2] G. Nikolentzos, G. Siglidis, and M. Vazirgiannis, "Graph kernels: A survey," *arXiv preprint arXiv:1904.12218*, 2019.

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

[4] A. Calderone, M. Formenti, F. Aprea, M. Papa, L. Alberghina, A. M. Colangelo, and P. Bertolazzi, "Comparing alzheimer's and parkinson's diseases networks using graph communities structure," *BMC systems biology*, vol. 10, no. 1, p. 25, 2016.

[5] J. K. Morrow, L. Tian, and S. Zhang, "Molecular networks in drug discovery," *Critical Reviews^{TM} in Biomedical Engineering*, vol. 38, no. 2, 2010.

[6] L. Carpi, P. Saco, O. Rosso, and M. G. Ravetti, "Structural evolution of the tropical pacific climate network," *The European Physical Journal B*, vol. 85, no. 11, p. 389, 2012.

[7] L. d. F. Costa, F. A. Rodrigues, G. Travieso, and P. R. Villas Boas, "Characterization of complex networks: A survey of measurements," *Advances in physics*, vol. 56, no. 1, pp. 167–242, 2007.

[8] T. A. Schieber, L. Carpi, A. Díaz-Guilera, P. M. Pardalos, C. Masoller, and M. G. Ravetti, "Quantification of network structural dissimilarities," *Nature communications*, vol. 8, p. 13928, 2017.

[9] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *Journal of Machine Learning Research*, vol. 11, no. Apr, pp. 1201–1242, 2010.

[10] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*. Springer, 2003, pp. 63–71.

[11] B. Scholkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[12] B. Schölkopf, A. Smola, and K.-R. Müller, "Kernel principal component analysis," in *International conference on artificial neural networks*. Springer, 1997, pp. 583–588.

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

[14] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *Advances in neural information processing systems*, 2002, pp. 585–591.

[15] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[16] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning distributed representations of graphs," *arXiv preprint arXiv:1707.05005*, 2017.

[17] L. Wu, Z. Zhang, A. Nehorai, L. Zhao, and F. Xu, "Sage: Scalable attributed graph embed-dings for graph classification," *The International Conference on Learning Representations Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[18] S. Bhagat, G. Cormode, and S. Muthukrishnan, "Node classification in social networks," in *Social network data analytics*. Springer, 2011, pp. 115–148.

[19] C. H. Ding, X. He, H. Zha, M. Gu, and H. D. Simon, "A min-max cut algorithm for graph partitioning and data clustering," in *Proceedings 2001 IEEE International Conference on Data Mining*. IEEE, 2001, pp. 107–114.

[20] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[21] R. S. Lee and J. N. Liu, "An oscillatory elastic graph matching model for recognition of offline handwritten chinese characters," in *1999 Third International Conference on Knowledge-Based Intelligent Information Engineering Systems. Proceedings (Cat. No. 99TH8410)*. IEEE, 1999, pp. 284–287.

[22] B. Banerjee, F. Bovolo, A. Bhattacharya, L. Bruzzone, S. Chaudhuri, and K. M. Buddhiraju, "A novel graph-matching-based approach for domain adaptation in classification of remote sensing image pair," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 7, pp. 4045–4062, 2015.

[23] M. Zaslavskiy, F. Bach, and J.-P. Vert, "Global alignment of protein–protein interaction networks by graph matching methods," *Bioinformatics*, vol. 25, no. 12, pp. 1259–1267, 2009.

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

[25] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[26] T. C. Koopmans and M. Beckmann, "Assignment problems and the location of economic activities," *Econometrica: journal of the Econometric Society*, pp. 53–76, 1957.

[27] E. L. Lawler, "The quadratic assignment problem," *Management science*, vol. 9, no. 4, pp. 586–599, 1963.

[28] Z. Zhang, M. Wang, Y. Xiang, Y. Huang, and A. Nehorai, "Retgk: Graph kernels based on return probabilities of random walks," in *Advances in Neural Information Processing Systems*, 2018, pp. 3964–3974.

[29] Z. Zhang, Y. Xiang, L. Wu, B. Xue, and A. Nehorai, "Kergm: Kernelized graph matching," in *Advances in Neural Information Processing Systems*, 2019, pp. 3330–3341.

[30] N. Aronszajn, "Theory of reproducing kernels," *Transactions of the American mathematical society*, vol. 68, no. 3, pp. 337–404, 1950.

[31] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge university press, 2012.

[32] A. Gretton, "Introduction to rkhs, and some simple kernel algorithms."

[33] I. J. Schoenberg, "Metric spaces and positive definite functions," *Transactions of the American Mathematical Society*, vol. 44, no. 3, pp. 522–536, 1938.

[34] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Advances in neural information processing systems*, 2008, pp. 1177–1184.

[35] W. Rudin, *Fourier analysis on groups*. Wiley Online Library, 1962, vol. 121967.

[36] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics New York, 2001, vol. 1, no. 10.

[37] D. Haussler, "Convolution kernels on discrete structures," Technical report, Department of Computer Science, University of California, Tech. Rep., 1999.

[38] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *Artificial Intelligence and Statistics*, 2009, pp. 488–495.

[39] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.

[40] G. Da San Martino, N. Navarin, and A. Sperduti, "Tree-based kernel for graphs with continuous attributes," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 7, pp. 3270–3276, 2017.

[41] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *Fifth IEEE international conference on data mining (ICDM'05)*. IEEE, 2005, pp. 8–pp.

[42] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Learning theory and kernel machines*. Springer, 2003, pp. 129–143.

[43] N. Kriege and P. Mutzel, "Subgraph matching kernels for attributed graphs," *arXiv preprint arXiv:1206.6483*, 2012.

[44] F. Orsini, P. Frasconi, and L. De Raedt, "Graph invariant kernels," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[45] P. Yanardag and S. Vishwanathan, "A structural smoothing framework for robust graph comparison," in *Advances in neural information processing systems*, 2015, pp. 2134–2142.

[46] J. Kandola, T. Graepel, and J. Shawe-Taylor, "Reducing kernel matrix diagonal dominance using semi-definite programming," in *Learning Theory and Kernel Machines*. Springer, 2003, pp. 288–302.

[47] S. Verma and Z.-L. Zhang, "Hunt for the unique, stable, sparse and fast feature learning on graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 88–98.

[48] E. Cinlar, *Introduction to stochastic processes*. Courier Corporation, 2013.

[49] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[50] F. R. Chung and F. C. Graham, *Spectral graph theory*. American Mathematical Soc., 1997, no. 92.

[51] G. Nikolentzos, P. Meladianos, and M. Vazirgiannis, "Matching node embeddings for graph similarity," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[52] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *Journal of Machine Learning Research*, vol. 13, no. Mar, pp. 723–773, 2012.

[53] Z. Szabó and B. K. Sriperumbudur, "Characteristic and universal tensor product kernels," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 8724–8752, 2017.

[54] K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann, "Benchmark data sets for graph kernels," 2016, http://graphkernels.cs.tu-dortmund.de.

[55] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1365–1374.

[56] P. D. Dobson and A. J. Doig, "Distinguishing enzyme structures from non-enzymes without alignments," *Journal of molecular biology*, vol. 330, no. 4, pp. 771–783, 2003.

[57] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of medicinal chemistry*, vol. 34, no. 2, pp. 786–797, 1991.

[58] C. Helma, R. D. King, S. Kramer, and A. Srinivasan, "The predictive toxicology challenge 2000–2001," *Bioinformatics*, vol. 17, no. 1, pp. 107–108, 2001.

[59] J. Kazius, R. McGuire, and R. Bursi, "Derivation and validation of toxicophores for mutagenicity prediction," *Journal of medicinal chemistry*, vol. 48, no. 1, pp. 312–320, 2005.

[60] A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. Borgwardt, "Scalable kernels for graphs with continuous attributes," in *Advances in Neural Information Processing Systems*, 2013, pp. 216–224.

[61] C. Morris, N. M. Kriege, K. Kersting, and P. Mutzel, "Faster kernels for graphs with continuous attributes via hashing," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 1095–1100.

[62] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl_1, pp. 147–156, 2005.

[63] J. J. Sutherland, L. A. O'brien, and D. F. Weaver, "Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships," *Journal of chemical information and computer sciences*, vol. 43, no. 6, pp. 1906–1915, 2003.

[64] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *International conference on machine learning*, 2016, pp. 2014–2023.

[65] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[66] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[67] L. Wu, I. E.-H. Yen, F. Xu, P. Ravikumar, and M. Witbrock, "D2ke: From distance to kernel and embedding," *arXiv preprint arXiv:1802.04956*, 2018.

[68] G. J. Székely, "E-statistics: The energy of statistical samples," *Bowling Green State University, Department of Mathematics and Statistics Technical Report*, vol. 3, no. 05, pp. 1–18, 2003.

[69] C. Ionescu, A. Popa, and C. Sminchisescu, "Large-scale data-dependent kernel approximation," in *Artificial Intelligence and Statistics*, 2017, pp. 19–27.

[70] H. Whitney, "Congruent graphs and the connectivity of graphs," in *Hassler Whitney Collected Papers*. Springer, 1992, pp. 61–79.

[71] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler, "Skip-thought vectors," in *Advances in neural information processing systems*, 2015, pp. 3294–3302.

[72] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *Journal of machine learning research*, vol. 9, no. Aug, pp. 1871–1874, 2008.

[73] H. Almohamad and S. O. Duffuaa, "A linear programming approach for the weighted graph matching problem," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 15, no. 5, pp. 522–525, 1993.

[74] S. Gold and A. Rangarajan, "A graduated assignment algorithm for graph matching," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 18, no. 4, pp. 377–388, 1996.

[75] Y. Kushinsky, H. Maron, N. Dym, and Y. Lipman, "Sinkhorn algorithm for lifted assignment problems," *SIAM Journal on Imaging Sciences*, vol. 12, no. 2, pp. 716–735, 2019.

[76] C. Schellewald, S. Roth, and C. Schnörr, "Evaluation of convex optimization techniques for the weighted graph-matching problem in computer vision," in *Joint Pattern Recognition Symposium*. Springer, 2001, pp. 361–368.

[77] H. Maron and Y. Lipman, "(probably) concave graph matching," in *Advances in Neural Information Processing Systems*, 2018, pp. 408–418.

[78] J. Maciel and J. P. Costeira, "A global solution to sparse correspondence problems," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 2, pp. 187–199, 2003.

[79] A. Chinchuluun, E. Rentsen, and P. M. Pardalos, "A numerical method for concave programming problems," in *Continuous Optimization*. Springer, 2005, pp. 251–273.

[80] M. Zaslavskiy, F. Bach, and J.-P. Vert, "A path following algorithm for the graph matching problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 12, pp. 2227–2242, 2008.

[81] Z.-Y. Liu and H. Qiao, "Gnccp—graduated nonconvexityand concavity procedure," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 6, pp. 1258–1267, 2013.

[82] T. Wang, H. Ling, C. Lang, and S. Feng, "Graph matching with adaptive and branching path following," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2853–2867, 2017.

[83] T. Wang, H. Ling, C. Lang, and J. Wu, "Branching path following for graph matching," in *European Conference on Computer Vision*. Springer, 2016, pp. 508–523.

[84] T. Yu, J. Yan, Y. Wang, W. Liu *et al.*, "Generalizing graph matching beyond quadratic assignment model," in *Advances in Neural Information Processing Systems*, 2018, pp. 853–863.

[85] R. Zass and A. Shashua, "Probabilistic graph and hypergraph matching," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2008, pp. 1–8.

[86] J. Lee, M. Cho, and K. M. Lee, "Hyper-graph matching via reweighted random walks," in *2011 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2011, pp. 1633–1640.

[87] J. Yan, J. Wang, H. Zha, X. Yang, and S. Chu, "Consistency-driven alternating optimization for multigraph matching: A unified approach," *IEEE Transactions on Image Processing*, vol. 24, no. 3, pp. 994–1009, 2015.

[88] J. Yan, X.-C. Yin, W. Lin, C. Deng, H. Zha, and X. Yang, "A short survey of recent advances in graph matching," in *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*. ACM, 2016, pp. 167–174.

[89] Y. Aflalo, A. Bronstein, and R. Kimmel, "On convex relaxation of graph isomorphism," *Proceedings of the National Academy of Sciences*, vol. 112, no. 10, pp. 2942–2947, 2015.

[90] M. Leordeanu, M. Hebert, and R. Sukthankar, "An integer projected fixed point method for graph matching and map inference," in *Advances in neural information processing systems*, 2009, pp. 1114–1122.

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

[92] F. Zhou and F. De la Torre, "Factorized graph matching," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 9, pp. 1774–1789, 2015.

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

[94] M. Cuturi, "Sinkhorn distances: Lightspeed computation of optimal transport," in *Advances in neural information processing systems*, 2013, pp. 2292–2300.

[95] M. Cho, J. Lee, and K. M. Lee, "Reweighted random walks for graph matching," in *European conference on Computer vision*. Springer, 2010, pp. 492–505.

[96] F. Pedregosa, A. Askari, G. Negiar, and M. Jaggi, "Step-size adaptivity in projection-free optimization," *arXiv preprint arXiv:1806.05123*, 2018.

[97] M. Jaggi, "Revisiting frank-wolfe: Projection-free sparse convex optimization." in *2013 International Conference on Machine Learning*, 2013, pp. 427–435.

[98] T. Cour, P. Srinivasan, and J. Shi, "Balanced graph matching," in *Advances in Neural Information Processing Systems*, 2007, pp. 313–320.

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

[100] S. Mohammadi, D. F. Gleich, T. G. Kolda, and A. Grama, "Triangular alignment tame: A tensor-based approach for higher-order network alignment," *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 14, no. 6, pp. 1446–1458, 2017.

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

[102] B. Bollobás and O. M. Riordan, "Mathematical results on scale-free random graphs," *Handbook of graphs and networks: from the genome to the internet*, pp. 1–34, 2003.

[103] D.-T. Lee and B. J. Schachter, "Two algorithms for constructing a delaunay triangulation," *International Journal of Computer & Information Sciences*, vol. 9, no. 3, pp. 219–242, 1980.

[104] M. Leordeanu, R. Sukthankar, and M. Hebert, "Unsupervised learning for graph matching," *International journal of computer vision*, vol. 96, no. 1, pp. 28–45, 2012.

[105] S. R. Collins, P. Kemmeren, X.-C. Zhao, J. F. Greenblatt, F. Spencer, F. C. Holstege, J. S. Weissman, and N. J. Krogan, "Toward a comprehensive atlas of the physical interactome of saccharomyces cerevisiae," *Molecular & Cellular Proteomics*, vol. 6, no. 3, pp. 439–450, 2007.

[106] N. Hu, R. M. Rustamov, and L. Guibas, "Stable and informative spectral signatures for graph matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 2305–2312.

[107] A. Kucyi, M. J. Hove, M. Esterman, R. M. Hutchison, and E. M. Valera, "Dynamic brain network correlates of spontaneous fluctuations in attention," *Cerebral cortex*, vol. 27, no. 3, pp. 1831–1840, 2017.

[108] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[109] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Hypercolumns for object segmentation and fine-grained localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 447–456.

[110] L. Zadeh and C. Desoer, *Linear system theory: the state space approach*. Courier Dover Publications, 2008.

[111] L. L. Pennisi, "Coefficients of the characteristic polynomial," *Mathematics magazine*, vol. 60, no. 1, pp. 31–33, 1987.

[112] H. L. Royden and P. Fitzpatrick, *Real analysis*. Macmillan New York, 1988, vol. 32.

[113] R. Sinkhorn and P. Knopp, "Concerning nonnegative matrices and doubly stochastic matrices," *Pacific Journal of Mathematics*, vol. 21, no. 2, pp. 343–348, 1967.

Appendix A

Proof of the Informativeness of RPF

Before proceeding to the main proof, we first present some useful lemmas.

Lemma A.1. *(Cayley-Hamilton Theorem, see[111, 110]) Let* $\boldsymbol{A}$ *be an* $n \times n$ *matrix, and let* $P(\lambda) = \det(\lambda \boldsymbol{I}_n - \boldsymbol{A})$ *be the corresponding characteristic polynomial of* $\boldsymbol{A}$*, then* $P(\boldsymbol{A}) = \boldsymbol{0}$*, i.e.,*

$$\boldsymbol{A}^n + c_{n-1}\boldsymbol{A}^{n-1} + \cdots + c_1\boldsymbol{A} + (-1)^n \det(\boldsymbol{A})\boldsymbol{I}_n = \boldsymbol{0}, \tag{A.1}$$

where $c_{n-k} = \frac{(-1)^k}{k!} B_k\big(s_1, (-1)s_2, 2!s_3, \cdots, (-1)^{k-1}(k-1)!s_k\big)$*, and* B_k *is the Bell polynomial and* $s_i = \mathrm{trace}(\boldsymbol{A}^i)$*. In particular,* $\det(\boldsymbol{A}) = \frac{1}{n!} B_n\big(s_1, (-1)s_2, 2!s_3, \cdots, (-1)^{n-1}(n-1)!s_n\big)$.

Remark. We observe that all the coefficients in (A.1) are determined by $\mathrm{trace}(\boldsymbol{A})$, $\mathrm{trace}(\boldsymbol{A}^2)$,$\cdots$,$\mathrm{trace}(\boldsymbol{A}^n)$.

Corollary A.1. *Let* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *be two* $n \times n$ *matrices. If* $\mathrm{trace}(\boldsymbol{A}^k) = \mathrm{trace}(\boldsymbol{B}^k)$*,* $k = 1, 2, \cdots, n$*, then* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *have the same eigenvalues set.*

Proof. Let $P^A(\lambda)$ and $P^B(\lambda)$ be the characteristic polynomials of $\boldsymbol{A}$ and $\boldsymbol{B}$ respectively, and let c^A_{n-k} and c^B_{n-k}, $k = 1, 2, \cdots, n-1$ be the corresponding coefficients. Since $\mathrm{trace}(\boldsymbol{A}^k) = \mathrm{trace}(\boldsymbol{B}^k)$, $k = 1, 2, \cdots, n$, we have that $c^A_{n-k} = c^B_{n-k}$, $k = 1, 2, \cdots, n-1$, and $\det(\boldsymbol{A}) = \det(\boldsymbol{B})$.

Therefore, the roots of $P^A(\lambda)$ and $P^B(\lambda)$ are the same, which is equivalent to $\boldsymbol{A}$ and $\boldsymbol{B}$ having the same eigenvalues set. □

Corollary A.2. *Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be two $n \times n$ matrices. If $\boldsymbol{A}^s(i,i) = \boldsymbol{B}^s(i,i)$, $s = 1, 2, \cdots, n$, $i = 1, 2, \cdots, n$, then $\boldsymbol{A}^s(i,i) = \boldsymbol{B}^s(i,i)$, $s = n+1, n+2, \cdots$, $i = 1, 2, \cdots, n$.*

Proof. It is easy to obtain $\text{trace}(\boldsymbol{A}^s) = \text{trace}(\boldsymbol{B}^s)$, $s = 1, 2, \cdots, n$. Then based on the lemma A.1, the characteristic polynomials of $\boldsymbol{A}$ and $\boldsymbol{B}$ are same. Moreover,

$$\boldsymbol{A}^n = -c_{n-1}\boldsymbol{A}^{n-1} - c_{n-2}\boldsymbol{A}^{n-2} - \cdots - c_1\boldsymbol{A} - (-1)^n \text{det}(\boldsymbol{A})\boldsymbol{I}_n. \tag{A.2}$$

Multiply $\boldsymbol{A}^{s-n}$, $s \geq n+1$ on both sides, and we have

$$\boldsymbol{A}^s = -c_{n-1}\boldsymbol{A}^{s-1} - c_{n-2}\boldsymbol{A}^{s-2} - \cdots - c_1\boldsymbol{A}^{s-(n-1)} - (-1)^n \text{det}(\boldsymbol{A})\boldsymbol{A}^{s-n}. \tag{A.3}$$

Immediately, for any $i = 1, 2, \cdots, n$,

$$\boldsymbol{A}^s(i,i) = -c_{n-1}\boldsymbol{A}^{s-1}(i,i) - c_{n-2}\boldsymbol{A}^{s-2}(i,i) - \cdots - c_1\boldsymbol{A}^{s-(n-1)}(i,i) - (-1)^n \text{det}(\boldsymbol{A})\boldsymbol{A}^{s-n}(i,i). \tag{A.4}$$

From the iterative formula (A.4), we can see that $\boldsymbol{A}^s(i,i)$, $s = n+1, n+2, \cdots$ are uniquely determined by $\{\boldsymbol{A}(i,i), \boldsymbol{A}^2(i,i), \cdots, \boldsymbol{A}^n(i,i)\}$. Similarly, $\boldsymbol{B}^s(i,i)$, $s = n+1, n+2, \cdots$ are uniquely determined by $\{\boldsymbol{B}(i,i), \boldsymbol{B}^2(i,i), \cdots, \boldsymbol{B}^n(i,i)\}$. Combining with the fact $\boldsymbol{A}^s(i,i) = \boldsymbol{B}^s(i,i), s = 1, 2, .., n$, we obtain the desired result. □

Now we prove theorem 1.

Proof. **(1).** Let $\boldsymbol{\Pi}$ be the permutation matrix induced by τ, i.e., $\boldsymbol{\Pi}(i,j) = \delta_{j=\tau(i)}$. Then we have $\forall i,j = 1,2,\cdots,n$, $\boldsymbol{P}_H(\tau(i),\tau(j)) = (\boldsymbol{\Pi}\boldsymbol{P}_H\boldsymbol{\Pi}^T)(i,j)$. Since $\boldsymbol{P}_G^s(i,i) = \boldsymbol{P}_H^s(\tau(i),\tau(i)) = (\boldsymbol{\Pi}\boldsymbol{P}_H^s\boldsymbol{\Pi}^T)(i,i) = (\boldsymbol{\Pi}\boldsymbol{P}_H\boldsymbol{\Pi}^T)^s(i,i)$, $\forall v_i \in V_G$, and $\forall s = 1,2,\cdots,n$, by Corollary A.2, we have $\boldsymbol{P}_G^s(i,i) = (\boldsymbol{\Pi}\boldsymbol{P}_H\boldsymbol{\Pi}^T)^s(i,i) = \boldsymbol{P}_H^s(\tau(i),\tau(i))$, $\forall s = n+1, n+2\cdots+\infty$. Now, the first conclusion has been proved.

(2). The second one can be directly concluded from corollary A.1.

(3). Let $\boldsymbol{D}_G$ and $\boldsymbol{D}_H$ be the degree matrices of graph G and H, respectively. Then by Proposition 3.2

$$\frac{\boldsymbol{D}_G(i,i)}{\text{Vol}} = \lim_{s\to+\infty} \boldsymbol{P}_G^s(i,i) = \lim_{s\to+\infty} \boldsymbol{P}_H^s(\tau(i),\tau(i)) = \frac{\boldsymbol{D}_H(\tau(i),\tau(i))}{\text{Vol}}. \tag{A.5}$$

So

$$\boldsymbol{D}_G(i,i) = \boldsymbol{D}_H(\tau(i),\tau(i)), \forall v_i \in V_G. \tag{A.6}$$

Let $\boldsymbol{A}_G$ and $\boldsymbol{A}_H$ be the adjacent matrices of G and H respectively, and write $\boldsymbol{P}_G = \boldsymbol{D}_G^{-1}\boldsymbol{A}_G = \boldsymbol{D}_G^{-\frac{1}{2}}(\boldsymbol{D}_G^{-\frac{1}{2}}\boldsymbol{A}_G\boldsymbol{D}_G^{-\frac{1}{2}})\boldsymbol{D}_G^{\frac{1}{2}}$. Let $\boldsymbol{B}_G = \boldsymbol{D}_G^{-\frac{1}{2}}\boldsymbol{A}_G\boldsymbol{D}_G^{-\frac{1}{2}} \implies \boldsymbol{P}_G = \boldsymbol{D}_G^{-\frac{1}{2}}\boldsymbol{B}_G\boldsymbol{D}_G^{\frac{1}{2}} \implies \boldsymbol{P}_G^s = \boldsymbol{D}_G^{-\frac{1}{2}}\boldsymbol{B}_G^s\boldsymbol{D}_G^{\frac{1}{2}} \implies \boldsymbol{P}_G^s(i,i) = \boldsymbol{B}_G^s(i,i)$. $\boldsymbol{B}_G$ is a symmetric matrix, and has the same eigenvalues as $\boldsymbol{P}_G$. Write the orthonormal eigen-decomposition of $\boldsymbol{B}_G$ as $\boldsymbol{B}_G = \sum_{k=1}^n \lambda_k \vec{\boldsymbol{u}}_k \vec{\boldsymbol{u}}_k^T$, then

$$\boldsymbol{P}_G^s(i,i) = \boldsymbol{B}_G^s(i,i) = \sum_{k=1}^n \lambda_k^s \vec{\boldsymbol{u}}_k(i)^2, \tag{A.7}$$

where $\vec{\boldsymbol{u}}_k(i)$ denotes the ith component of the eigenvector $\vec{\boldsymbol{u}}_k$. Similarly, we have

$$\boldsymbol{P}_H^s(\tau(i),\tau(i)) = \boldsymbol{B}_H^s(\tau(i),\tau(i)) = \sum_{k=1}^n \mu_k^s \vec{\boldsymbol{v}}_k(\tau(i))^2 = \sum_{k=1}^n \lambda_k^s \vec{\boldsymbol{v}}_k(\tau(i))^2, \tag{A.8}$$

where $\vec{\boldsymbol{v}}_k(\tau(i))$ denotes the $\tau(i)$th component of $\vec{\boldsymbol{v}}_k$, and $\vec{\boldsymbol{v}}_k$ is the kth eigenvector of $\boldsymbol{B}_H = \boldsymbol{D}_H^{-\frac{1}{2}}\boldsymbol{A}_H\boldsymbol{D}_H^{-\frac{1}{2}}$. The last equality of (A.8) holds because $\{\lambda_1, \lambda_2, ..., \lambda_n\} = \{\mu_1, \mu_2, ..., \mu_n\}$.

Next, we use mathematical induction to show that $|\vec{\boldsymbol{u}}_k(i)| = |\vec{\boldsymbol{v}}_k(\tau(i))|$, $\forall v_i \in V_G$, $\forall k = 1, 2, \cdots, m$.

Step1: For $k = 1$, $\boldsymbol{P}_G\vec{\mathbf{1}} = \vec{\mathbf{1}} \Leftrightarrow \boldsymbol{B}_G\boldsymbol{D}_G^{\frac{1}{2}}\vec{\mathbf{1}} = \boldsymbol{D}_G^{\frac{1}{2}}\vec{\mathbf{1}} \Leftrightarrow \vec{\boldsymbol{u}}_1 = \pm\boldsymbol{D}_G^{\frac{1}{2}}\vec{\mathbf{1}}/\sqrt{\text{Vol}}$. Similarly, we have $\vec{\boldsymbol{v}}_1 = \pm\boldsymbol{D}_H^{\frac{1}{2}}\vec{\mathbf{1}}/\sqrt{\text{Vol}}$. Since $\forall v_i \in V_G$, $\boldsymbol{D}_G(i,i) = \boldsymbol{D}_H(\tau(i), \tau(i))$, we have $|\vec{\boldsymbol{u}}_1(i)| = |\vec{\boldsymbol{v}}_1(\tau(i))|$, $\forall v_i \in V_G$.

Step2: We show that if the first k eigenvectors satisfy, $|\vec{\boldsymbol{u}}_1(i)| = |\vec{\boldsymbol{v}}_1(\tau(i))|$, $|\vec{\boldsymbol{u}}_2(i)| = |\vec{\boldsymbol{v}}_2(\tau(i))|$,$\cdots$, $|\vec{\boldsymbol{u}}_k(i)| = |\vec{\boldsymbol{v}}_k(\tau(i))|$, then $|\vec{\boldsymbol{u}}_{k+1}(i)| = |\vec{\boldsymbol{v}}_{k+1}(\tau(i))|, \forall v_i \in V_G, \forall k = 1, 2, ..., m-2$.

We suppose that $\exists v_{i^*} \in V_G$, such that $|\vec{\boldsymbol{u}}_{k+1}(i^*)| \neq |\vec{\boldsymbol{v}}_{k+1}(\tau(i^*))|$. Without loss of generality, we assume that $\vec{\boldsymbol{u}}_{k+1}(i^*)^2 - \vec{\boldsymbol{v}}_{k+1}(\tau(i^*))^2 = \epsilon > 0$. Write

$$
\begin{aligned}
&\boldsymbol{P}_G^s(i^*, i^*) - \boldsymbol{P}_H^s(\tau(i^*), \tau(i^*)) \\
=&\sum_{l=1}^{n} \lambda_l^s \vec{\boldsymbol{u}}_l(i^*)^2 - \sum_{l=1}^{n} \lambda_l^s \vec{\boldsymbol{v}}_l(\tau(i^*))^2 \\
=&\sum_{l=1}^{m} \lambda_l^s \vec{\boldsymbol{u}}_l(i^*)^2 - \sum_{l=1}^{m} \lambda_l^s \vec{\boldsymbol{v}}_l(\tau(i^*))^2 \\
=&\sum_{l=k+1}^{m} \lambda_l^s \vec{\boldsymbol{u}}_l(i^*)^2 - \sum_{l=k+1}^{m} \lambda_l^s \vec{\boldsymbol{v}}_l(\tau(i^*))^2 \\
=&\lambda_{k+1}^s \big[\vec{\boldsymbol{u}}_{k+1}(i^*)^2 - \vec{\boldsymbol{v}}_{k+1}(\tau(i^*))^2\big] - \sum_{l=k+2}^{m} \lambda_l^s \big[\vec{\boldsymbol{u}}_l(i^*)^2 - \vec{\boldsymbol{v}}_l(\tau(i^*))^2\big] \\
=&\lambda_{k+1}^s \Big(\epsilon - \sum_{l=k+2}^{m} (\frac{\lambda_l}{\lambda_{k+1}})^s \big[\vec{\boldsymbol{u}}_l(i^*)^2 - \vec{\boldsymbol{v}}_l(\tau(i^*))^2\big]\Big),
\end{aligned}
\tag{A.9}
$$

where the second equality holds because $|\lambda_1| > |\lambda_2| > ... > |\lambda_m| > 0$, $|\lambda_{m+1}| = ... = |\lambda_n| = 0$. With the fact that $|\vec{\boldsymbol{u}}_l(i^*)^2 - \vec{\boldsymbol{v}}_l(\tau(i^*))^2| \leq 1$ $\Big(\text{since } 0 \leq \vec{\boldsymbol{u}}_l(i^*)^2, \vec{\boldsymbol{v}}_l(\tau(i^*))^2 \leq 1\Big)$, and $|\frac{\lambda_l}{\lambda_{k+1}}| <$

1, we have that there is a positive integer, M, such that,

$$\epsilon - \sum_{l=k+2}^{m} (\frac{\lambda_l}{\lambda_{k+1}})^{2M} \big[\vec{\boldsymbol{u}}_l(i^*)^2 - \vec{\boldsymbol{v}}_l(\tau(i^*))^2\big] > 0. \tag{A.10}$$

Therefore, $\boldsymbol{P}_G^{2M}(i^*, i^*) - \boldsymbol{P}_H^{2M}(\tau(i^*), \tau(i^*)) > 0$, contradicting the fact that

$$\boldsymbol{P}_G^s(i,i) = \boldsymbol{P}_H^s(\tau(i), \tau(i)), \forall v_i \in V_G, \forall s = 1, 2, ..., \infty. \tag{A.11}$$

So $|\vec{\boldsymbol{u}}_{k+1}(i)| = |\vec{\boldsymbol{v}}_{k+1}(\tau(i))|, \forall v_i \in V_G$.

Step 3: We show that if $|\vec{\boldsymbol{u}}_1(i)| = |\vec{\boldsymbol{v}}_1(\tau(i))|$, $|\vec{\boldsymbol{u}}_2(i)| = |\vec{\boldsymbol{v}}_2(\tau(i))|$,$\cdots$, $|\vec{\boldsymbol{u}}_{m-1}(i)| = |\vec{\boldsymbol{v}}_{m-1}(\tau(i))|$, then $|\vec{\boldsymbol{u}}_m(i)| = |\vec{\boldsymbol{v}}_m(\tau(i))|, \forall v_i \in V_G$. Since

$$0 = \boldsymbol{P}_G^s(i^*, i^*) - \boldsymbol{P}_H^s(\tau(i^*), \tau(i^*)) = \lambda_m^s \big[\vec{\boldsymbol{u}}_m(i^*)^2 - \vec{\boldsymbol{v}}_m(\tau(i^*))^2\big], \tag{A.12}$$

and $\lambda_m^s \neq 0$, we immediately have that $|\vec{\boldsymbol{u}}_m(i^*)| = |\vec{\boldsymbol{v}}_m(\tau(i^*))|$.

Combining all these three steps, we obtain the desired result $|\vec{\boldsymbol{u}}_k(i)| = |\vec{\boldsymbol{v}}_k(\tau(i))|$, $\forall v_i \in V_G$, $\forall k = 1, 2, \cdots, m$.

Since $\boldsymbol{P}_G = \boldsymbol{D}_G^{-\frac{1}{2}} \boldsymbol{B}_G \boldsymbol{D}_G^{\frac{1}{2}}$, we have the fact that $(\lambda_k, \vec{\boldsymbol{u}}_k)$ is an eigenpair of $\boldsymbol{B}_G$ if and only if $(\lambda_k, \boldsymbol{D}_G^{-\frac{1}{2}} \vec{\boldsymbol{u}}_k)$ is an eigenpair of $\boldsymbol{P}_G$. The above implies that $\vec{\boldsymbol{\psi}}_k = \boldsymbol{D}_G^{-1} \vec{\boldsymbol{u}}_k$, and similarly $\vec{\boldsymbol{\varphi}}_k = \boldsymbol{D}_H^{-1} \vec{\boldsymbol{v}}_k$. Now, $\forall v_i \in V_G$, $\forall k = 1, 2, ..., m$, we have

$$|\vec{\boldsymbol{\varphi}}_k(\tau(i))| = \boldsymbol{D}_H^{-1}(\tau(i)) |\vec{\boldsymbol{v}}_k(\tau(i))| = \boldsymbol{D}_G^{-1}(i) |\vec{\boldsymbol{v}}_k(\tau(i))| = \boldsymbol{D}_G^{-1}(i) |\vec{\boldsymbol{u}}_k(i)| = |\vec{\boldsymbol{\psi}}_k(i)|. \tag{A.13}$$

□

Appendix B

The Description of Datasets for Graph Kernels

The statistics of the benchmark graph datasets used in the paper are reported in Table B.1. Next, we describe in these datasets in detail.

Remark. "DA" is short for "discrete attributes" and "CA" is short for "continuous attributes".

B.1 Non-attributed (Unlabeled) Graphs

COLLAB [55] is a scientific collaboration dataset that consists of the ego-networks of 5,000 researchers from three scientific fields: *High Energy Physics*, *Condensed Matter Physics*, and *Astro Physics*. The task is to determine the field of each researcher based on their ego-networks.

IMDB-BINARY [55] is a movie collaboration dataset that consists of the ego-networks of 1,000 actors/actresses who played roles in movies in IMDB. In each graph, nodes represent

Table B.1: Statistics of the benchmark graph datasets

Datasets	graph #	class #	average node #	average edge #	DA	CA (Dim)
COLLAB	5000	3	74.49	2457.78	×	×
IMDB-BINARY	1000	2	19.77	96.53	×	×
IMDB-MULTI	1500	3	13.00	65.94	×	×
REDDIT-BINARY	2000	2	429.63	497.75	×	×
REDDIT-MULTI(5K)	4999	5	508.52	594.87	×	×
REDDIT-MULTI(12K)	11929	11	391.41	456.89	×	×
MUTAG	188	2	17.93	19.79	√	×
DD	1178	2	284.32	715.66	√	×
NCI1	4110	2	29.87	32.30	√	×
PTC-FM	349	2	14.11	14.48	√	×
PTC-FR	351	2	14.56	15.00	√	×
PTC-MM	336	2	13.97	14.32	√	×
PTC-MR	344	2	14.29	14.69	√	×
FRANK	4337	2	16.90	17.88	×	√ (780)
SYNTHETIC	300	2	100	196.25	×	√ (1)
Synthie	400	4	95.00	172.93	×	√ (15)
ENZYMES	600	6	32.63	64.14	√	√ (18)
PROTEINS	1113	2	39.06	72.82	√	√ (1)
BZR	405	2	35.75	38.36	√	√ (3)
COX2	467	2	41.22	43.45	√	√ (3)
DHFR	467	2	42.43	44.54	√	√ (3)

actors/actress, and there is an edge between them if they appear in the same movie. These graphs are derived from the *Action* and *Romance genres*.

IMDB-MULTI [55] is generated in a similar way to **IMDB-BINARY**. The difference is that it is derived from three genres: *Comedy*, *Romance*, and *Sci-Fi*.

REDDIT-BINARY [55] consists of graphs corresponding to online discussions on Reddit. In each graph, nodes represent users, and there is an edge between them if at least one of them respond to the other's comment. There are four popular subreddits, namely, *IAmA*, *AskReddit*, *TrollXChromosomes*, and *atheism*. *IAmA* and *AskReddit* are two question/answerbased *subreddits*, and *TrollXChromosomes* and *atheism* are two discussion-based subreddits. A graph is labeled according to whether it belongs to a question/answer-based community or a discussion-based community.

REDDIT-MULTI(5K) [55] is generated in a similar way to **REDDIT-BINARY**. The difference is that there are five subreddits involved, namely, *worldnews*, *videos*, *AdviceAnimals*, *aww*, and *mildlyinteresting*. Graphs are labeled with their corresponding subreddits.

REDDIT-MULTI(12K) [55] is generated in a similar way to **REDDIT-BINARY** and **REDDIT-MULTI(5K)**. The difference is that there are eleven subreddits involved, namely, *AskReddit*, *AdviceAnimals*, *atheism*, *aww*, *IAmA*, *mildlyinteresting*, *Showerthoughts*, *videos*, *todayilearned*, *worldnews*, and *TrollXChromosomes*. Still, graphs are labeled with their corresponding subreddits.

B.2 Graphs with Discrete Attributes

MUTAG [57] consists of graph representations of 188 mutagenic aromatic and heteroaromatic nitro chemical compounds. These graphs are labeled according to whether or not they have a mutagenic effect on the Gramnegative bacterium Salmonella typhimurium.

DD [56] consists of graph representations of 1,178 proteins. In each graph, nodes represent amino acids, and there is an edge if they are less than six Angstroms apart. Graphs are labeled according to whether they are enzymes or not.

NCI1 [39] consists of graph representations of 4,110 chemical compounds s screened for activity against non-small cell lung cancer and ovarian cancer cell lines, respectively.

PTC [58] consists of graph representations of chemical molecules. In each graph, nodes represent atoms, and edges represent chemical bonds. Graphs are labeled according to

carcinogenicity on rodents, divided into male mice (**MM**), male rats (**MR**), female mice (**FM**), and female rats (**FR**).

B.3 Graphs with Continuous Attributes

FRANK [59] is a chemical molecule dataset that consists of 2,401 mutagens and 1,936 nonmutagens. Originally, nodes are associated with chemical atom symbols. The most frequent atom symbols are mapped to MNIST digit images. By doing this, the original atom symbols can be recovered through the high dimensional MNIST vectors of pixel intensities, which are treated as the continuous attributes on graphs.

SYNTHETIC [60] consists of 300 random graphs. The continuous node attributes are sampled from the distribution $N(0, 1)$. There are two classes, A and B. Class A has 150 graphs, which are generated by randomly rewiring five edges and permuting ten node attributes. Class B has 150 graphs, which are generated by randomly rewiring ten edges and permuting five node attributes.

Synthie [61] consists of 400 random graphs, all of which are variants of two Erdos-Renyi graphs. The nodes are associated with 15-dimensional continuous attributes. All graphs are divided into four classes. The generation process of these graphs is described in [61].

B.4 Graphs with Discrete and Continuous Attributes

ENZYMES and **PROTEINS** [62] consist of graph representations of proteins. Nodes represent secondary structure elements (SSE), and there is an edge if they are neighbors

along the amino acid sequence or one of three neareset neighbors in space. The discrete attributes are SSE's types. The continuous attributes are the 3D length of the SSE. Graphs are labeled according to which EC top-level class they belong to.

BZR, **COX2**, and **DHFR** [63], [43] all are chemical compound datasets. Still, in each graph, nodes represent atoms, and edges represent chemical bonds. The discrete attributes correspond to atom types. The continuous attributes are 3D coordinates.

Appendix C

Gradient Computation for KerGM

C.1 Proving Proposition 5.2

Before proceeding to the main proof, we first present two useful lemmas.

Lemma C.1. $\langle \boldsymbol{\Psi}, \boldsymbol{\Xi} \odot \boldsymbol{X} \rangle_{\mathrm{F}_{\mathcal{H}}} = \langle \boldsymbol{\Xi}^T * \boldsymbol{\Psi}, \boldsymbol{X} \rangle_{\mathrm{F}}$, *and* $\langle \boldsymbol{\Psi}, \boldsymbol{X} \odot \boldsymbol{\Xi} \rangle_{\mathrm{F}_{\mathcal{H}}} = \langle \boldsymbol{\Psi} * \boldsymbol{\Xi}^T, \boldsymbol{X} \rangle_{\mathrm{F}}$.

Remark. In our paper, we only consider the kernel values. That is, the inner product values are real numbers. So $\langle \psi, \varphi \rangle_{\mathcal{H}} = \langle \varphi, \psi \rangle_{\mathcal{H}}$, $\forall \psi, \varphi \in \mathcal{H}$.

Proof. **(1).**

$$\begin{aligned}
\langle \boldsymbol{\Psi}, \boldsymbol{\Xi} \odot \boldsymbol{X} \rangle_{\mathrm{F}_{\mathcal{H}}} &= \sum_{i=1}^{n}\sum_{j=1}^{n} \langle \boldsymbol{\Psi}_{ij}, [\boldsymbol{\Xi} \odot \boldsymbol{X}]_{ij} \rangle_{\mathcal{H}} = \sum_{i=1}^{n}\sum_{j=1}^{n} \langle \boldsymbol{\Psi}_{ij}, \sum_{k=1}^{n} \boldsymbol{\Xi}_{ik} \boldsymbol{X}_{kj} \rangle_{\mathcal{H}} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} \langle \boldsymbol{\Psi}_{ij}, \boldsymbol{\Xi}_{ik} \boldsymbol{X}_{kj} \rangle_{\mathcal{H}} = \sum_{k=1}^{n}\sum_{j=1}^{n} \boldsymbol{X}_{kj} \sum_{i=1}^{n} \langle \boldsymbol{\Psi}_{ij}, \boldsymbol{\Xi}_{ik} \rangle_{\mathcal{H}} \\
&= \sum_{k=1}^{n}\sum_{j=1}^{n} [\boldsymbol{\Xi}^T * \boldsymbol{\Psi}]_{kj} \boldsymbol{X}_{kj} = \langle \boldsymbol{\Xi}^T * \boldsymbol{\Psi}, \boldsymbol{X} \rangle_{\mathrm{F}}. \qquad \text{(C.1)}
\end{aligned}$$

(2).

$$\begin{aligned}\langle \boldsymbol{\Psi}, \boldsymbol{X} \odot \boldsymbol{\Xi} \rangle_{\mathrm{F}_{\mathcal{H}}} &= \sum_{i=1}^{n}\sum_{j=1}^{n}\langle \boldsymbol{\Psi}_{ij}, [\boldsymbol{X} \odot \boldsymbol{\Xi}]_{ij} \rangle_{\mathcal{H}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\langle \boldsymbol{\Psi}_{ij}, \sum_{k=1}^{n} \boldsymbol{X}_{ik}\boldsymbol{\Xi}_{kj} \rangle_{\mathcal{H}} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\langle \boldsymbol{\Psi}_{ij}, \boldsymbol{X}_{ik}\boldsymbol{\Xi}_{kj} \rangle_{\mathcal{H}} = \sum_{i=1}^{n}\sum_{k=1}^{n} \boldsymbol{X}_{ik} \sum_{j=1}^{n}\langle \boldsymbol{\Psi}_{ij}, \boldsymbol{\Xi}_{kj} \rangle_{\mathcal{H}} \\ &= \sum_{i=1}^{n}\sum_{k=1}^{n}[\boldsymbol{\Psi} * \boldsymbol{\Xi}^T]_{ik}\boldsymbol{X}_{ik} = \langle \boldsymbol{\Psi} * \boldsymbol{\Xi}^T, \boldsymbol{X} \rangle_{\mathrm{F}}. \end{aligned} \tag{C.2}$$

□

Lemma C.2. $\boldsymbol{\Psi} * (\boldsymbol{\Xi} \odot \boldsymbol{X}) = (\boldsymbol{\Psi} * \boldsymbol{\Xi})\boldsymbol{X}$, $(\boldsymbol{X} \odot \boldsymbol{\Psi}) * \boldsymbol{\Xi} = \boldsymbol{X}(\boldsymbol{\Psi} * \boldsymbol{\Xi})$, *and* $\boldsymbol{\Psi} * (\boldsymbol{X} \odot \boldsymbol{\Xi}) = (\boldsymbol{\Psi} \odot \boldsymbol{X}) * \boldsymbol{\Xi}$.

Proof. The proof procedure is very similar with Corollary 5.1. □

Now we prove the equality (5.23)**.**

Proof. **(1).** We first rewrite the function $J_\alpha(\boldsymbol{X})$ as

$$J_\alpha(\boldsymbol{X}) = -\langle \boldsymbol{K}^N, \boldsymbol{X} \rangle_{\mathrm{F}} + (1-\alpha)J_1(\boldsymbol{X}) - \alpha J_2(\boldsymbol{X}),$$

where $J_1(\boldsymbol{X}) = \frac{1}{2}\|\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X} - \boldsymbol{X} \odot \boldsymbol{\Psi}^{(2)}\|^2_{\mathrm{F}_{\mathcal{H}}}$ and $J_2(\boldsymbol{X}) = \frac{1}{2}\|\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X} + \boldsymbol{X} \odot \boldsymbol{\Psi}^{(2)}\|^2_{\mathrm{F}_{\mathcal{H}}}$

We first compute the gradient of $J_1(\boldsymbol{X})$. We employ the following fact:

$$\forall \boldsymbol{E} \in \mathbb{R}^{n\times n},\ \langle \nabla J_1(\boldsymbol{X}), \boldsymbol{E} \rangle_{\mathrm{F}} = \lim_{t\to 0} \frac{J_1(\boldsymbol{X} + t\boldsymbol{E}) - J_1(\boldsymbol{X})}{t}. \tag{C.3}$$

$$
\begin{aligned}
&J_1(\boldsymbol{X}+t\boldsymbol{E}) - J_1(\boldsymbol{X}) \\
=&\frac{1}{2}\|\boldsymbol{\Psi}^{(1)} \odot (\boldsymbol{X}+t\boldsymbol{E}) - (\boldsymbol{X}+t\boldsymbol{E}) \odot \boldsymbol{\Psi}^{(2)}\|_{\mathrm{F}_{\mathcal{H}}}^2 - \frac{1}{2}\|\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X} - \boldsymbol{X} \odot \boldsymbol{\Psi}^2\|_{\mathrm{F}_{\mathcal{H}}}^{(2)} \\
=&\frac{1}{2}t^2\|\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{E} - \boldsymbol{E} \odot \boldsymbol{\Psi}^{(2)}\|_{\mathrm{F}_{\mathcal{H}}}^2 + t\langle \boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X} - \boldsymbol{X} \odot \boldsymbol{\Psi}^{(2)}, \boldsymbol{\Psi}^{(1)} \odot \boldsymbol{E} - \boldsymbol{E} \odot \boldsymbol{\Psi}^{(2)} \rangle_{\mathrm{F}_{\mathcal{H}}}
\end{aligned} \tag{C.4}
$$

Immediately, $\langle \nabla J_1(\boldsymbol{X}), \boldsymbol{E} \rangle_{\mathrm{F}} = \langle \boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X} - \boldsymbol{X} \odot \boldsymbol{\Psi}^{(2)}, \boldsymbol{\Psi}^{(1)} \odot \boldsymbol{E} - \boldsymbol{E} \odot \boldsymbol{\Psi}^{(2)} \rangle_{\mathrm{F}_{\mathcal{H}}}$.

We can rewrite the above formula as

$$
\begin{aligned}
&\langle \boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X} - \boldsymbol{X} \odot \boldsymbol{\Psi}^{(2)}, \boldsymbol{\Psi}^{(1)} \odot \boldsymbol{E} - \boldsymbol{E} \odot \boldsymbol{\Psi}^{(2)} \rangle_{\mathrm{F}_{\mathcal{H}}} \\
=&\langle \boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X} - \boldsymbol{X} \odot \boldsymbol{\Psi}^{(2)}, \boldsymbol{\Psi}^{(1)} \odot \boldsymbol{E} \rangle_{\mathrm{F}_{\mathcal{H}}} - \langle \boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X} - \boldsymbol{X} \odot \boldsymbol{\Psi}^{(2)}, \boldsymbol{E} \odot \boldsymbol{\Psi}^{(2)} \rangle_{\mathrm{F}_{\mathcal{H}}} \\
=&\langle \boldsymbol{\Psi}^{(1)} * (\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X}) - \boldsymbol{\Psi}^{(1)} * (\boldsymbol{X} \odot \boldsymbol{\Psi}^{(2)}), \boldsymbol{E} \rangle_{\mathrm{F}} - \langle (\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X}) * \boldsymbol{\Psi}^{(2)} - (\boldsymbol{X} \odot \boldsymbol{\Psi}^{(2)}) * \boldsymbol{\Psi}^{(2)}, \boldsymbol{E} \rangle_{\mathrm{F}_{\mathcal{H}}} \\
=&\langle (\boldsymbol{\Psi}^{(1)} * \boldsymbol{\Psi}^{(1)})\boldsymbol{X} - 2(\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X}) * \boldsymbol{\Psi}^{(2)} + \boldsymbol{X}(\boldsymbol{\Psi}^{(2)} * \boldsymbol{\Psi}^{(2)}), \boldsymbol{E} \rangle_{\mathrm{F}},
\end{aligned} \tag{C.5}
$$

where the 3rd equality holds because of Lemma C.1 and both $\boldsymbol{\Psi}^{(1)}$ and $\boldsymbol{\Psi}^{(2)}$ are symmetric, and the last equality holds because of Lemma C.2. Therefore,

$$
\nabla J_1(\boldsymbol{X}) = (\boldsymbol{\Psi}^{(1)} * \boldsymbol{\Psi}^{(1)})\boldsymbol{X} - 2(\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X}) * \boldsymbol{\Psi}^{(2)} + \boldsymbol{X}(\boldsymbol{\Psi}^{(2)} * \boldsymbol{\Psi}^{(2)}).
$$

Similarly, we have

$$
\nabla J_2(\boldsymbol{X}) = (\boldsymbol{\Psi}^{(1)} * \boldsymbol{\Psi}^{(1)})\boldsymbol{X} + 2(\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X}) * \boldsymbol{\Psi}^{(2)} + \boldsymbol{X}(\boldsymbol{\Psi}^{(2)} * \boldsymbol{\Psi}^{(2)}).
$$

Finally, substituting $\nabla J_1(\boldsymbol{X})$ and $\nabla J_2(\boldsymbol{X})$ into $\nabla J_\alpha(\boldsymbol{X}) = -\boldsymbol{K}^N + (1-\alpha)\nabla J_1(\boldsymbol{X}) - \alpha \nabla J_2(\boldsymbol{X})$, we obtain the result (5.23).

(2). For the first term $\mathbf{\Psi}^{(1)} * \mathbf{\Psi}^{(1)}$, we have

$$[\mathbf{\Psi}^{(1)} * \mathbf{\Psi}^{(1)}]_{ij} = \sum_{k=1}^{n} \langle \mathbf{\Psi}^{(1)}_{ik}, \mathbf{\Psi}^{(1)}_{kj} \rangle_{\mathcal{H}_\mathcal{K}} = \sum_{e^1_{ik}, e^1_{kj} \in \mathcal{E}_1} \langle \psi(\vec{\boldsymbol{q}}^1_{ik}), \psi(\vec{\boldsymbol{q}}^1_{kj}) \rangle_{\mathcal{H}_\mathcal{K}} = \sum_{e^1_{ik}, e^1_{kj} \in \mathcal{E}_1} k^E(\vec{\boldsymbol{q}}^1_{ik}, \vec{\boldsymbol{q}}^1_{kj}). \tag{C.6}$$

For the second term $\mathbf{\Psi}^{(2)} * \mathbf{\Psi}^{(2)}$, we have similar explanations.

For the third term $(\mathbf{\Psi}^{(1)} \odot \boldsymbol{X}) * \mathbf{\Psi}^{(2)}$, we have

$$\begin{aligned}
[(\mathbf{\Psi}^{(1)} \odot \boldsymbol{X}) * \mathbf{\Psi}^{(2)}]_{ia} &= \sum_{c=1}^{n} \langle [\mathbf{\Psi}^{(1)} \odot \boldsymbol{X}]_{ic}, \mathbf{\Psi}^{(2)}_{ca} \rangle_{\mathcal{H}_\mathcal{K}} = \sum_{c=1}^{n} \langle \sum_{k=1}^{n} \mathbf{\Psi}^{(1)}_{ik} \boldsymbol{X}_{kc}, \mathbf{\Psi}^{(2)}_{ca} \rangle_{\mathcal{H}_\mathcal{K}} \\
&= \sum_{c=1}^{n} \sum_{k=1}^{n} \langle \mathbf{\Psi}^{(1)}_{ik} \boldsymbol{X}_{kc}, \mathbf{\Psi}^{(2)}_{ca} \rangle_{\mathcal{H}_\mathcal{K}} = \sum_{e^1_{ik} \in \mathcal{E}_1, e^2_{ca} \in \mathcal{E}_2} \boldsymbol{X}_{kc} \langle \psi(\vec{\boldsymbol{q}}^1_{ik}), \psi(\vec{\boldsymbol{q}}^2_{ca}) \rangle_{\mathcal{H}_\mathcal{K}} \\
&= \sum_{e^1_{ik} \in \mathcal{E}_1, e^2_{ca} \in \mathcal{E}_2} \boldsymbol{X}_{kc} k^E(\vec{\boldsymbol{q}}^1_{ik}, \vec{\boldsymbol{q}}^2_{ca}).
\end{aligned} \tag{C.7}$$

□

C.2 Proving Proposition 5.3

Proof. **(1).** We first prove the equality (5.32). It suffices to show that the (i, a) term in the left part equals to the (i, a) term in the right part.

Let $e^1_{ik_1}, e^1_{ik_2}, ..., e^1_{ik_{D^i}}$ be the edges incident to the node v_i in graph $\mathcal{G}_1$, where D^i is the degree of the node v_i. Without loss of generality, we assume the assigned directions of these edges are, for some $1 \leq s \leq D^i$,

$$v_i \to v_{k_1}, v_i \to v_{k_2}, ..., v_i \to v_{k_s} \quad \text{and} \quad v_{k_{s+1}} \to v_i, v_{k_{s+2}} \to v_i, ..., v_{k_{D^i}} \to v_i. \tag{C.8}$$

Considering the ith row, $\boldsymbol{G}_1(v_i,:)$ and $\boldsymbol{H}_1(v_i,:)$, of the matrix $\boldsymbol{G}_1$ and $\boldsymbol{H}_1$, we have

$$\boldsymbol{G}_1(v_i, e^1_{ik_1}) = \boldsymbol{G}_1(v_i, e^1_{ik_2}) = ... = \boldsymbol{G}_1(v_i, e^1_{ik_s}) = 1, \tag{C.9}$$

$$\text{and} \quad \boldsymbol{H}_1(v_i, e^1_{ik_{s+1}}) = \boldsymbol{H}_1(v_i, e^1_{ik_{s+2}}) = ... = \boldsymbol{H}_1(v_i, e^1_{ik_{D^i}}) = 1. \tag{C.10}$$

Any other item in the ith row of the matrix $\boldsymbol{G}_1$ and $\boldsymbol{H}_1$ is zero.

Let $e^2_{C_1a}, e^2_{C_2a}, ..., e^2_{C_{D^a}a}$ be the edges incident to the node v_a in graph $\mathcal{G}_2$, where D^a is the degree of the node v_a. Without loss of generality, we assume the assigned directions of these edges are, for some $1 \leq t \leq D^a$,

$$v_{C_1} \to v_a, v_{C_2} \to v_a, ..., v_{C_t} \to v_a \quad \text{and} \quad v_a \to v_{C_{t+1}}, v_a \to v_{C_{t+2}}, ..., v_a \to v_{C_{D^a}}. \tag{C.11}$$

Considering the ath row, $\boldsymbol{G}_2(v_a,:)$ and $\boldsymbol{H}_2(v_a,:)$, of the matrix $\boldsymbol{G}_2$ and $\boldsymbol{H}_2$, we have

$$\boldsymbol{G}_2(v_a, e^2_{C_{t+1}a}) = \boldsymbol{G}_2(v_a, e^2_{C_{t+2}a}) = ... = \boldsymbol{G}_2(v_a, e^2_{C_{D^a}a}) = 1, \tag{C.12}$$

$$\text{and} \quad \boldsymbol{H}_2(v_a, e^2_{C_1a}) = \boldsymbol{H}_2(v_a, e^2_{C_2a}) = ... = \boldsymbol{H}_2(v_a, e^2_{C_ta}) = 1. \tag{C.13}$$

Any other item in the ath row of the matrix $\boldsymbol{G}_2$ and $\boldsymbol{H}_2$ is zero.

We first consider the first term $\boldsymbol{H}_1(\boldsymbol{G}_1^T\boldsymbol{X}\boldsymbol{G}_2 \circ \boldsymbol{K}_{12}^E)\boldsymbol{H}_2^T$ in $(\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X}) * \boldsymbol{\Psi}^{(2)}$ (see (5.32)),

$$
\begin{aligned}
&[\boldsymbol{H}_1(\boldsymbol{G}_1^T\boldsymbol{X}\boldsymbol{G}_2 \circ \boldsymbol{K}_{12}^E)\boldsymbol{H}_2^T]_{ia} && \text{(C.14)}\\
=&[\boldsymbol{H}_1(v_i,:)](\boldsymbol{G}_1^T\boldsymbol{X}\boldsymbol{G}_2 \circ \boldsymbol{K}_{12}^E)[\boldsymbol{H}_2(v_a,:)]^T && \text{(C.15)}\\
=&\sum_{\alpha=1}^{D^i}\sum_{\beta=1}^{D^a}\boldsymbol{H}_1(v_i,e^1_{ik_\alpha})\boldsymbol{H}_2(v_a,e^2_{C_\beta a})[\boldsymbol{G}_1^T\boldsymbol{X}\boldsymbol{G}_2 \circ \boldsymbol{K}_{12}^E](e^1_{ik_\alpha},e^2_{C_\beta a}) && \text{(C.16)}\\
=&\sum_{\alpha=s+1}^{D^i}\sum_{\beta=1}^{t}[\boldsymbol{G}_1^T\boldsymbol{X}\boldsymbol{G}_2 \circ \boldsymbol{K}_{12}^E](e^1_{ik_\alpha},e^2_{C_\beta a}) \quad \text{By the definition of}\boldsymbol{H}_1\text{(C.10)},\boldsymbol{H}_2\text{(C.13)} && \text{(C.17)}\\
=&\sum_{\alpha=s+1}^{D^i}\sum_{\beta=1}^{t}[\boldsymbol{G}_1^T\boldsymbol{X}\boldsymbol{G}_2](e^1_{ik_\alpha},e^2_{C_\beta a}) \times [\boldsymbol{K}_{12}^E](e^1_{ik_\alpha},e^2_{C_\beta a}) && \text{(C.18)}\\
=&\sum_{\alpha=s+1}^{D^i}\sum_{\beta=1}^{t}[\boldsymbol{K}_{12}^E](e^1_{ik_\alpha},e^2_{C_\beta a})\big[\boldsymbol{G}_1(:,e^1_{ik_\alpha})^T\boldsymbol{X}\boldsymbol{G}_2(:,e^2_{C_\beta a})\big] && \text{(C.19)}\\
=&\sum_{\alpha=s+1}^{D^i}\sum_{\beta=1}^{t}\boldsymbol{X}_{k_\alpha C_\beta}[\boldsymbol{K}_{12}^E](e^1_{ik_\alpha},e^2_{C_\beta a}), && \text{(C.20)}\\
=&\sum_{\alpha=1}^{s}\sum_{\beta=t+1}^{D^a}\boldsymbol{X}_{k_\alpha C_\beta}k^E(\vec{\boldsymbol{q}}^1_{ik_\alpha},\vec{\boldsymbol{q}}^2_{C_\beta a}). && \text{(C.21)}
\end{aligned}
$$

where $\boldsymbol{G}_1(:,e^1_{ik_\alpha})$ is a column of $\boldsymbol{G}_1$, corresponding to the edge $e^1_{ik_\alpha}$, and $\boldsymbol{G}_2(:,e^2_{C_\beta a})$ is a column of $\boldsymbol{G}_2$, corresponding to the edge $e^2_{C_\beta a}$. By the definition of $\boldsymbol{G}_1$ (C.9) and $\boldsymbol{G}_2$ (C.12), $\boldsymbol{G}_1(v_{k_\alpha},e^1_{ik_\alpha})=1$, since the direction of edge $e^1_{ik_\alpha}$ is assigned as $v_{k_\alpha}\rightarrow v_i$, for $\alpha = s+1, s+2, .., D^i$. We also have $\boldsymbol{G}_2(v_{C_\beta},e^2_{C_\beta a})=1$, since the direction of edge $e^2_{C_\beta a}$ is assigned as $v_{C_\beta}\rightarrow v_a$, for $\beta = 1,2,...,t$. All the other terms in $\boldsymbol{G}_1(:,e^1_{ik_\alpha})$ and $\boldsymbol{G}_2(:,e^2_{C_\beta a})$ are zero. The above discussion explains why the equality (C.20) holds.

Similarly, we can prove that

$$[\boldsymbol{H}_1(\boldsymbol{G}_1^T\boldsymbol{X}\boldsymbol{H}_2 \circ \boldsymbol{K}_{12}^E)\boldsymbol{G}_2^T]_{ia} = \sum_{\alpha=s+1}^{D^i}\sum_{\beta=t+1}^{D^a} \boldsymbol{X}_{k_\alpha C_\beta} k^E(\vec{\boldsymbol{q}}_{ik_\alpha}^1, \vec{\boldsymbol{q}}_{C_\beta a}^2), \tag{C.22}$$

$$[\boldsymbol{G}_1(\boldsymbol{H}_1^T\boldsymbol{X}\boldsymbol{G}_2 \circ \boldsymbol{K}_{12}^E)\boldsymbol{H}_2^T]_{ia} = \sum_{\alpha=1}^{s}\sum_{\beta=1}^{t} \boldsymbol{X}_{k_\alpha C_\beta} k^E(\vec{\boldsymbol{q}}_{ik_\alpha}^1, \vec{\boldsymbol{q}}_{C_\beta a}^2), \tag{C.23}$$

$$[\boldsymbol{G}_1(\boldsymbol{H}_1^T\boldsymbol{X}\boldsymbol{H}_2 \circ \boldsymbol{K}_{12}^E)\boldsymbol{G}_2^T]_{ia} = \sum_{\alpha=1}^{s}\sum_{\beta=t+1}^{D^a} \boldsymbol{X}_{k_\alpha C_\beta} k^E(\vec{\boldsymbol{q}}_{ik_\alpha}^1, \vec{\boldsymbol{q}}_{C_\beta a}^2). \tag{C.24}$$

Adding (C.14), (C.22), (C.23), and (C.24), we can obtain

$$\begin{aligned}&\big[\boldsymbol{H}_1(\boldsymbol{G}_1^T\boldsymbol{X}\boldsymbol{G}_2 \circ \boldsymbol{K}_{12}^E)\boldsymbol{H}_2^T + \boldsymbol{H}_1(\boldsymbol{G}_1^T\boldsymbol{X}\boldsymbol{H}_2 \circ \boldsymbol{K}_{12}^E)\boldsymbol{G}_2^T + \boldsymbol{G}_1(\boldsymbol{H}_1^T\boldsymbol{X}\boldsymbol{G}_2 \circ \boldsymbol{K}_{12}^E)\boldsymbol{H}_2^T + \\ &\boldsymbol{G}_1(\boldsymbol{H}_1^T\boldsymbol{X}\boldsymbol{H}_2 \circ \boldsymbol{K}_{12}^E)\boldsymbol{G}_2^T\big]_{ia} = \sum_{e_{ik}^1\in\mathcal{E}_1, e_{ca}^2\in\mathcal{E}_2} \boldsymbol{X}_{kc} k^E(\vec{\boldsymbol{q}}_{ik}^1, \vec{\boldsymbol{q}}_{ca}^2) = [(\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{X}) * \boldsymbol{\Psi}^{(2)}]_{ia},\end{aligned} \tag{C.25}$$

where the equality holds because of the fact that $e_{ik_\alpha}^1 \in \mathcal{E}_1$, $\alpha = 1, 2, ..., D^i$, and $e_{C_\beta a}^2 \in \mathcal{E}_2$, $\beta = 1, 2, ..., D^a$.

We finish the proof of the equality (5.32)!

(2). For proving the equality (5.30) and (5.31), we can write

$$\boldsymbol{\Psi}^{(1)} * \boldsymbol{\Psi}^{(1)} = (\boldsymbol{\Psi}^{(1)} \odot \boldsymbol{I}) * \boldsymbol{\Psi}^{(1)} \quad \text{and} \quad \boldsymbol{\Psi}^{(2)} * \boldsymbol{\Psi}^{(2)} = (\boldsymbol{\Psi}^{(2)} \odot \boldsymbol{I}) * \boldsymbol{\Psi}^{(2)}, \tag{C.26}$$

and directly apply the proved equality (5.32). □

Appendix D

Proof of the Convergence Rate of EnFW

Before proceeding to the main proofs, we introduce a lemma.

Lemma D.1. *The generated objective function value sequence,* $\{F_\alpha(\boldsymbol{X}_t)\}_{t=0}$*, satisfies*

$$F_\alpha(\boldsymbol{X}_{t+1}) \leq F_\alpha(\boldsymbol{X}_t) - (G_t - \frac{L}{n}s)s,\ \forall s \in [0,1]. \tag{D.1}$$

Proof. Since $H(\boldsymbol{X})$ is convex, we have

$$\forall s \in [0,1], \lambda H\big(\boldsymbol{X}_t + s(\boldsymbol{Y}_t - \boldsymbol{X}_t)\big) \leq \lambda H(\boldsymbol{X}_t) + s\big(\lambda H(\boldsymbol{Y}_t) - \lambda H(\boldsymbol{X}_t)\big). \tag{D.2}$$

Write the quadratic function $J_\alpha(\boldsymbol{X}_t + s(\boldsymbol{Y}_t - \boldsymbol{X}_t))$ as

$$J_\alpha(\boldsymbol{X}_t + s(\boldsymbol{Y}_t - \boldsymbol{X}_t)) = J_\alpha(\boldsymbol{X}_t) + s\langle \nabla J_\alpha(\boldsymbol{X}_t), \boldsymbol{Y}_t - \boldsymbol{X}_t\rangle_{\mathrm{F}} + Q(\boldsymbol{X}_t, \boldsymbol{Y}_t)s^2. \tag{D.3}$$

Adding (D.2) and (D.3), and reordering the resulting inequality, we have

$$F_\alpha(\boldsymbol{X}_t+s(\boldsymbol{Y}_t-\boldsymbol{X}_t)) \leq F_\alpha(\boldsymbol{X}_t)+s\big[\lambda H(\boldsymbol{Y}_t)-\lambda H(\boldsymbol{X}_t)+\langle\nabla J_\alpha(\boldsymbol{X}_t),\boldsymbol{Y}_t-\boldsymbol{X}_t\rangle_{\mathrm{F}}\big]+Q(\boldsymbol{X}_t,\boldsymbol{Y}_t)s^2. \tag{D.4}$$

Since $\lambda H(\boldsymbol{Y}_t) + \langle\nabla J_\alpha(\boldsymbol{X}_t),\boldsymbol{Y}_t\rangle_{\mathrm{F}} = \min_{\boldsymbol{Y}\in\mathcal{D}_n}\ \langle\nabla J_\alpha(\boldsymbol{X}_t),\boldsymbol{Y}\rangle_{\mathrm{F}} + \lambda H(\boldsymbol{Y})$ (See the 4th line in Algorithm 4), we have

$$G_t = g(\boldsymbol{X}_t) = -\big[\lambda H(\boldsymbol{Y}_t) - \lambda H(\boldsymbol{X}_t) + \langle\nabla J_\alpha(\boldsymbol{X}_t),\boldsymbol{Y}_t-\boldsymbol{X}_t\rangle_{\mathrm{F}}\big], \tag{D.5}$$

which is based on the definition of $g(\boldsymbol{X})$ (5.50) and G_t (See the 5th line in Algorithm 4). Substituting (D.5) into (D.4), we have

$$\forall s\in[0,1], F_\alpha(\boldsymbol{X}_t+s(\boldsymbol{Y}_t-\boldsymbol{X}_t)) \leq F_\alpha(\boldsymbol{X}_t)-(G_t-Q_ts)s. \tag{D.6}$$

Set $s = s_t$ (See the 6th and 7th line in Algorithm 4), we have

$$F_\alpha(\boldsymbol{X}_{t+1}) \leq F_\alpha(\boldsymbol{X}_t)-(G_t-Q_ts_t)s_t.$$

Now we consider the function

$$A_t(s) = (G_t-Q_ts)s, s\in[0,1].$$

We discuss the maximizer of $A_t(s)$ for $s\in[0,1]$. Considering that G_t is nonnegative (this is because of the definition of $g(\boldsymbol{X})$), we have

1. If $Q_t \leq 0$, then $A_t(s)$ achieve its maximum at $s=1$,
2. If $Q_t > 0$, then $A_t(s)$ achieve its maximum at $s=\min\{G_t/(2Q_t),1\}$.

Therefore, our stepsize s_t is just the maximizer of $A_t(s)$. That is

$$\begin{aligned} F_\alpha(\boldsymbol{X}_{t+1}) \leq F_\alpha(\boldsymbol{X}_{t+1}) \leq & F_\alpha(\boldsymbol{X}_t) - (G_t - Q_t s_t)s_t = F_\alpha(\boldsymbol{X}_t) - \max_{s\in[0,1]}(G_t - Q_t s)s \\ \leq & F_\alpha(\boldsymbol{X}_t) - (G_t - Q_t s)s \quad \forall s \in [0,1]. \end{aligned} \tag{D.7}$$

Since

$$Q_t = Q(\boldsymbol{X}_t, \boldsymbol{Y}_t) = \frac{1}{2}\mathrm{vec}(\boldsymbol{Y}_t - \boldsymbol{X}_t)^T \nabla^2 J_\alpha(\boldsymbol{X}_t)\mathrm{vec}(\boldsymbol{Y}_t - \boldsymbol{X}_t) \leq \frac{L}{2}\|\boldsymbol{X}_t - \boldsymbol{Y}_t\|_{\mathrm{F}}^2,$$

and

$$\|\boldsymbol{X} - \boldsymbol{Y}\|_{\mathrm{F}}^2 \leq \frac{2}{n}, \forall \boldsymbol{X}, \boldsymbol{Y} \in \mathcal{D}_n,$$

we have $Q_t \leq \frac{L}{n}$.

Combining (D.7) and the fact $Q_t \leq \frac{L}{n}$, we obtain the desired result. □

D.1 Proving Theorem 5.1

Proof. We consider the inequality (D.1) in Lemma D.1.

If $G_t > \frac{2L}{n}$, then $(G_t - \frac{L}{n}s)s$ is maximized at $s = 1$. So

$$F_\alpha(\boldsymbol{X}_{t+1}) \leq F_\alpha(\boldsymbol{X}_t) - (G_t - \frac{L}{n}) \leq F_\alpha(\boldsymbol{X}_t) - \frac{G_t}{2},$$

where the last equality holds because $\frac{L}{n} \leq \frac{G_t}{2}$.

If $G_t \leq \frac{2L}{n}$, then $(G_t - \frac{L}{n}s)s$ is maximized at $s = \frac{nG_t}{2L}$. So

$$F_\alpha(\boldsymbol{X}_{t+1}) \leq F_\alpha(\boldsymbol{X}_t) - (\frac{G_t}{2})\frac{nG_t}{2L}.$$

In summary,

$$F_\alpha(\boldsymbol{X}_{t+1}) \leq F_\alpha(\boldsymbol{X}_t) - \frac{G_t}{2}\min\{1, \frac{nG_t}{2L}\} \tag{D.8}$$

(I). Since $G_t \geq 0$ by definition (5.50), we have $F_\alpha(\boldsymbol{X}_{t+1}) \leq F_\alpha(\boldsymbol{X}_t)$, i.e., $\{F_\alpha(\boldsymbol{X}_t)\}_{t=0}$ is a decreasing sequence. $F_\alpha(\boldsymbol{X})$ is continuous on the compact region $\mathcal{D}_n$, which implies that $F_\alpha(\boldsymbol{X})$ is bounded below. So the sequence $\{F_\alpha(\boldsymbol{X}_t)\}_{t=0}$ will converge [112].

(II). Taking the sum of (D.8) over $t = 0, 1, 2, ..., T$, we obtain,

$$F_\alpha(\boldsymbol{X}_{T+1}) - F_\alpha(\boldsymbol{X}_0) \leq -\sum_{t=0}^{T}\frac{G_t}{2}\min\{1, \frac{G_t n}{2L}\}. \tag{D.9}$$

Let

$$G_T^* = \min_{0\leq t\leq T} G_t = \min_{0\leq t\leq T} G(\boldsymbol{X}_t).$$

Considering the additional fact that $-\triangle_0 \triangleq F_\alpha(\boldsymbol{X}^*) - F_\alpha(\boldsymbol{X}_0) \leq F_\alpha(\boldsymbol{X}_{T+1}) - F_\alpha(\boldsymbol{X}_0)$, we have

$$\triangle_0 \geq \sum_{t=0}^{T}\frac{G_t}{2}\min\{1, \frac{G_t n}{2L}\} \geq (T+1)\frac{G_T^*}{2}\min\{1, \frac{G_T^* n}{2L}\}.$$

(a). If $\frac{nG_T^*}{2L} \geq 1$, then $\triangle_0 \geq (T+1)\frac{G_T^*}{2} \iff G_T^* \leq \frac{2\triangle_0}{T+1} \leq \frac{2\triangle_0}{\sqrt{T+1}}$;

(b). If $\frac{nG_T^*}{2L} < 1$, then $\triangle_0 \geq (T+1)(\frac{G_T^*}{2})\frac{nG_T^*}{2L} \iff G_T^* \leq \sqrt{\frac{4L\triangle_0}{n(T+1)}}$.

In summary, we have $G_T^* \leq \frac{2\max\{\triangle_0, \sqrt{L\triangle_0/n}\}}{\sqrt{T+1}}$. □

D.2 Proving Theorem 5.2

Before proceeding to the main proof, we introduce another lemma.

Lemma D.2. *If $\boldsymbol{J}_\alpha(\boldsymbol{X})$ is convex, and let $\boldsymbol{X}^*$ is a global minimizer of problem* (5.45)*, then* $g(\boldsymbol{X}_t) \geq F_\alpha(\boldsymbol{X}_t) - F_\alpha(\boldsymbol{X}^*)$.

Proof.

$$
\begin{aligned}
g(\boldsymbol{X}_t) &= \langle \nabla J_\alpha(\boldsymbol{X}_t), \boldsymbol{X}_t \rangle_{\mathrm{F}} + \lambda H(\boldsymbol{X}_t) - \min_{\boldsymbol{Y} \in \mathcal{D}_n} \langle \nabla J_\alpha(\boldsymbol{X}_t), \boldsymbol{Y} \rangle_{\mathrm{F}} + \lambda H(\boldsymbol{Y}). && \text{(D.10)} \\
&= \max_{\boldsymbol{Y} \in \mathcal{D}_n} \Big\{ \langle \nabla J_\alpha(\boldsymbol{X}_t), \boldsymbol{X}_t - \boldsymbol{Y} \rangle_{\mathrm{F}} + \lambda H(\boldsymbol{X}_t) - \lambda H(\boldsymbol{Y}) \Big\} && \text{(D.11)} \\
&\geq \langle \nabla J_\alpha(\boldsymbol{X}_t), \boldsymbol{X}_t - \boldsymbol{X}^* \rangle_{\mathrm{F}} + \lambda H(\boldsymbol{X}_t) - \lambda H(\boldsymbol{X}^*) && \text{(D.12)} \\
&\geq J_\alpha(\boldsymbol{X}_t) - J_\alpha(\boldsymbol{X}^*) + \lambda H(\boldsymbol{X}_t) - \lambda H(\boldsymbol{X}^*) && \text{(D.13)} \\
&= F_\alpha(\boldsymbol{X}_t) - F_\alpha(\boldsymbol{X}^*), && \text{(D.14)}
\end{aligned}
$$

where the inequality (D.13) holds because $J_\alpha(\boldsymbol{X})$ is convex. □

Now we prove Theorem 5.2.

Proof. We still need the inequality (D.1) in Lemma D.1.

$$
\begin{aligned}
F_\alpha(\boldsymbol{X}_{t+1}) &\leq F_\alpha(\boldsymbol{X}_t) - (G_t - \frac{L}{n}s)s \\
&\leq F_\alpha(\boldsymbol{X}_t) - \big[F_\alpha(\boldsymbol{X}_t) - F_\alpha(\boldsymbol{X}^*)\big]s + \frac{L}{n}s^2, \ \forall s \in [0,1], && \text{(D.15)}
\end{aligned}
$$

where the inequality (D.15) holds because of Lemma D.2. Reordering (D.15) yields

$$\big[F_\alpha(\boldsymbol{X}_{t+1}) - F_\alpha(\boldsymbol{X}^*)\big] \leq (1-s)\big[F_\alpha(\boldsymbol{X}_t) - F_\alpha(\boldsymbol{X}^*)\big] + \frac{L}{n}s^2. \tag{D.16}$$

We set $s = \frac{2}{t+1}$, and multiply $\frac{t(t+1)}{2}$ on both sides. Then we have

$$\begin{aligned}\frac{t(t+1)}{2}\big[F_\alpha(\boldsymbol{X}_{t+1}) - F_\alpha(\boldsymbol{X}^*)\big] &\leq \frac{t(t-1)}{2}\big[F_\alpha(\boldsymbol{X}_t) - F_\alpha(\boldsymbol{X}^*)\big] + \frac{L}{n}\frac{2t}{t+1} \\ &\leq \frac{t(t-1)}{2}\big[F_\alpha(\boldsymbol{X}_t) - F_\alpha(\boldsymbol{X}^*)\big] + \frac{2L}{n} \end{aligned} \tag{D.17}$$

Taking the sum of (D.17) over $t = 1, 2, ..., T$, we have

$$\frac{T(T+1)}{2}\big[F_\alpha(\boldsymbol{X}_{T+1}) - F_\alpha(\boldsymbol{X}^*)\big] \leq \frac{2TL}{n} \iff F_\alpha(\boldsymbol{X}_{T+1}) - F_\alpha(\boldsymbol{X}^*) \leq \frac{4L}{n(T+1)}. \tag{D.18}$$

We obtain the desired result. □

Appendix E

Implementation Details of EnFW

The Sinkhorn-Knopp algorithm [113, 94] is used to solve the problem

$$\min_{\boldsymbol{X}} \langle \boldsymbol{D}, \boldsymbol{X} \rangle_{\mathrm{F}} + \lambda H(\boldsymbol{X}) \quad \text{s.t.} \quad \boldsymbol{X} \geq 0, \quad \boldsymbol{X}\vec{\mathbf{1}}_n = \vec{\boldsymbol{a}}, \text{ and } \boldsymbol{X}^T\vec{\mathbf{1}}_n = \vec{\boldsymbol{b}}, \tag{E.1}$$

where $\vec{\boldsymbol{a}}$ and $\vec{\boldsymbol{b}}$ satisfy $\sum_{i=1}^{n} \vec{\boldsymbol{a}}_i = \sum_{j=1}^{n} \vec{\boldsymbol{b}}_j = 1$. In our case $\vec{\boldsymbol{a}} = \frac{1}{n}\vec{\mathbf{1}}_n$ and $\vec{\boldsymbol{b}} = \frac{1}{n}\vec{\mathbf{1}}_n$.

The algorithm is shown in Algorithm 5. The detailed derivations can be found in [94]. In

Algorithm 5 The Sinkhorn-Knopp optimization algorithm for minimizing (E.1)

1: Initialize $\vec{\boldsymbol{r}} = \frac{1}{n}\vec{\mathbf{1}}_n$, $t = 0$, and RelativeError $= 1$, and write $\boldsymbol{C} = \exp(-\frac{\boldsymbol{D}}{\lambda})$ (pointwise). $t <$ MaxNum$_{\text{Sinkhorn}}$ and RelativeError $>$ Tolerance
2: $\vec{\boldsymbol{r}}_{\text{new}} = (\frac{1}{n}\vec{\mathbf{1}}_n)./\left\{\boldsymbol{C}\left[(\frac{1}{n}\vec{\mathbf{1}}_n)./(\boldsymbol{C}^T\vec{\boldsymbol{r}})\right]\right\}$.
3: RelativeError $= \frac{\|\vec{\boldsymbol{r}}_{\text{new}} - \vec{\boldsymbol{r}}\|_2}{\|\vec{\boldsymbol{r}}\|_2}$.
4: $t = t + 1$
5: **end**
6: $\vec{\boldsymbol{s}} = (\frac{1}{n}\vec{\mathbf{1}}_n)./(\boldsymbol{C}^T\vec{\boldsymbol{r}})$, $\boldsymbol{X}^* = \mathrm{diag}(\vec{\boldsymbol{r}})\boldsymbol{C}\mathrm{diag}(\vec{\boldsymbol{r}})$.
7: Output the solution $\boldsymbol{X}^*$.

our experiments, we set MaxNum$_{\text{Sinkhorn}} = 10000$, and set Tolerance $= 10^{-6}$.

Vita

Zhen Zhang

Degrees

Ph.D., Electrical Engineering, Washington University in St. Louis, Missouri, USA, December 2019
M.S., Electrical Engineering, Washington University in St. Louis, Missouri, USA, May 2017
B.S., Electronic Engineering and Information Science, University of Science and Technology of China, Hefei, China, June 2014

Professional Memberships

The Institute of Electrical and Electronics Engineers (IEEE)
IEEE Signal Processing Society

Publications

Journal Publications:

Z. Zhang, M. Wang, and A. Nehorai, "Optimal transport in reproducing kernel Hilbert spaces: theory and applications," to appear in *IEEE Trans. on Pattern Analysis and Machine Intelligence*.

Y. Huang, G. Liao, **Z. Zhang**, Y. Xiang, J. Li, and A. Nehorai, "Reweighted Nuclear Norm and Reweighted Frobenius Norm Minimizations for Narrowband RFI Suppression on SAR System," *IEEE Trans. on Geoscience and Remote Sensing*, Vol. 57, No. 8, pp. 5949-5962, Aug. 2019.

M. Wang, **Z. Zhang**, and A. Nehorai, "Grid-less DOA estimation using sparse linear arrays based on Wasserstein distance," *IEEE Signal Processing Letters*, Vol. 26, No. 6, pp. 838-842, June 2019.

M. Wang, **Z. Zhang**, and A. Nehorai, "Further results on the Cramér Rao bound for sparse linear arrays" *IEEE Trans. on Signal Processing*, Vol. 67, No. 6, pp. 1493-1507, Mar. 2019.

M. Wang, **Z. Zhang**, and A. Nehorai, "Performance analysis of coarray-based MUSIC in the presence of sensor location errors," *IEEE Trans. on Signal Processing*, Vol. 66, pp. 3074-3085, June 2018.

Y. Huang, G. Liao, **Z. Zhang**, Y. Xiang, J. Li, and A. Nehorai, "Fast narrowband RFI suppression algorithms for SAR systems via matrix-factorization techniques," *IEEE Trans. on Geoscience and Remote Sensing*, Vol. 57, No. 1, pp. 250-262, Jan. 2019.

Y. Huang, G. Liao, **Z. Zhang**, Y. Xiang, J. Li, and A. Nehorai, "SAR automatic target recognition using joint low-rank and sparse multi-view denoising," *IEEE Geoscience and Remote Sensing Letters*, Vol. 15, No. 10, pp. 1570-1574, Oct. 2018.

Conference Publications:

Z. Zhang, Y. Xiang, L. Wu, B. Xue, and A. Nehorai, "KerGM: Kernelized graph matching," to appear in *Advances in Neural Information Processing Systems (**NeurIPS**)*, Vancouver, CA, Dec. 8-14, 2019.

L. Wu, I. Yen, **Z. Zhang**, K. Xu, L. Zhao, X. Peng, Y. Xia and C. Aggarwal, "Scalable global alignment graph kernel using random features: from node embedding to graph embedding," *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(**KDD**),* Anchorage, USA, Aug. 4-8, 2019.

L. Wu*, **Z. Zhang***, A. Nehorai, L. Zhao, and F. Xu, "SAGE: Scalable attributed graph embeddings for graph classification," *The International Conference on Learning Representations (**ICLR**) 2019 Workshop on Representation Learning on Graphs and Manifolds.* (* indicates equal contribution.)

Z. Zhang, M. Wang, Y. Xiang, Y. Huang, and A. Nehorai, "RetGK: Graph kernels based on return probabilities of random walks," *Advances in Neural Information Processing Systems (**NeurIPS**)*, Montréal, CA, Dec. 3-8, 2018.

Z. Zhang, M. Wang, Y. Huang, and A. Nehorai, "Aligning infinite-dimensional covariance matrices in reproducing kernel Hilbert spaces for domain adaptation," *IEEE Conference on Computer Vision and Pattern Recognition (**CVPR**)*, Salt Lake City, USA, June 22–28, 2018.

Z. Zhang, M. Wang, Y. Xiang, and A. Nehorai, "Geometry-adapted Gaussian random field regression," *Proc. 42nd IEEE Int. Conf. Acoustics, Speech, Signal Processing (**ICASSP**),* New Orleans, LA, Mar. 5-9, 2017.

M. Wang, **Z. Zhang**, and A. Nehorai, "Direction finding using sparse linear arrays with missing data," *Proc. 42nd IEEE Int. Conf. Acoustics, Speech, Signal Processing (**ICASSP**),* New Orleans, LA, Mar. 5-9, 2017.

M. Wang, **Z. Zhang**, and A. Nehorai, "Performance analysis of coarray-based MUSIC and the Cramér Rao bound," *Proc. 42nd IEEE Int. Conf. Acoustics, Speech, Signal Processing (**ICASSP**),* New Orleans, LA, Mar. 5-9, 2017.

December 2019

Kernels for Analyzing Graphs, Zhang, Ph.D. 2019