Signal Processing Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
https://sps.ewi.tudelft.nl/

SPS-2023-5442761

M.Sc. Thesis

Subgraph Matching via Fused Gromov-Wasserstein Distance

Wenxin Pan

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Subgraph Matching via Fused Gromov-Wasserstein Distance

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Electrical Engineering

by

Wenxin Pan
born in Chengdu, China

This work was performed in:

Signal Processing Systems Group
Department of Microelectronics
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Signal Processing Laboratory 4
Institute of Electrical and Micro Engineering
School of Engineering
École Polytechnique Fédérale de Lausanne

Delft University of Technology

Delft University of Technology
Department of
Microelectronics

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Subgraph Matching via Fused Gromov-Wasserstein Distance"** by **Wenxin Pan** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: November 17th, 2023

Chairman:

prof.dr.ir. G. Leus

Advisors:

prof.dr.ir. G. Leus

dr. I. Haasler

Committee Members:

prof.dr.ir. J.H. Weber

Abstract

Subgraph matching is a fundamental problem in various fields such as machine learning, computer vision, image processing, and bioinformatics, where detecting specific substructures within an object is often crucial. In these domains, not only structure plays an essential role, but also the feature information on nodes should be incorporated, thus highlighting the necessity for comprehensive analytical approaches.

In this thesis, we propose two novel subgraph matching frameworks using the Fused Gromov-Wasserstein (FGW) distance, namely the Subgraph Optimal Transport (SOT) and the Sliding Subgraph Optimal Transport (SSOT). Both frameworks integrate a dummy node strategy to handle the discrepancy between two graphs of different sizes. The SSOT extends upon the SOT by incorporating a sliding window framework and Wasserstein pruning to enhance the performance, especially for sparse large graphs. Our frameworks can be easily implemented and are adaptable for problems of exact matching, top-k approximate matching, and inexact matching.

We further propose a normalized FGW distance to cater to the practical interests and enhance the performance evaluation. We adopt the Frank-Wolfe algorithm for optimization and develop computation-reducing techniques by isolating the dummy node.

By conducting experiments on both synthetic and real-world datasets, we demonstrate that the SOT method achieves excellent performance on small graphs, and the SSOT method improves the accuracy over the SOT on large graphs. Both these two methods show the ability to outperform the state-of-the-art methods in noisy environments in terms of accuracy and efficiency.

Code availability: The implementation of this work is available at https://github.com/pandadada123/FGWD_on_Graphs_subgraph

Acknowledgments

First and foremost, I would like to express my sincerest gratitude to my daily supervisor Dr. Isabel Haasler, for her continuous guidance, patience, and forthright feedback. I also earnestly thank Prof. Pascal Frossard for accepting me in his lab and providing timely supervision. I also appreciate Prof. Geert Leus, for his ongoing support during the thesis period. I am also heartfully grateful to Prof. Jos Weber, for his encouragement, and for graciously agreeing to be a member of my committee.

Beyond this thesis, I am thankful for all the invaluable support and guidance I have received throughout my master's studies at both TU Delft and EPFL. My heartfelt thanks go to Dr. Olivier Lévêque, for his guidance through the semester project. I would also like to extend my thanks to the exchange coordinator Ms. Rianne Smits. Throughout this journey, I am inspired by various professors, researchers, and students that I have had the privilege to meet at both universities, whose passion for science and technology has continuously motivated me. Last but not least, I wish to thank all my friends and my family for their continuous support and companionship.

Wenxin Pan
Delft, The Netherlands
November 17th, 2023

Nomenclature

Mathematical objects

x	Scalar
$\boldsymbol{x}$	Column vector
$\boldsymbol{x}_i$	The ith entry of column vector $\boldsymbol{x}$
$\boldsymbol{X}$	Matrix or tensor
$\boldsymbol{X}_{i,j}$	Entry (i, j) of matrix $\boldsymbol{X}$
$\mathcal{X}$	Set or function
x_i or $\boldsymbol{x}^{(i)}$	The ith element of set $\mathcal{X}$
i, i', j, j'	Discrete value
x, x', y, y'	Discrete or continuous value

Linear algebra

$\mathbf{1}_n$	All-ones column vector in $\mathbb{R}^n$
$\mathbf{0}_n$	All-zeros column vector in $\mathbb{R}^n$
$(\cdot)^\top$	Transpose
$\mathrm{tr}(\cdot)$	Trace
$\langle \cdot, \cdot \rangle_F$	Frobenius inner product
$\otimes_K$	Kronecker product
$\otimes$	Tensor-matrix product (self-defined)
$\mathrm{vec}(\cdot)$	Column-stacking operator
$\| \cdot \|_p$	p-norm of a vector
$\| \cdot \|_F$	Frobenius-norm of a matrix

Probability

μ	Probability measure
$\mathrm{supp}[\mu]$	Support of probability measure μ
$\boldsymbol{p}, \boldsymbol{q}$	Probability vector
$\mathrm{unif}(n)$	Probability vector of uniform distribution of n entries

Contents

1 Introduction

The *graph* is a powerful model that helps us understand this complex world, no matter what kind of systems, purely natural or artificial. It comes from the real world and then pays back to the real world. Often it has an interchangeable name as *network*. It has various applications ranging from engineering, network science, machine learning, bioinformatics, image analysis, pattern recognition, computer vision, natural language processing, etc.

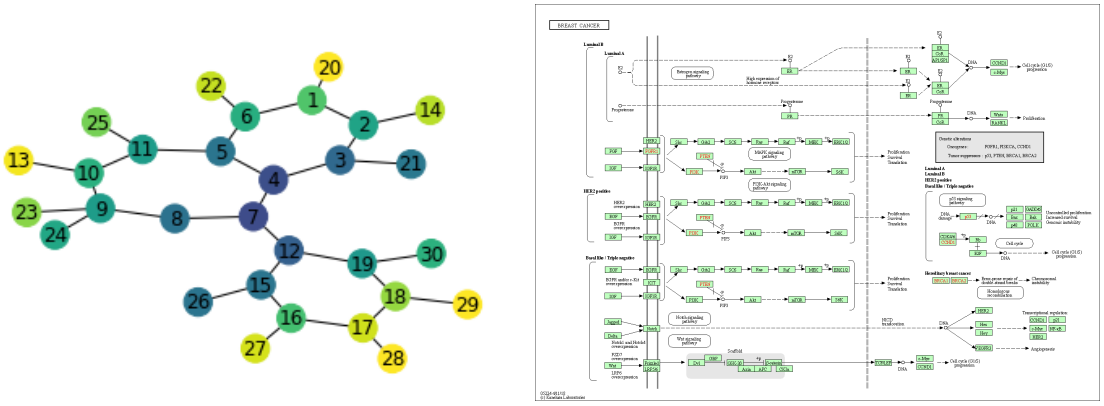
1.1 Motivation

In this thesis, we specifically focus on the *subgraph* matching problem. Subgraph matching is a fundamental problem with broad applications. In biology, it is necessary to detect a specific pattern to understand the biological function of an object. It is also a common practice to search for the same or similar patterns within the same category of objects, which may indicate common characteristics of these objects. In chemistry, detecting chemical patterns helps in formulating new chemical compounds, which will contribute to various areas such as drug discovery. In cybersecurity, subgraph matching is utilized to identify patterns of malicious behavior within a network. It is critical to recognize that in the mentioned fields, the essential information lies not only in the structural part but also largely in the attributed data associated with the nodes.

To be more specific, Figure 1.1a shows a chemical compound from the chemical BZR dataset [1], with colors indicating the node feature values. Chemists are particularly detecting "rings" within different compounds. In this compound, we can observe three distinct rings. We are interested in whether other compounds in the dataset also contain rings with similar features and structure, which suggests potential relationships among them. Figure 1.1b illustrates the biological pathway of breast cancer, showing the mechanism of this disease. When analyzing different types of cancer, it is crucial to identify if they possess common characteristics within their mechanisms. This knowledge could enable the use of similar therapeutic strategies across different cancer types. In addition, we can also make use of known signaling pathways that exhibit specific characteristics, to determine the type of disease pathway in question.

1.2 Problem statement

Given a *test graph*, our primary objective is to determine if it contains a *subgraph* that aligns exactly or closely with a predefined *query graph*. If such a subgraph exists, pinpointing its position within the test graph is of interest. We are also interested in identifying the top-k subgraphs that are similar to the query graph ranked by their similarity scores.



(a) Chemical compound (b) Biological pathway

Figure 1.1: Examples of graphs in applications

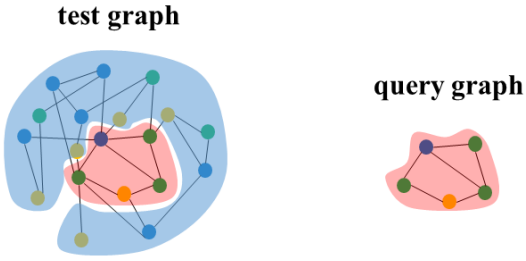


Figure 1.2: Problem statement

1.3 Contributions

In this thesis, we propose two frameworks for the subgraph matching problem: Subgraph Optimal Transport (SOT) and Sliding Subgraph Optimal Transport (SSOT), with adopting the Fused-Gromov Wasserstein (FGW) distance. Both the proposed frameworks incorporate a dummy node strategy, bridging the gap between the classical (sub)graph matching quadratic assignment problem and the partial optimal transport problem. Based on the dummy node settings, we develop specific techniques to reduce the computational complexity. We further propose a normalized FGW (nFGW) distance fitting for performance evaluation and practical interests. We conduct comprehensive experiments for performance evaluation of the algorithms. We also compare our method with existing methods and provide examples of real-world applications. We will further discuss our contributions at the end of the thesis in Section 6.1.

1.4 Outline

- **Chapter 2** presents the basic concepts of (sub)graph matching, and a review of subgraph matching problems.
- **Chapter 3** introduces the background of optimal transport and FGW distance.

- **Chapter 4** formally proposes our algorithms for subgraph matching in details.
- **Chapter 5** presents the performance evaluation of the algorithm on synthetic datasets and real-world datasets, as well as practical examples in the biomedical field.
- **Chapter 6** concludes the thesis and suggests future research directions.

2 Background

2.1 Graph basics

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a pair of two sets, where $\mathcal{V}$ is the set of nodes (also known as vertices), and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. A graph emphasizes the connections or relationships between nodes, in contrast to points in Euclidean space where the specific positions or distances between them are important. A labeled graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \ell_f)$ (with only labels on nodes) includes an additional labeling function ℓ_f that assigns labels to nodes [1]. For each node $v_i \in \mathcal{V}$,

$$\ell_f : \mathcal{V} \to \mathcal{A}, \quad a_i \overset{\text{def.}}{=} \ell_f(v_i), \tag{2.1}$$

where $\mathcal{A}$ is the set that includes all the possible labels assigned to the nodes, and a_i is the label for node v_i. In this thesis, unless otherwise specified, we always consider graphs that are undirected, non-weighted, labeled on nodes, and possibly with self-loops. Some fundamental terms of graph used in this thesis are listed as follows.

Adjacency matrix. The adjacency matrix of a graph of size n is a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, in which each entry $\boldsymbol{A}_{i,j}$ indicates the number of edges directly between nodes v_i and v_j in $\mathcal{V}$. This matrix provides the complete information of a graph's topology. For an undirected graph with no self-loops , the adjacency matrix is symmetric, with zeros on the diagonal. For a graph that allows self-loops, the diagonals can also be one.

Walk. A walk is a succession of nodes and edges, in which the nodes and edges can be possibly repeated. The length of walk is defined as the number of edges in the sequence. The number of l-length walks between node v_i and v_j can be computed as $\left(\boldsymbol{A}^l\right)_{i,j}$, in which $\boldsymbol{A}^l$ denotes the l-th power of the matrix $\boldsymbol{A}$.

Path. A Path is a special kind of walk, in which both nodes and edges can not be repeated. Unfortunately, there exists no solution for the number of l-length paths between every pair of nodes. Nevertheless, the number of *shortest* paths between two nodes can be obtained, since a shortest walk is also a shortest path.

To obtain the length of shortest path between node v_i and v_j, compute the entry $\left(\boldsymbol{A}^l\right)_{i,j}$ recursively with $l = 1, 2, ...,$ until the first nonzero result $a_{i,j}$ appears when $l = L$. This means there are $a_{i,j}$ different shortest paths of length L between node v_i and v_j. Node v_i is a **L-hop** node of node v_j, and vice versa.

[1] Labels, features, and attributes are used interchangeably in this thesis. All of them refer to some characteristics or properties that nodes are endowed with.

Let $d(v_i, v_j)$ denotes the **shorstest-path distance** between nodes v_i and v_j. For nodes v_i, v_j, v_k in $\mathcal{V}$, the lengths of shortest paths between them satisfy the following properties:

1. Positivity: $d(v_i, v_j) \geq 0$, with equality if and only if $v_i = v_j$ (i.e., $i = j$),
2. Symmetry: $d(v_i, v_j) = d(v_j, v_i)$,
3. Triangle inequality: $d(v_i, v_j) \leq d(v_i, v_k) + d(v_k, v_j)$.

Thus, the shortest-path distance defines a metric on the node set $\mathcal{V}$. For future discussion, we define a **shortest-path distance matrix** $\boldsymbol{D}$ is with each entry defined as $\boldsymbol{D}_{i,j} \overset{\text{def.}}{=} d(v_i, v_j)$.

2.2 Subgraph matching: a review

2.2.1 Basics

Definition. A subgraph of a graph $\mathcal{G}$ is another graph formed from a subset of the nodes and edges of $\mathcal{G}$, with the natural requirement that endpoints of all the edges in the subset are included in the node subset. An *induced* subgraph of $\mathcal{G}$ is a subgraph that includes all edges in $\mathcal{G}$ that connect the nodes of the subgraph.

Exact matching. The term *graph isomorphism* is usually used to define exact graph matching, which decides if two labeled graphs are identical. Based on this, we can further define the exact subgraph matching.

Definition 1 (Graph isomorphism of labeled graphs)**.** [2, 3] Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \ell_f)$ and $\mathcal{G}' = \left(\mathcal{V}', \mathcal{E}', \ell'_f\right)$ be two labeled graphs, where ℓ_f and ℓ'_f are two labeling functions. Graphs $\mathcal{G}$ and $\mathcal{G}'$ are isomorphic if and only if there exists a bijection $f : \mathcal{V} \to \mathcal{V}'$ satisfying: $\forall u, v \in \mathcal{V}$,

1. $\ell_f(u) = \ell'_f(f(u))$
2. $(u, v) \in \mathcal{E} \Leftrightarrow (f(u), f(v)) \in \mathcal{E}'$

The bijection f is called the *isomorphism* of graphs $\mathcal{G}$ and $\mathcal{G}'$

The classical (induced) exact subgraph matching problem is usually defined by the *(induced) subgraph isomorphism*, that is, graph $\mathcal{G}$ is isomorphic to an (induced) subgraph of $\mathcal{G}'$.

Related problems. Multiple graph-related problems share similar ideas or methods to subgraph matching. *Graph clustering* involves grouping nodes of one graph into clusters based on their overall connectivity or similarity. The common practice is the nodes within one cluster are strongly connected or similar with each other. On the other hand, *clustering of graphs* intends to classify a number of graphs into different different categories, based on various characteristics of the graphs. *Graph partitioning* divides a graph into several partitions, aiming to minimize the interactions between

them. *(Sub)Graph mining* aims to automatically detect subgraphs with some specific criteria [4].

Different from the problems above, *subgraph matching* focuses on identifying predefined patterns or structures that occur within a single large graph, or a set of graphs. Detecting a predefined pattern in multiple graphs is referred to as the *frequent* subgraph finding problem [5, 6]. Similar ideas can be found in temporal subgraph matching, in which people aim to find a certain pattern in a few time stamps but may not occur in most of the time stamps [7, 8].

2.2.2 Graph matching: the quadratic assignment program

The early stage graph matching problems are formulated as a quadratic assignment program (QAP). In this section, we introduce the classic Koopmans-Beckmann's QAP and Lawler's QAP, in terms of two **unlabeled**, **equal size** graphs.

For two graphs (referred to as *source* graph and *query* graph to align with the notations in the following chapters) of n nodes, suppose we have structure matrices $\boldsymbol{C}^s \in \mathbb{R}^{n\times n}$ and $\boldsymbol{C}^q \in \mathbb{R}^{n\times n}$ for each graph. A structure matrix represents the structure of a graph. Two examples of a structure matrix are the adjacency matrix and the shortest-path distance matrix. We aim to find an mapping matrix $\boldsymbol{X} \in \{0,1\}^{n\times n}$ that can match the two graphs that are as similar as possible. The entry $\boldsymbol{X}_{i,j}$ indicates whether node i in the source graph matches with node j in the query graph (1 for a match, 0 otherwise).

Koopmans-Beckmann's QAP. A traditional formulation is to compare two structure matrices after permutation. We want to minimize the discrepancy of $\boldsymbol{C}^s$ and the permuted version of $\boldsymbol{C}^q$. The mapping matrix $\boldsymbol{X}$ is constrained to be a permutation matrix, with only a single one in each row and column. Totally there are n ones and $n^2 - n$ zeros in $\boldsymbol{X}$.

$$\min_{\boldsymbol{X}} \ \left\|\boldsymbol{C}^s - \boldsymbol{X}\boldsymbol{C}^q\boldsymbol{X}^\top\right\|_F^2 \tag{2.2}$$

$$\text{s.t. } \boldsymbol{X} \in \mathcal{X} = \left\{\boldsymbol{X} \in \{0,1\}^{n\times n} \mid \boldsymbol{X}\mathbf{1}_n = \mathbf{1}_n, \boldsymbol{X}^\top\mathbf{1}_n = \mathbf{1}_n\right\} \tag{2.3}$$

The objective function can be expanded as [9]

$$\left\|\boldsymbol{C}^s - \boldsymbol{X}\boldsymbol{C}^q\boldsymbol{X}^\top\right\|_F^2 = \|\boldsymbol{C}^s\boldsymbol{X} - \boldsymbol{X}\boldsymbol{C}^q\|_F^2 \tag{2.4}$$

$$= \|\boldsymbol{C}^s\|_F^2 + \|\boldsymbol{C}^q\|_F^2 - 2\langle\boldsymbol{C}^s\boldsymbol{X}, \boldsymbol{X}\boldsymbol{C}^q\rangle_F. \tag{2.5}$$

Since the first and second terms in (2.5) are constant, the original formulation can be equivalently written as

$$\min_{\boldsymbol{X}\in\mathcal{X}} \ \left\{-\langle\boldsymbol{C}^s\boldsymbol{X}, \boldsymbol{X}\boldsymbol{C}^q\rangle_F = -\text{tr}(\boldsymbol{C}^s\boldsymbol{X}\boldsymbol{C}^{q\top}\boldsymbol{X}^\top) = -\langle\boldsymbol{C}^s, \boldsymbol{X}\boldsymbol{C}^q\boldsymbol{X}^\top\rangle_F\right\} \tag{2.6}$$

or

$$\max_{\boldsymbol{X}\in\mathcal{X}} \ \langle\boldsymbol{C}^s, \boldsymbol{X}\boldsymbol{C}^q\boldsymbol{X}^\top\rangle_F \tag{2.7}$$

The formulation $\langle\boldsymbol{C}^s, \boldsymbol{X}\boldsymbol{C}^q\boldsymbol{X}^\top\rangle_F$ is the same as the original quadratic assignment problem proposed by Koopmans and Beckmann [10, 11]. This formulation and its

variations are commonly named as Koopmans-Beckmann's QAP.

Lawler's QAP. A generalized version of Koopmans-Beckmann's QAP is the Lawler's QAP [10, 12]. The cost function first defines cost functions within each graph and then defines the costs built on each pair of nodes. After this, it summarizes all the costs based on all pairs of nodes. For each graph, let $\boldsymbol{C}^s_{i,i'} = \phi(v_i, v_{i'})$ and $\boldsymbol{C}^q_{j,j'} = \phi(u_j, u_{j'})$. Let us define a cost function $d^S : \mathbb{R} \times \mathbb{R} \to [0, 1]$ between the two elements, such that value 0 indicates no difference, and value 1 indicates the most difference. We want to minimize

$$\min_{\boldsymbol{X} \in \mathcal{X}} \sum_{i,i',j,j'} d^S(\boldsymbol{C}^s_{i,i'}, \boldsymbol{C}^q_{j,j'}) \boldsymbol{X}_{i,j} \boldsymbol{X}_{i',j'} \tag{2.8}$$

The structure costs $d^S(\boldsymbol{C}^s_{i,i'}, \boldsymbol{C}^q_{j,j'})$ for all node pairs $(v_i, v_{i'})$ and $(u_j, u_{j'})$ can be arranged into a matrix $\boldsymbol{Q} \in \mathbb{R}^{n^2 \times n^2}$. Then the formulation is equivalent to

$$\min_{\boldsymbol{X} \in \mathcal{X}} \ \mathrm{vec}(\boldsymbol{X})^\top \boldsymbol{Q} \, \mathrm{vec}(\boldsymbol{X}) \tag{2.9}$$

where each entry of $\boldsymbol{Q}$ is

$$\boldsymbol{Q}_{n(j-1)+i, n(j'-1)+i'} = d^S(\boldsymbol{C}^s_{i,i'}, \boldsymbol{C}^q_{j,j'}) \tag{2.10}$$

and $\mathrm{vec}(\boldsymbol{X})$ represents vectorization of $\boldsymbol{X}$ via column stack.

$$\mathrm{vec}(\boldsymbol{X}) = (\boldsymbol{X}_{1,1}, ..., \boldsymbol{X}_{n,1}, \boldsymbol{X}_{1,2}, ..., \boldsymbol{X}_{n,2}, ..., \boldsymbol{X}_{1,n}, ..., \boldsymbol{X}_{n,n})^\top. \tag{2.11}$$

Relationships. Koopmans-Beckmann's QAP is a special case of the Lawler's QAP (2.8) and (2.9) with

$$d^S(\boldsymbol{C}^s_{i,i'}, \boldsymbol{C}^q_{j,j'}) = -\boldsymbol{C}^s_{i,i'} \boldsymbol{C}^q_{j,j'}. \tag{2.12}$$

or

$$\boldsymbol{Q} = -\boldsymbol{C}^q \otimes_K \boldsymbol{C}^s \tag{2.13}$$

Where $\otimes_K$ denotes the Kronecker product. This applies similarly if d^S is set to be L^2 norm, since

$$d^S(\boldsymbol{C}^s_{i,i'}, \boldsymbol{C}^q_{j,j'}) = |\boldsymbol{C}^s_{i,i'} - \boldsymbol{C}^q_{j,j'}|^2 \tag{2.14}$$

$$= |\boldsymbol{C}^s_{i,i'}|^2 + |\boldsymbol{C}^q_{j,j'}|^2 - 2\boldsymbol{C}^s_{i,i'} \boldsymbol{C}^q_{j,j'}. \tag{2.15}$$

The first two terms to be applied in the cost function turns out to be

$$\sum_{i,i',j,j'} \left(|\boldsymbol{C}^s_{i,i'}|^2 + |\boldsymbol{C}^q_{j,j'}|^2\right) \boldsymbol{X}_{i,j} \boldsymbol{X}_{i',j'} = \sum_{i,i',j,j'} \left(|\boldsymbol{C}^s_{i,i'}|^2 + |\boldsymbol{C}^q_{j,j'}|^2\right) = \sum_{i,i'} |\boldsymbol{C}^s_{i,i'}|^2 + \sum_{j,j'} |\boldsymbol{C}^q_{j,j'}|^2 \tag{2.16}$$

which is only a scalar. Thus the problem is equivalent to Koopmans-Beckmann's QAP in this case with the definition of 2.12.

The cost matrices in these two formulations are usually defined as (weighted) **adjacency matrices** [9] for graph matching. These two formulations both consider *pairs* of nodes (or say edges). Formulation of this kind is named as the *second-order* It is also possible to consider *high-order* cost matrix [13, 10], to jointly consider multiple nodes within each graph. These programs are **non-convex** naturally due to the joint matching of two node endings.

2.2.3 Subgraph matching

We have so far discussed graph matching for two graphs of equal size, in which each node in one graph is matched to a unique node in the other graph, and vice versa. This forms a one-to-one correspondence, creating a perfect permutation matrix $\boldsymbol{X}$ with exactly one "1" in each row and each column.

When we extend these ideas to subgraph matching, where the source graph n is larger than the query graph m, we do not require that there is always a "1" in each row. Instead, we aim to find a partial permutation matrix with the following constraints:

$$\mathcal{X} = \left\{ \boldsymbol{X} \in \{0,1\}^{n \times m} \mid \boldsymbol{X}\mathbf{1}_m \leq \mathbf{1}_n, \boldsymbol{X}^\top \mathbf{1}_n = \mathbf{1}_m \right\}. \tag{2.17}$$

These constraints allow for only m rows containing a "1", indicating which nodes in the source graph are matched with nodes in the query graph. The objective function for the optimization problem remains the same as in the equal-size case.

To solve this subgraph matching problem, we can adopt different approaches. One can design optimization algorithms that directly utilize the feasible set $\mathcal{X}$ [14]. Alternatively, we can leverage the techniques developed for classic graph matching QAP's by introducing a possibly different number of *dummy nodes* into the query graph to effectively balance the sizes of the two graphs. Dummy nodes, also referred to as virtual nodes, are non-connected nodes added to the query graph to make up for the size discrepancy. They serve as placeholders to transform the inequality constraints into equality constraints, similar to the role of *slack variables* in linear programming [15]. By incorporating dummy nodes, we also revise the structure cost function $d^S(\boldsymbol{C}^s_{i,i'}, \boldsymbol{C}^q_{j,j'})$ to zero for any component involving these dummy nodes, ensuring they do not contribute to the total cost. This allows us to use existing optimization methods while still accounting for the different graph sizes [12, 15].

One can choose to add a different number of dummy nodes. The simplest way is to add $n-m$ dummy nodes to the query graph to make the two graphs of exactly the same size [12]. However, this method can be demanding regarding its computational costs. Alternatively, one can only add one dummy node to the query graph with modifications of the feasible set. This idea can be equivalently illustrated by adding slack variables as follows.

The inequality constraint (2.17) can be equivalently written by equality constraints by adding slack variables $\boldsymbol{s} \in \mathbb{R}^n$ as

$$\mathcal{X} = \left\{ \boldsymbol{X} \in \{0,1\}^{n \times m} \mid \boldsymbol{X}\mathbf{1}_m + \boldsymbol{s} = \mathbf{1}_n, \boldsymbol{X}^\top \mathbf{1}_n = \mathbf{1}_m, \boldsymbol{s} \in \{0,1\}^n \right\}. \tag{2.18}$$

These constraints can be reformulated by expanding the matrix $\boldsymbol{X}$ to include the slack variables as an additional column, creating a new matrix $\boldsymbol{Y}$. The revised constraints

are:

$$\mathcal{Y} = \left\{ \boldsymbol{Y} \in \{0,1\}^{n\times(m+1)} \mid \boldsymbol{Y}\mathbf{1}_{m+1} = \mathbf{1}_n, \boldsymbol{Y}^\top \mathbf{1}_n = [\mathbf{1}_m; \|\boldsymbol{s}\|_1] \right\}. \tag{2.19}$$

Actually $\|\boldsymbol{s}\|_1 = n - m$. We can notice that this extension of the mapping matrix $\boldsymbol{X}$ is equivalent to adding one dummy node to the query graph. By integrating the slack variables into the objective function as dummy nodes and setting their related costs to zero, we can proceed with the optimization without altering the function's original intent. This method essentially allows us to apply established graph matching algorithms (such as graph matching QAP's) to the subgraph matching scenario.

2.2.4 Subgraph matching for labeled graphs

For labeled graphs, we have to jointly consider feature cost and structure cost. A general way is to use linear combinations to combine the information of two parts. As default, for both feature and structure cost, we set 0 to indicate no difference, and 1 to indicate the most difference. With Lawler's QAP for structure cost, the objective function is written as

$$\min_{\boldsymbol{X}\in\mathcal{X}} \quad (1-\alpha)\cdot \text{FeatureCost} + \alpha \cdot \text{StructureCost} \tag{2.20}$$

where

$$\text{FeatureCost} = \sum_{i,j} d^F(i,j)\boldsymbol{X}_{i,j}, \tag{2.21}$$

$$\text{StructureCost} = \sum_{i,i',j,j'} d^S(\boldsymbol{C}^s_{i,i'}, \boldsymbol{C}^q_{j,j'})\boldsymbol{X}_{i,j}\boldsymbol{X}_{i',j'} \tag{2.22}$$

Since the feature cost is defined by one single node with another, this kind of cost is also referred as the *first-order* cost.

2.2.5 Related works

Different works utilize these formulations, frequently found in computer vision research. The tensor-styled formulation (2.8) can be found in [13, 16, 17, 18, 19]. The matrix-styled formulation (2.9) can be found in [10, 20, 21, 22]. Some works view the node features as external information for the original unlabeled graphs. They have shown that additional features can actually help find the optimal solutions [9] compared with the original QAP problem.

Instead of the second-order cost matrix, in practice, first-order structure cost is also considered,

$$\text{StructureCost} = \sum_{i,j} d^S(i,j)\boldsymbol{X}_{i,j} \tag{2.23}$$

An example of this simple framework is shown in NeMa [23]. For the structure cost, they first defined the *proximity* and *neighborhood vector* within each graph, and then calculated the cost of the neighborhood vectors of nodes v_i and u_j respectively in two graphs. Similarly, in G-Finder [24], the structure cost is defined using the number of neighbors of each node.

2.2.6 Practical issues and solutions

Nevertheless, the exact subgraph matching problem is known to be NP-complete in nature [25], and hard to be approximated with polynomial-time algorithms [23]. Balancing between the accuracy and efficiency of the algorithms is a tricky question. Different algorithms may be chosen depending on specific interests and applications.

2.2.6.1 From discrete to continuous

The graph matching QAP's are all integer programs and result in a *combinatorial* optimization problem that requires a large amount of computation [26, 16]. A common category of sub-optimal algorithms relax this *discrete* optimization problem into a *continuous* one, and thus can relieve computation burden by using classical techniques like gradient descent. More specifically, the discrete mapping matrix $\boldsymbol{X} \in \{0,1\}^{n\times m}$ is relaxed to a continuous one $\boldsymbol{D} \in \mathbb{R}^{n\times m}$. The continuous solutions are rounded in the final step when necessary.

The optimization problem now becomes

$$\min_{\boldsymbol{D}\in\mathcal{D}} \quad (1-\alpha)\cdot \text{FeatureCost} + \alpha \cdot \text{StructureCost} \tag{2.24}$$

where

$$\mathcal{D} = \left\{ \boldsymbol{D} \in \mathbb{R}_{+}^{n\times m} \mid \boldsymbol{D}\mathbf{1}_m \leq \mathbf{1}_n, \boldsymbol{D}^{\top}\mathbf{1}_n = \mathbf{1}_m \right\}. \tag{2.25}$$

From a probabilistic view, we relax a *deterministic* mapping into a *probabilistic* mapping. One node $v \in \mathcal{V}^q$ is allowed to match with multiple nodes $u \in \mathcal{V}^s$ with different probabilities.

Instead of relaxing the mapping matrix directly, there are also other continuation methods. In [22], the original objective function is convolved with a Gaussian function to obtain a new continuous function.

2.2.6.2 Categories

As mentioned in Section 2.2.6.1, the non-convexity of the problem brings up computational issues. Thus, what we obtain is often a suboptimal solution. Besides, practically the exact matching itself may not exist, and that is what we refer to as inexact matching. Further details are illustrated below. Note that the words "inexact" and "approximate" are often used interchangeably in the literature, but we assign them with different meanings for essential different objectives.

Exact and approximate matching. Assume the exact matching exists in the test graph. Due to the large computational burden required for exact subgraph matching, *suboptimal* methods are proposed for practical use [27]. These methods are mostly heuristic and have no guarantee of converging the optimal solutions [23]. Usually, these methods prioritize efficiency and practicality, while aiming to find solutions that are only acceptable for certain criteria. We refer to these suboptimal solutions as *approximate matching*. The continuation method in Section 2.2.6.1 is also a suboptimal

method.

Inexact matching. In practice, an exact matching itself may not exist. Firstly, the dataset is often noisy and incomplete. Secondly, researchers may aim to detect some similar patterns within a test graph, not to find the exact matching from the outset. We refer these cases to be *inexact matching*. In general, we still want to find the *optimal* subgraph that most closely resembles the query. In order to tolerate more possible graph deformations, the conditions of exact matching algorithms are relaxed. For this *optimal* inexact matching, it can be viewed as a generalization of exact algorithms [28]. Analogously, there are also *suboptimal* solutions in inexact matching.

ε-suboptimal solutions. For both approximate matching and inexact matching, if the matching cost is less or equal to ε, we say an ε-suboptimal subgraph is found. Thus, an exact-matched subgraph is also an ε-suboptimal subgraph.

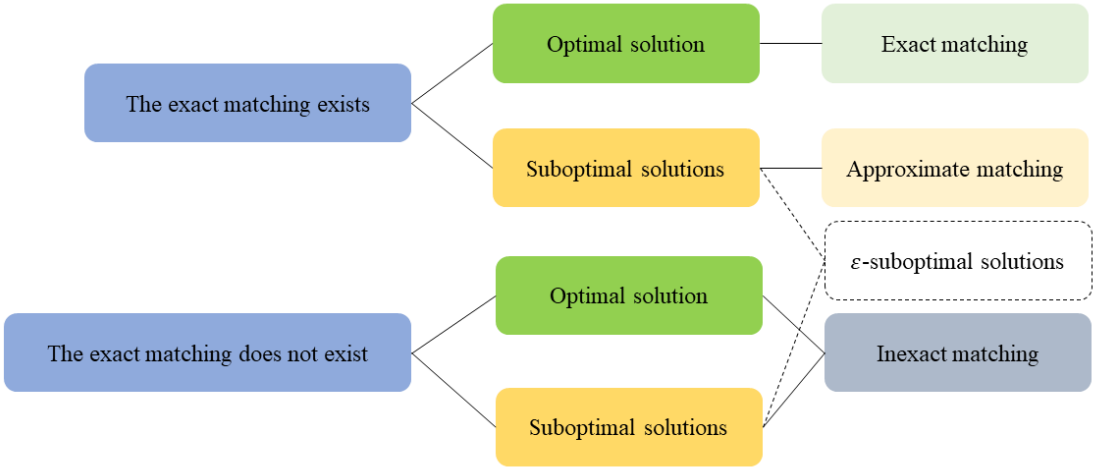


Figure 2.1: Exact, approximate and inexact matching

2.2.6.3 Graph database query and online query

There is a large category of methods for database query problems. These methods are commonly referred to as indexed-based methods, including SAGA [19], NeMa [23], and G-Finder [24], etc. The framework of these methods consists of the *off-line* preprocessing step (indexing step) and the *online* query step. The indexing step evaluates various statistics of the test graphs in the database beforehand, which relatively takes a large amount of time. Thanks to this beforehand information, the online query can be conducted efficiently.

These methods do not adopt a standard framework for constructing the loss function in the same way as the classic graph QAP's. Instead, they are purpose-built for graph query problems. They are particularly suitable for graph matching problems involving of a small query graph against an immense test graph, while QAP's mostly serve for

graphs of the same size or of the same order. Additionally, they tend to consider the subgraph matching problem from a practical perspective.

In terms of loss functions, SAGA used a second-order structure matrix; NeMa and G-Finder used a self-defined first-order matrix. The loss function of G-Finder is a weighted linear combination of costs of the number of missing query nodes, the number of missing query edges, and the number of intermediate nodes.

However, index-based methods are mostly suitable for *static* database, where indexing is only required once for multiple queries. They are not effective for non-static temporal/dynamic graphs. Since in these cases, the whole process will be *online*. Then it is the total processing time that matters, but not solely the query time.

2.2.7 Other frameworks

In general, instead of a linear combination of feature cost and structure cost, there are various ways to design an efficient cost function, with the idea of graph distance or dissimilarity. Researchers have explored various formulations, either heuristic or mathematically rigorous, to tackle this problem.

Graph edit distance. Graph edit distance has attracted great attention in recent years. It has a similar idea to optimal transport on graphs, i.e. transfer one graph to another. The total cost is summed up with the cost of each operation. This distance is known for its high computation complexity, and numerous works have tried to relieve its computational burden [29, 30, 31].

Graph kernels. Graph kernels quantify the similarity between graphs by mapping them into a high-dimensional feature space (different from the node features defined previously). They are well-fitted for kernel methods, such as support vector machines. It is an established method for graph classification. A review of graph kernels can be seen in [32]. For the subgraph matching problem, one may check [33].

Implicit cost functions. In VELSET, NAGA [34], and VerSaChI [35], the feature and structure cost function are not explicit cost functions. They evaluate graph similarity based on *statistical significance* captured by chi-square statistics. The optimization process of these methods resorts to choosing the node pair that is the most statistically significant.

Multiobjective optimization. [36] Instead of introducing a trade-off parameter α and form the FGW distance, a more direct way is to jointly optimize the objective functions of Wasserstein distance and Gromov-Wasserstein distance. Under the framework of *Pareto optimality*, multiple optimal results can be provided with different trade-offs of feature and structure, while it is more computationally expensive.

Graph neural network. Recent progress in graph neural networks (GNN) has attracted great attention. The fundamental understanding of GNN is still far from mature and thus limits its usage in high-risk areas. In [37], a neural subgraph matching

(NeuroMatch) is proposed. Test graph and query graph are first decomposed into small overlapping subgraphs, and embedded using graph neural network.

3 Optimal Transport for Graphs

3.1 Introduction

Optimal Transport, also known as optimal mass transport or the Wasserstein distance, is a mathematical framework used to measure the most efficient way to transport one distribution to another. The main objective is to determine the optimal transportation plan that minimizes the cost of moving a given quantity of mass from one configuration to another, considering the distance or cost associated with this transportation. This field offers powerful tools to analyze and solve problems related to data alignment, image comparison, and more. The sections in this chapter are formulated in a discrete context with the purpose of graph analysis. For the general formulation of optimal transport, please refer to Appendix A.1.

Relationship with structured data. Lots of objects can be modeled as probability measures by normalizing the weights/frequencies, such as graphs, images, documents, etc. Then we are able to compare these objects via metrics between probability measures with the tool of optimal transport.

Outline. We first introduce the traditional formulation of optimal transport and the Wasserstein distance that is rooted in probability theory in Section 3.2. Next, we introduce the probabilistic formulation of labeled graphs in Section 3.3. The framework of analyzing labeled graphs is introduced succeedingly with the Wasserstein distance (Section 3.4), Gromov-Wasserstein distance (Section 3.5), and Fused Gromov-Wasserstein distance (Section 3.6). Then, we present simple examples of these distances and the visualization of the non-convexity of GW and FGW distances in Section 3.7. Lastly, we illustrate the connections between FGW distance and classic graph matching QAP's in Section 3.8.

3.2 Optimal transport and Wasserstein distance

3.2.1 The background problem: mines and factories

The historical problem of optimal transport is a "real" transportation problem in civil engineering, also called the Earth Movers Distance (EMD) problem [38]. We aim to find a transport plan $\boldsymbol{T} \in \mathbb{R}^{n \times m}$ to transport mines from n minerals to m factories with the minimum total cost.

Suppose the problem is set on 2-dimensional Euclidean space. Mines are located at $\boldsymbol{x}^{(i)} \in \mathbb{R}^2, i = 1, ..., n$, while factories are located at $\boldsymbol{y}^{(j)} \in \mathbb{R}^2, j = 1, ..., m$. Each mines at $\boldsymbol{x}^{(i)}$ possesses a quantity of $\boldsymbol{P}_i$ minerals, i.e.-, minerals are *distributed* with

distribution $\boldsymbol{P} \in \mathbb{R}^n$ among all locations of mines. The minerals need to be transported to different factories, and we say the minerals received at different factories satisfy a distribution $\boldsymbol{Q} \in \mathbb{R}^m$. We only consider the case that the total minerals are conserved before and after transportation, i.e., $\boldsymbol{P}^\top \mathbf{1}_n = \boldsymbol{Q}^\top \mathbf{1}_m$, where $\mathbf{1}_n \in \mathbb{R}^n$ and $\mathbf{1}_m \in \mathbb{R}^m$ are all-ones vectors. Each entry $\boldsymbol{T}_{i,j}$ indicates the amount of mines that are transported from the ith mine to the jth factory, or from location $\boldsymbol{x}^{(i)}$ to location $\boldsymbol{y}^{(j)}$.

We let the Euclidean distance between $\boldsymbol{x}^{(i)}$ and $\boldsymbol{y}^{(j)}$ define the cost of transporting a unit of minerals between two locations. Further we define a cost matrix $\boldsymbol{M} \in \mathbb{R}^{n \times m}$ with entries $\boldsymbol{M}_{i,j} = \|\boldsymbol{x}^{(i)} - \boldsymbol{y}^{(j)}\|_2$. To minimize the total cost of transportation, the following constrained optimization problem is formulated.

$$\boldsymbol{T}^* = \underset{\boldsymbol{T}}{\operatorname{argmin}} \left\{ \langle \boldsymbol{T}, \boldsymbol{M} \rangle_F \overset{\text{def.}}{=} \sum_{i,j} \boldsymbol{T}_{i,j} \boldsymbol{M}_{i,j} \right\} \tag{3.1}$$

$$\text{s.t. } \boldsymbol{T}\mathbf{1}_m = \boldsymbol{P}; \boldsymbol{T}^\top \mathbf{1}_n = \boldsymbol{Q}; \boldsymbol{T} \geq \mathbf{0} \tag{3.2}$$

where $\langle \boldsymbol{T}, \boldsymbol{M} \rangle_F$ denotes the Frobenius inner product between two matrices, which can be also written as $\operatorname{tr}(\boldsymbol{T}^\top \boldsymbol{M})$ or $\operatorname{tr}(\boldsymbol{T}\boldsymbol{M}^\top)$. The constraint $\boldsymbol{T} \geq \mathbf{0}$ means all the entries of $\boldsymbol{T}$ should be non-negative.

The scope of this problem can cover any transportation scenario from n *sources* to m *targets*. Instead of mines and factories, the study objects can be diverse. Regarding our work, similar formulations are used for probability distributions and graphs. Instead of Euclidean distance, the cost matrix $\boldsymbol{M}$ can be defined specifically for different scenarios. Instead of minerals, the transport object is usually called *mass*.

3.2.2 Abstract formulation and Wasserstein distance

Since we supposed that the total mass remains unchanged as $\boldsymbol{P}^\top \mathbf{1}_n = \boldsymbol{Q}^\top \mathbf{1}_m$, we can obtain two probability vectors without loss of information, by normalizing the source and target distributions by the total mass,

$$\boldsymbol{p} \overset{\text{def.}}{=} \frac{\boldsymbol{P}}{\boldsymbol{P}^\top \mathbf{1}_n} \quad \text{and} \quad \boldsymbol{q} \overset{\text{def.}}{=} \frac{\boldsymbol{Q}}{\boldsymbol{Q}^\top \mathbf{1}_m}. \tag{3.3}$$

Then $\boldsymbol{p}^\top \mathbf{1}_n = \boldsymbol{q}^\top \mathbf{1}_m = 1$. Together with locations $\boldsymbol{x}^{(i)}$ and $\boldsymbol{y}^{(j)}$, we obtain two probability measures,

$$\mu_s = \sum_{i=1}^{n} \boldsymbol{p}_i \delta_{\boldsymbol{x}^{(i)}} \quad \text{and} \quad \mu_t = \sum_{j=1}^{m} \boldsymbol{q}_j \delta_{\boldsymbol{y}^{(j)}}. \tag{3.4}$$

where $\boldsymbol{p}_i$ is the ith element of $\boldsymbol{p}$, and $\boldsymbol{q}_j$ is the jth element of $\boldsymbol{q}$. The EMD problem can be then reformulated with probability measures. The optimal value of this objective function is defined as the Wasserstein distance.

Definition 2 (Wasserstein distance)**.** [38, 39, 40] Let μ_s and μ_t be two discrete probability measures in the same metric space (Ω, d_Ω), where d_Ω is the metric on space Ω. The Wasserstein distance between μ_s and μ_t is defined as

$$\mathcal{W}^\Omega(\mu_s, \mu_t) = \mathcal{W}(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{M}) \overset{\text{def.}}{=} \min_{\boldsymbol{T}} \langle \boldsymbol{T}, \boldsymbol{M} \rangle_F \tag{3.5}$$

$$\text{s.t. } \boldsymbol{T}\mathbf{1}_m = \boldsymbol{p}; \boldsymbol{T}^\top \mathbf{1}_n = \boldsymbol{q}; \boldsymbol{T} \geq \mathbf{0}. \tag{3.6}$$

where $\boldsymbol{p}$ and $\boldsymbol{q}$ are the probability vectors of μ_s and μ_t respectively. For the matrix $\boldsymbol{M}$, each entry $\boldsymbol{M}_{i,j} \overset{\text{def.}}{=} d_\Omega(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(j)})$ indicates the cost between points $\boldsymbol{x}^{(i)} \in \text{supp}[\mu_s]$, and $\boldsymbol{y}^{(j)} \in \text{supp}[\mu_t]$.

For the transport matrix $\boldsymbol{T}$, each entry $\boldsymbol{T}_{i,j}$ indicates the amount of mass that is transported from location $\boldsymbol{x}^{(i)}$ to location $\boldsymbol{y}^{(j)}$ [38]. It is also worth noting that $\sum_{i,j} \boldsymbol{T}_{i,j} = 1$. Based on this, the transport matrix $\boldsymbol{T}$ is actually a joint probability distribution with marginals $\boldsymbol{p}$ and $\boldsymbol{q}$. We define the feasible set containing all possible $\boldsymbol{T}$ as

$$\mathcal{T}(\boldsymbol{p}, \boldsymbol{q}) = \left\{ \boldsymbol{T} \in \mathbb{R}_+^{n \times m} \mid \boldsymbol{T}\mathbf{1}_m = \boldsymbol{p}, \boldsymbol{T}^\top \mathbf{1}_n = \boldsymbol{q} \right\}, \tag{3.7}$$

which is always convex and compact. Then, Definition 2 can be rewritten as

$$\mathcal{W}^\Omega(\mu_s, \mu_t) = \mathcal{W}(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{M}) \overset{\text{def.}}{=} \min_{\boldsymbol{T} \in \mathcal{T}(\boldsymbol{p}, \boldsymbol{q})} \langle \boldsymbol{T}, \boldsymbol{M} \rangle_F. \tag{3.8}$$

This optimization problem is a **Linear Program** and is typically easy to solve. In addition, the optimal solutions are not always unique [38]. Since the feasible set is a polyhedron, the problem has a unique solution if and only if the solution lies at the intersection of two or more constraints. If there is one solution that is situated within one edge of the polyhedron, then all points on this edge will be solutions to the problem.

The Wasserstein distance satisfies the following properties for all probability measures μ, ν and σ in the same metric space (Ω, d_Ω) [38]:

1. Positivity: $\mathcal{W}^\Omega(\mu, \nu) \geq 0$, with equality if and only if $\mu = \nu$,
2. Symmetry: $\mathcal{W}^\Omega(\mu, \nu) = \mathcal{W}^\Omega(\nu, \mu)$,
3. Triangle inequality: $\mathcal{W}^\Omega(\mu, \nu) \leq \mathcal{W}^\Omega(\mu, \sigma) + \mathcal{W}^\Omega(\sigma, \nu)$.

Thus the Wasserstein distance is a metric on the space of probability measures.

3.3 Labeled graphs as probability measures

A graph is a special kind of discrete structured object. Consider an undirected labeled graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \ell_f, h_\mathcal{G})$, where $\mathcal{V}$ and $\mathcal{E}$ denote its sets of nodes and edges. The function ℓ_f assigns feature information in some feature metric space $(\mathcal{A}, d_\mathcal{A})$, for each node $v_i \in \mathcal{V}$,

$$\ell_f : \mathcal{V} \to \mathcal{A}, \quad a_i \overset{\text{def.}}{=} \ell_f(v_i), \tag{3.9}$$

where $\mathcal{A}$ includes all the possible features on $\mathcal{V}$, and $d_\mathcal{A}$ defines the cost function in between. The function $h_\mathcal{G}$ assigns different weights to the nodes to indicate relative importance,

$$h_\mathcal{G} : \mathcal{V} \to \mathbb{R}, \quad \boldsymbol{h}_i \overset{\text{def.}}{=} h_\mathcal{G}(v_i). \tag{3.10}$$

Without loss of generality, we constrain $\boldsymbol{h}$ to be a probability vector, i.e., $\sum_i \boldsymbol{h}_i = 1$. The vector can be assigned differently for different purposes of the applications. Usually, equal weights are assigned if no *a prior* information is available.

For graphs or general structured objects, often only the *relative* positions matter. The structure information of a graph is embedded in its edge set $\mathcal{E}$. The neighborhood of each node *implicitly* embeds its structural information. Thus a function ℓ_s can be analogously defined and implies the relative relationship with other nodes, in some structure space $(\mathcal{X}, d_{\mathcal{X}})$,

$$\ell_s : \mathcal{V} \to \mathcal{X}, \quad x_i \overset{\text{def.}}{=} \ell_s(v_i), \tag{3.11}$$

where $\mathcal{X}$ includes all the possible relative structural information of nodes in $\mathcal{G}$, and ℓ_s defines the cost function in between. For graphs, x refers to a list of neighbors of each node [41], or generally neighborhood information [23, 42]. The metric $d_{\mathcal{X}}$ can be set as the shortest-path distance between two nodes (See further in Section 3.5).

To combine the above information, a graph with n nodes can be represented by a probability measure on the metric space $\mathcal{X} \times \mathcal{A}$,

$$\mu = \sum_{i=1}^{n} \boldsymbol{h}_i \delta_{(x_i, a_i)}. \tag{3.12}$$

Each node is attached with its own information of feature, structure, and relative importance. We say the triplet $(\mathcal{X} \times \mathcal{A}, d_{\mathcal{X}}, \mu)$ is a **structured object** on the product space $\mathcal{X} \times \mathcal{A}$ (See formal definition in Appendix A.1.4) [40]. On this metric space, μ entirely captures the information of $\mathcal{G}$. Two marginals of μ are μ^F and μ^S, respectively for the features and structure,

$$\mu^F = \sum_{i=1}^{n} \boldsymbol{h}_i \delta_{a_i} \quad \text{and} \quad \mu^S = \sum_{i=1}^{n} \boldsymbol{h}_i \delta_{x_i} \tag{3.13}$$

Suppose we have two labeled graphs (named source and target) represented by $(\mathcal{X} \times \mathcal{A}, d_{\mathcal{X}}, \mu_s)$ and $(\mathcal{Y} \times \mathcal{B}, d_{\mathcal{Y}}, \mu_t)$, with

$$\mu_s = \sum_{i=1}^{n} \boldsymbol{p}_i \delta_{(x_i, a_i)} \quad \text{and} \quad \mu_t = \sum_{j=1}^{m} \boldsymbol{q}_j \delta_{(y_j, b_j)}, \tag{3.14}$$

then we are able to compare them with the framework of optimal transport.

In the following sections, we first illustrate the feature comparison via the Wasserstein distance, then we present the structure comparison via the Gromov-Wasserstein distance. Finally, we combine the formulations for features and structure, and present the Fused Gromov-Wasserstein distance.

3.4 Wasserstein distance on graph

We consider the case that the feature measures of two graphs are within the same metric space (Ω, d_Ω), i.e., $\mathcal{A} \subset \Omega$ and $\mathcal{B} \subset \Omega$, represented by

$$\mu_s^F = \sum_{i=1}^{n} \boldsymbol{p}_i \delta_{a_i} \quad \text{and} \quad \mu_t^F = \sum_{j=1}^{m} \boldsymbol{q}_j \delta_{b_j}. \tag{3.15}$$

The Wasserstein distance can be directly used to compare these two measures.

For the cost matrix $\boldsymbol{M}$, we have $\boldsymbol{M}_{i,j} = d_\Omega(a_i, b_j)$. The distance function d_Ω can be defined in several ways. In order to adapt to a wide range of applications in practice, the cost function d_Ω is not necessarily a metric [43]. We refer to the general cost function as $d(a_i, b_j)$. In this thesis, we always set the cost function $d(a_i, b_j)$ to range between 0 and 1, either discrete or continuous. The two extreme values 0 and 1 are achieved when the matrix $\boldsymbol{M}$ is entirely filled with zeros or ones, respectively. We present two normalized cost functions that we will use in this thesis as follows.

- For discrete-valued or categorical features, we use Dirac distance for one-dimensional values,

$$d\left(a_i, b_j\right) = \delta\left(a_i, b_j\right) = \begin{cases} 1, & \text{for} \quad a_i \neq b_j \\ 0, & \text{for} \quad a_i = b_j. \end{cases} \tag{3.16}$$

- For multi-dimensional real-valued features, we use the normalized square L^2 norm distance,

$$d\left(\boldsymbol{a}^{(i)}, \boldsymbol{b}^{(j)}\right) = 1 - \frac{1}{1 + \|\boldsymbol{a}^{(i)} - \boldsymbol{b}^{(j)}\|_2^2}. \tag{3.17}$$

3.5 Gromov-Wasserstein Distance

Unfortunately, the Wasserstein distance is not able to compare objects across different metric spaces. If one object (e.g. a point cloud) embeds in 2-dimensional Euclidean space, and the other one embeds in 3-dimensional Euclidean space, we are unable to compare their structure directly by transporting the mass from node to node. It is also not straightforward to define a cost function across these two spaces. Fortunately, the Gromov-Wasserstein (GW) distance can play a role.

Regarding labeled graphs, we can use the GW distance to compare two structure measures respectively lying on two possibly different metric spaces $(\mathcal{X}, d_\mathcal{X})$ and $(\mathcal{Y}, d_\mathcal{Y})$,

$$\mu_s^S = \sum_{i=1}^{n} \boldsymbol{p}_i \delta_{x_i} \quad \text{and} \quad \mu_t^S = \sum_{j=1}^{m} \boldsymbol{q}_j \delta_{y_j}. \tag{3.18}$$

The GW distance first defines cost functions between each pair of points *within* different metric spaces, and then evaluates these costs *across* two spaces via another loss function. Even if the two objects are embedded in the same metric space, it is also beneficial to adopt the formulation of the GW distance, since it is the intra-relationship of an object that matters. For instance, in a graph, we only emphasize how the nodes are connected with each other, or what are the neighbors of each node. From a mathematical view, the GW distance is a relaxation of the Gromov–Hausdorf distance, which is a combinatorial optimization problem [26] (See details in Appendix A.1.2).

Suppose we obtain two structure matrices $\boldsymbol{C}^s \in \mathbb{R}^{n \times n}$ and $\boldsymbol{C}^t \in \mathbb{R}^{m \times m}$ for two graphs. The difference between $\boldsymbol{C}^s_{i,i'}$ and $\boldsymbol{C}^t_{j,j'}$ is evaluated by a loss function $\mathcal{L}$. We thus define a 4-dimensional tensor with elements [39]

$$\boldsymbol{L}_{i,i',j,j'} = \mathcal{L}\left(\boldsymbol{C}^s_{i,i'}, \boldsymbol{C}^t_{j,j'}\right). \tag{3.19}$$

The full tensor is denoted as

$$\boldsymbol{L} = \mathcal{L}\left(\boldsymbol{C}^s, \boldsymbol{C}^t\right) \in \mathbb{R}^{n\times n\times m\times m}. \tag{3.20}$$

With a transport matrix $\boldsymbol{T}$, the entries $\boldsymbol{T}_{i,j}$ and $\boldsymbol{T}_{i',j'}$ indicate the amount of mass flowing from node i to node j, and node i' to node j', respectively. By taking the product as $\boldsymbol{T}_{i,j}\boldsymbol{T}_{i',j'}$, we obtain the *joint* probability of matching two pairs of nodes (i,j) and (i',j') from the two graphs. This joint probability can also be illustrated as the mass transported from the pair of nodes (i,i') in the source, to the pair of nodes (j,j') in the target (See further in Paragraph "Transport by pair of nodes").

Definition 3 (Gromov-Wasserstein distance)**.** [38, 39, 26, 40] Let μ_s and μ_t be two discrete probability measures that possibly live in different metric spaces $(\mathcal{X}, d_{\mathcal{X}})$ and $(\mathcal{Y}, d_{\mathcal{Y}})$, where $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$ are metrics on spaces $\mathcal{X}$ and $\mathcal{Y}$ respectively. The Gromov-Wasserstein distance between μ_s and μ_t is defined as

$$\mathcal{GW}^{\mathcal{X},\mathcal{Y}}(\mu_s,\mu_t) = \mathcal{GW}(\boldsymbol{p},\boldsymbol{q},\boldsymbol{C}^s,\boldsymbol{C}^t) \overset{\text{def.}}{=} \min_{\boldsymbol{T}\in\mathcal{T}(\boldsymbol{p},\boldsymbol{q})} \sum_{i,i',j,j'} \mathcal{L}\left(\boldsymbol{C}^s_{i,i'}, \boldsymbol{C}^t_{j,j'}\right)\boldsymbol{T}_{i,j}\boldsymbol{T}_{i',j'} \tag{3.21}$$

where $\boldsymbol{p}$ and $\boldsymbol{q}$ are the probability vectors of μ_s and μ_t respectively. The structure matrices are defined by $\boldsymbol{C}^s_{i,i'} = d_{\mathcal{X}}(x_i, x_{i'})$, and $\boldsymbol{C}^t_{j,j'} = d_{\mathcal{Y}}(y_j, y_{j'})$. The loss function $\mathcal{L}$ is some loss that measures the difference between $\boldsymbol{C}^s$ and $\boldsymbol{C}^t$.

The optimization problem involved is a **non-convex Quadratic Program** due to the product of two $\boldsymbol{T}_{i,j}$'s. The non-convexity can be understood from Lawler's QAP by analogy, since the matrix $\boldsymbol{Q}$ in (2.9) is not always a positive semi-definite matrix. It is generally an NP-hard problem [44]. If we say the structure matrices $\boldsymbol{C}^s$ and $\boldsymbol{C}^t$ indicate the *similarity* of the matrices, and $\boldsymbol{L}$ indicates the *distortion* of each pair of elements, the GW distance is built on the *distortion* of the *similarity* between each *pair* of points in both metric spaces [45].

For graphs, the general choice of the structure matrices $\boldsymbol{C}^s$ and $\boldsymbol{C}^t$ is the shortest-path distance matrix $\boldsymbol{D}$, since the shortest-path distance is a valid metric on a structure space (as shown in Section 2.1). Each element $\boldsymbol{D}_{i,j}$ indicates the shortest-path distance between nodes v_i and v_j. Other choices of structure matrix include edge information, or general neighborhood information between two nodes [44]. Similar to what we discussed in Section 3.4, the choices of structure matrices are not restricted to distance matrices in practice [43]. Adjacency matrices $\boldsymbol{A}$ are also frequently adopted in practice [46].

The loss function $\mathcal{L}$ is usually defined as square L^2 norm distance as $\mathcal{L}(a,b) \overset{\text{def.}}{=} |a-b|^2$. It can be also defined by the Kullback-Leibler (KL) divergence as $\mathcal{L}(a,b) = KL(a|b) \overset{\text{def.}}{=} a\log(a/b) - a + b$ [43].

Similar to the settings of the Wasserstein distance, in this thesis we use normalized versions of the cost functions for $\mathcal{L}$ as default, i.e., $\mathcal{L}(a,b)$ ranges between 0 and 1, either discrete or continuous. Note that the metric $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$ do not need to satisfy this requirement. The extreme values 0 and 1 are achieved when the tenser $\boldsymbol{L}$ is entirely filled with zeros or ones, respectively.

Transport by pair of nodes. For the graph QAP, several linearization methods have been proposed [10, 47]. Similar ideas can be adopted for the GW distance. Let us define another 4-dimensional tensor as

$$\boldsymbol{\Pi}_{i,i',j,j'} \overset{\text{def.}}{=} \boldsymbol{T}_{i,j}\boldsymbol{T}_{i',j'}, \tag{3.22}$$

which has the same element order as tensor $\boldsymbol{L}$. The objective function can be rewritten as

$$\mathcal{GW}(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{C}^s, \boldsymbol{C}^t) = \langle \boldsymbol{\Pi}, \boldsymbol{L} \rangle_F \tag{3.23}$$

However, the constraints need to be modified and the problem suffers from a large number of additional variables and constraints [10, 47]. We do not go further into details of this new formulation since solving this problem is also complex.

Nevertheless, this formulation provides another interpretation of the GW distance by considering pairs of nodes on graph matching. Assume that we have obtained the optimal transport matrix $\boldsymbol{T}^*$. Then we automatically obtain the optimal transport tensor $\boldsymbol{\Pi}^*$. Each element $\boldsymbol{\Pi}^*_{i,i',j,j'}$ indicates how the mass is transported from the pair (i, i') to the pair (j, j'), and $\sum_{i,i',j,j'} \boldsymbol{\Pi}^*_{i,i',j,j'} = 1$.

In addition, the tensors $\boldsymbol{\Pi}$ and $\boldsymbol{L}$ can be rearranged into $(n \times n) \times (m \times m)$ matrices. If we organize the indices (i, i') and (j, j') by increasing orders, the matrix version of $\boldsymbol{\Pi}$ can be directly written by the Kronecker product as $\boldsymbol{\Pi} = \boldsymbol{T} \otimes_K \boldsymbol{T}$. This representation comes from the definition of tensor $\boldsymbol{\Pi}$ (3.22). The marginals of $\boldsymbol{\Pi}$ are $\boldsymbol{p} \otimes_K \boldsymbol{p}$ and $\boldsymbol{q} \otimes_K \boldsymbol{q}$. Please refer to the example in Section 3.7.2 for more details.

Reformulations. The objective function of the GW distance can be reformulated concisely by introducing the tensor-matrix product [39],

$$(\boldsymbol{L} \otimes \boldsymbol{T})_{i,j} \overset{\text{def.}}{=} \sum_{i',j'} \boldsymbol{L}_{i,i',j,j'}\boldsymbol{T}_{i',j'}. \tag{3.24}$$

that is $\boldsymbol{L} \otimes \boldsymbol{T} \in \mathbb{R}^{n \times m}$, acts as the similar role as feature cost matrix $\boldsymbol{M}$. Then the cost function can be rewritten as

$$\sum_{i,i',j,j'} \boldsymbol{L}_{i,i',j,j'}\boldsymbol{T}_{i,j}\boldsymbol{T}_{i',j'} \tag{3.25}$$

$$= \sum_{i,j} \left(\sum_{i',j'} \boldsymbol{L}_{i,i',j,j'}\boldsymbol{T}_{i',j'} \right) \cdot \boldsymbol{T}_{i,j} \tag{3.26}$$

$$= \sum_{i,j} (\boldsymbol{L} \otimes \boldsymbol{T})_{i,j} \cdot \boldsymbol{T}_{i,j} \tag{3.27}$$

$$= \langle \boldsymbol{L} \otimes \boldsymbol{T}, \boldsymbol{T} \rangle_F \tag{3.28}$$

Then Definition 3 can be rewritten as

$$\mathcal{GW}^{\mathcal{X},\mathcal{Y}}(\mu_s, \mu_t) = \mathcal{GW}(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{C}^s, \boldsymbol{C}^t) = \min_{\boldsymbol{T} \in \mathcal{T}(\boldsymbol{p}, \boldsymbol{q})} = \langle \boldsymbol{L} \otimes \boldsymbol{T}, \boldsymbol{T} \rangle_F \tag{3.29}$$

3.6 Fused Gromov-Wasserstein distance

To include both feature and structure information, the Fused Gromov-Wasserstein (FGW) distance is formulated as a convex combination of the Wasserstein distance and the GW distance.

Definition 4 (Fused Gromov-Wasserstein distance)**.** [39] Suppose two labeled graphs are represented by $(\mathcal{X} \times \mathcal{A}, d_{\mathcal{X}}, \mu_s)$ and $(\mathcal{Y} \times \mathcal{B}, d_{\mathcal{Y}}, \mu_t)$, with

$$\mu_s = \sum_{i=1}^{n} \boldsymbol{p}_i \delta_{(x_i, a_i)} \quad \text{and} \quad \mu_t = \sum_{j=1}^{m} \boldsymbol{q}_j \delta_{(y_j, b_j)}, \tag{3.30}$$

in which sets $\mathcal{A}$ and $\mathcal{B}$ are within the same metric space (Ω, d_Ω), while $\mathcal{X}$ and $\mathcal{Y}$ are possibly within different metric spaces. Definitions of $\boldsymbol{M}$, $\boldsymbol{C}^s$, $\boldsymbol{C}^t$, and $\boldsymbol{L}$ are set the same as in Definitions 2, 3 and Equation (3.20). With a scalar $\alpha \in [0, 1]$, the Fused Gromov-Wasserstein distance between μ_s and μ_t is defined as

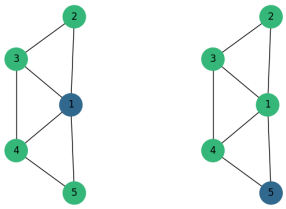
$$\mathcal{FGW}_\alpha^{\Omega,\mathcal{X},\mathcal{Y}}(\mu_s, \mu_t) = \mathcal{FGW}_\alpha(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{M}, \boldsymbol{C}^s, \boldsymbol{C}^t) \overset{\text{def.}}{=} \min_{\boldsymbol{T} \in \mathcal{T}(\boldsymbol{p}, \boldsymbol{q})} (1-\alpha)\langle \boldsymbol{T}, \boldsymbol{M} \rangle_F + \alpha \langle \boldsymbol{L} \otimes \boldsymbol{T}, \boldsymbol{T} \rangle_F \tag{3.31}$$

When $\alpha \to 0$, it is equivalent to the Wasserstein distance, while when $\alpha \to 1$, it is equivalent to the GW distance. Since one component of the formulation is the same as the objective function of the GW distance, the FGW distance is also a **non-convex Quadratic Program**.

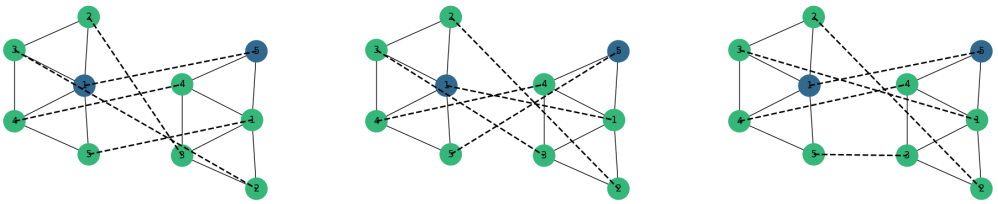
3.7 Examples on graphs

3.7.1 Example 1

Consider the following two graphs shown in Figure 3.1a, with colors indicating the node features [39]. The Wasserstein coupling, GW coupling, and FGW coupling between the two graphs are shown in Figure 3.1. We can obtain that both the Wasserstein distance and the Gromov-Wasserstein distance are zero, since the two graphs are of the same structure and contain the same features (both graphs contain one blue node and four green nodes). By setting parameter $\alpha = 0.5$, we are able to obtain the FGW distance is 0.16. Therefore, both the Wasserstein distance and the GW distance vanish, and only the FGW distance is able to distinguish between the two graphs.



(a) Two simple graphs



(b) Wasserstein coupling (c) GW coupling (d) FGW coupling

Figure 3.1: **Example 1 of Wasserstein coupling, GW coupling, and FGW coupling:** The dashed lines show the couplings between the nodes. The trade-off parameter is set to $\alpha = 0.5$ in FGW coupling.

3.7.2 Example 2

Consider the two simple graphs shown in Figure 3.2, with colors indicating the node features. Let the graph with three nodes be the source graph (left), and the graph with two nodes be the target graph (right). Both graphs are assigned with uniform weights $\boldsymbol{p} = \text{unif}(3)$ and $\boldsymbol{q} = \text{unif}(2)$.

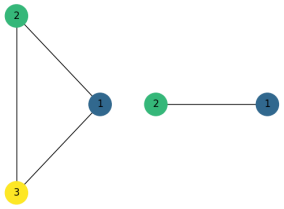


Figure 3.2: Simple graphs of Example 2.

The feature cost matrix $\boldsymbol{M}$ and structure matrices $\boldsymbol{C}^s$ and $\boldsymbol{C}^t$ are listed below, in which we use Dirac distance for the feature cost and adjacency matrices for the structure matrices.

$$\boldsymbol{M} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad \boldsymbol{C}^s = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \quad \boldsymbol{C}^t = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

The tensor $\boldsymbol{L} \in \mathbb{R}^{3\times3\times2\times2}$ shows the square L^2 norm costs between each pair of elements of $\boldsymbol{C}^s$ and $\boldsymbol{C}^t$. It can be represented in various dimensions by combining multiple dimensions. As shown below, $\boldsymbol{L}$ can be shown by $3 \times 3 \times 4$ (Left) or 9×4 (Right). In the first case, each 3×3 matrix shows the differences of $\boldsymbol{C}^s$ and one entry in $\boldsymbol{C}^t$. For example, $\boldsymbol{L}(:,:,1)$ shows the differences between $\boldsymbol{C}^s$ and the first entry 0 of $\boldsymbol{C}^t$. In the second case, each row is indexed by an edge (i, i') in the source graph, and each column is indexed by an edge (j, j') in the target graph. The indices are organized by increasing orders. Each 3×3 matrix on the left is stretched into a 9-dimensional column in the matrix on the right.

$$\boldsymbol{L}(:,:,1) = \boldsymbol{L}(:,:,4) = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

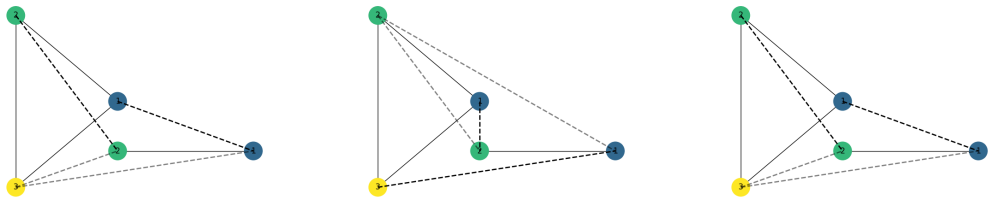
$$\boldsymbol{L}(:,:,2) = \boldsymbol{L}(:,:,3) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{L} = \begin{array}{c} \\ (i,i') \backslash (j,j') \\ (1,1) \\ (1,2) \\ (1,3) \\ (2,1) \\ (2,2) \\ (2,3) \\ (3,1) \\ (3,2) \\ (3,3) \end{array} \begin{array}{c} \begin{array}{cccc} (1,1) & (1,2) & (2,1) & (2,2) \end{array} \\ \left[\begin{array}{cc:cc} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ \hdashline 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ \hdashline 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right] \end{array}$$

Three couplings. We show the Wasserstein coupling, GW coupling, and FGW coupling in the following three figures in Figure 3.3. The results of the corresponding optimal transport matrices are listed below.

$$\boldsymbol{T}^*_{WD} = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/3 \\ 1/6 & 1/6 \end{bmatrix} \quad \boldsymbol{T}^*_{GWD} = \begin{bmatrix} 0 & 1/3 \\ 1/6 & 1/6 \\ 1/3 & 0 \end{bmatrix} \quad \boldsymbol{T}^*_{FGWD} = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/3 \\ 1/6 & 1/6 \end{bmatrix}$$

The nodes are attributed with three different colors: blue, green, and yellow. In the Wasserstein coupling, only features are taken into account, thus the blue nodes and green nodes are matched accordingly. In GW coupling, only the structure matters, so the features are actually in the wrong match in Figure 3.3b. One can imagine there are 6 equivalent coupling approaches with GW (discussed further in the next paragraph). In the FGW coupling, both features and structure are considered and optimally matched. Since these two graphs are not identical, the Wasserstein distance (0.333), the GW distance (0.278), and the FGW distance (0.306) are all larger than zero.



(a) Wasserstein coupling (b) GW coupling (c) FGW coupling

Figure 3.3: **Example 2 of Wasserstein coupling, GW coupling, and FGW coupling:** The dashed lines show the couplings between the nodes. Darker dashed lines indicate stronger connections between the two nodes (In this case, darker lines show 1/3 mass, while lighter lines show 1/6 mass.). The trade-off parameter is set to $\alpha = 0.5$ in FGW coupling.

Visualization of GW and FGW non-convexity. To further consider the GW distance and the FGW distance of this simple example, we visualize the objective functions of the GW distance and the FGW distance.

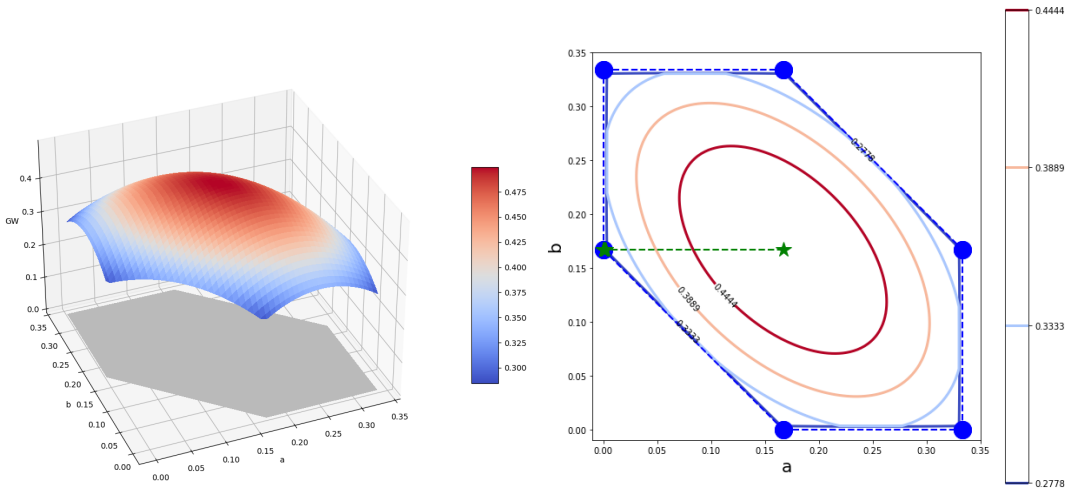
We set the first two entries of $\boldsymbol{T}$ to be the variables a and b. With the constraints of uniform probability vectors $\boldsymbol{p}$ and $\boldsymbol{q}$, we have to force each row to sum up to 1/3, and each column sums up to 1/2. Thus the remaining entries of $\boldsymbol{T}$ can be calculated by variables a and b. The whole matrix can be represented by a and b as

$$\boldsymbol{T} = \begin{bmatrix} a & 1/3 - a \\ b & 1/3 - b \\ 1/2 - a - b & -1/6 + a + b \end{bmatrix}.$$

Each entry of this matrix is also constrained to be a positive value. By substituting this $\boldsymbol{T}$ matrix, and the above $\boldsymbol{M}$,$\boldsymbol{C}^s$,$\boldsymbol{C}^t$ matrices into Definitions 3 and 4, the objective functions of both GW and FGW are only functions of two variables a and b.

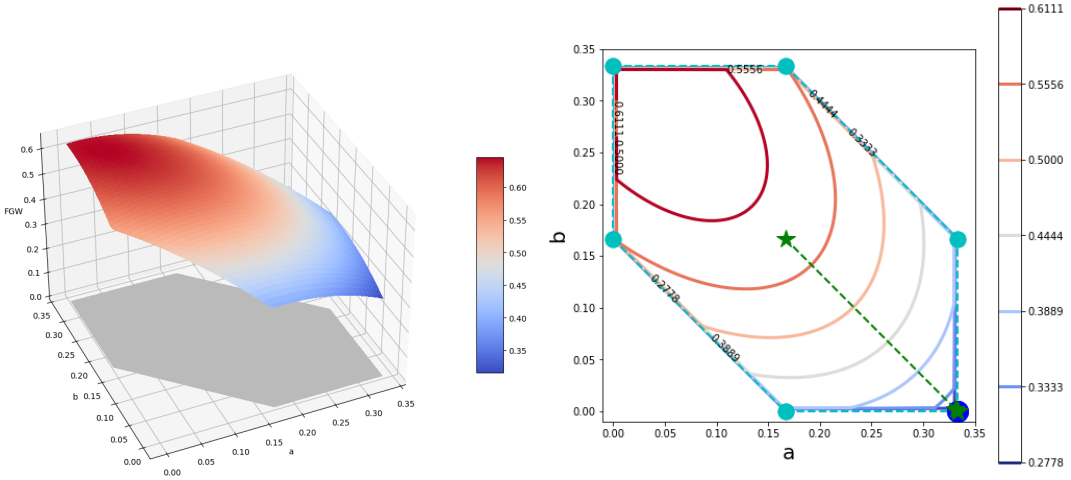
From the figures in Figure 3.4, we can easily see that the objective functions of the GW distance and the FGW distance are both non-convex. The GW objective function contains six equivalent global minima, with each corresponding to an optimal structure matching. The FGW objective function contains only one global minimum, together with five local minima. Thus, in this case, the optimal matching of FGW is unique. The global minima are notated in blue circles in contour plots for both GW and FGW distances. The five local minima of the FGW distance is notated in cyan circles.

The hexagonal feasible set $\mathcal{T}(\boldsymbol{p}, \boldsymbol{q})$ is shown in shadow in the surface plots, and framed in dashed lines in the contour plots. We can observe that all the local/global minima are located in the corners of the feasible set. The green dashed lines marked with stars show the optimization trajectories of GW and FGW distances (with implementation by Python Optimal Transport (POT) toolbox [48]). The matrix $\boldsymbol{T}$ is initialized at the point $\boldsymbol{p}\boldsymbol{q}^\top$, and then it quickly goes to a point very close to (one of) the global minimum (minima) in the second iteration.



(a) GW surface plot

(b) GW contour plot



(c) FGW surface plot

(d) FGW contour plot

Figure 3.4: **Surface plots and contour plots of GW and FGW objective functions:** The objective functions are plotted with two variables a and b. Function values from low to high are denoted by the color blue to red. The feasible set is shown in the gray shadow in the surface plots. The trade-off parameter is set to $\alpha = 0.5$ in FGW coupling.

Transport by pair of nodes. By taking the Kronecker product of optimal transport matrix $\boldsymbol{T}^*_{GWD}$ and obtaining the matrix version of optimal $\boldsymbol{\Pi}^* \in \mathbb{R}^{9\times 4}$, we can directly see how the mass is transported by pair of nodes as below.

$$\boldsymbol{\Pi}^* = \boldsymbol{T}^*_{GWD} \otimes_K \boldsymbol{T}^*_{GWD} =$$

(i,i') \ (j,j')	(1,1)	(1,2)	(2,1)	(2,2)
(1,1)	0	0	0	1/9
(1,2)	0	0	1/18	1/18
(1,3)	0	0	1/9	0
(2,1)	0	1/18	0	1/18
(2,2)	1/36	1/36	1/36	1/36
(2,3)	1/18	0	1/18	0
(3,1)	0	1/9	0	0
(3,2)	1/18	1/18	0	0
(3,3)	1/9	0	0	0

The matrix version of $\boldsymbol{\Pi}$ is a joint probability distribution with marginals $\boldsymbol{p} \otimes_K \boldsymbol{p} = \text{unif}(9)$, and $\boldsymbol{q} \otimes_K \boldsymbol{q} = \text{unif}(4)$, with the same increasing indices order as the matrix version of $\boldsymbol{L} \in \mathbb{R}^{9\times 4}$. In the matrix version of optimal $\boldsymbol{\Pi}^*$, the pair $(1,2)$ in the source graph owns $1/9$ amount of mass, which is transported to pair $(2,1)$ and pair $(2,2)$, each by $1/18$.

3.8 Connections with classic graph matching QAP's

In the field of graph matching, the FGW distance can be viewed as a generalization of both Koopmans-Beckmann's QAP and Lawler's QAP (introduced in Section 2.2.2), regarding its ability to compare graphs of different sizes, combine feature and structure information, and introduce relative importance to nodes [49].

As discussed in Section 2.2.6.1, the solution of a graph matching QAP can be approximated by relaxing the problem from a discrete domain to a continuous one. The continuous FGW problem owns analogous trajectories under the optimal transport framework. For the feature part, the Wasserstein distance (Kantorovitch formulation) is a continuous/probabilistic relaxation of the discrete/deterministic Monge formulation (Refer to Appendix A.1.1.). For the structure part, the GW distance is a continuous relaxation of the Gromov–Hausdorff distance (Refer to Appendix A.1.2.).

In Section 2.2.2, we presented that Lawler's QAP can be narrowed down to Koopmans-Beckmann's QAP if the structure loss function d^S is set to be square L^2 norm. Similarly, the GW problem can also be narrowed down to the continuous version of Koopmans-Beckmann's QAP if the loss function $\mathcal{L}$ is set as square L^2 norm [39].

Regarding the choice of structure matrix, the adjacency matrix is generally used for the graph matching QAP's. While theoretically, the adjacency matrix is not a natural choice for the FGW distance, since it is not a valid metric. While in practice, the adjacency matrix is also adopted for various applications [43, 46].

Furthermore, the FGW distance has more generalized usage, extending beyond graphs to any structured objects or metric spaces. It also makes cross-domain matching

possible between different kinds of objects, such as images and graphs.

4 Subgraph Matching

4.1 Introduction

In this chapter, we introduce two (induced) subgraph matching frameworks using the FGW distance, namely **S**ubgraph **O**ptimal **T**ransport (**SOT**), and **S**liding **S**ubgraph **O**ptimal **T**ransport (**SSOT**). Compared with the standard graph matching QAP's, the methods we proposed are more computationally efficient with satisfying matching accuracy. Despite the problem being continuous, in practice, we can always obtain a "sparse" optimal mapping matrix – meaning most of its entries are almost zero. This allows for an easy transition to a discrete mapping between graphs.

In this chapter, we introduce the components of our algorithms. In Section 4.2, we introduce the dummy node strategy, which directly forms the SOT method, and present the settings in detail. We then focus on the optimization method and computation issues in Section 4.3. Based on the settings in the dummy node strategy, we propose a normalized FGW distance in Section 4.4 for practical interests. In order to improve the general performance, we introduce the SSOT method in Section 4.5, in which we discuss two components: sliding window and Wasserstein pruning.

4.2 Dummy node strategy

In this section, we introduce the dummy node strategy to handle a large test graph and a small query graph. The section is presented with ideas and specific settings under the optimal transport framework.

Ideas. The dummy node strategy for graph matching has been introduced in Sections 2.2.3. We adopt the idea of adding one dummy node to the smaller query graph, as shown in Figure 4.1. If there is an induced subgraph same as the query graph in the test graph, there should exist an exact matching $\boldsymbol{T}$ that can match a certain part of the test graph to the query graph. Besides, the summation of weights on the subgraph nodes should be the same as that on the query graph. Under the optimal transport framework, we say the mass endowed with the subgraph is transported to the query graph, and the total mass is conserved before and after the transportation. Without *a prior* information, we assume that nodes in the test graph are endowed with uniform weights/mass.

While the mass of the subgraph is transported to the query, the mass from the remaining part of the test graph should also be transported. To "receive" this irrelevant mass, we add a *dummy* node to the query graph as a "dustbin". The test graph is considered as the *source*, while the combination of the query and the dummy node is considered as the *target*. The extra dummy node ensures that all mass is trans-

ported from the *source* to the *target*, and thus satisfies the requirements of the optimal transport framework.

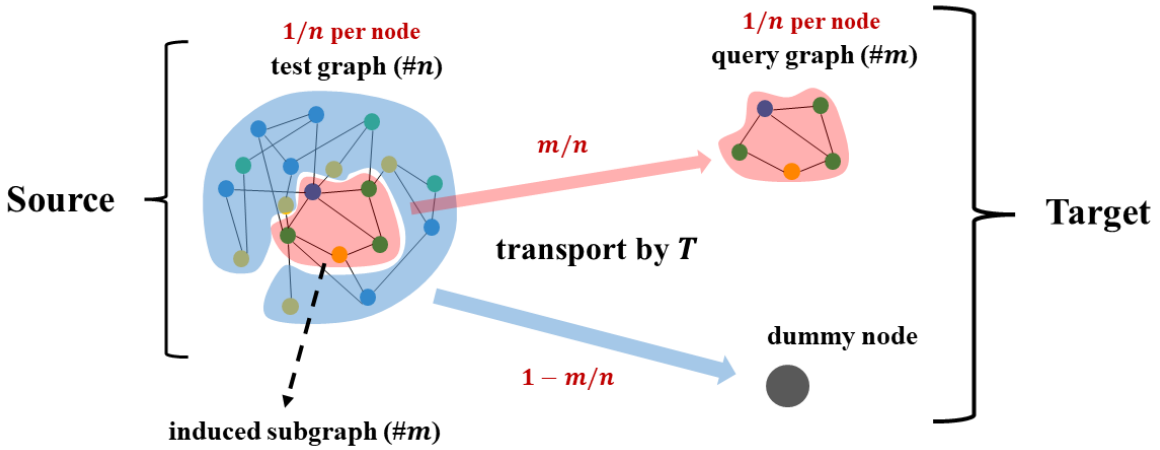


Figure 4.1: **Subgraph Optimal Transport (SOT) - dummy node strategy:** The subgraph (in red) should be matched to the query graph. The irrelevant part (in blue) should be matched to the dummy node.

We refer to this method as the Subgraph Optimal Transport (SOT). More generally, graphs of different sizes can be handled by adding dummy nodes to act as dustbins, which is not an unusual practice in graph-related tasks. Similar tricks can also be found in other graph matching/alignment/learning problems [14, 50, 51, 52].

Formulation. We now formulate the ideas above formally. With the optimal transport framework, the *source* is the test graph itself, denoted as $\mathcal{G}^s = (\mathcal{V}^s, \mathcal{E}^s)$. The *target* is the combination of the query graph $\mathcal{G}^q = (\mathcal{V}^q, \mathcal{E}^q)$ and a dummy node $\{dummy\}$. The sets $\mathcal{V}^s$, $\mathcal{E}^s$, $\mathcal{V}^q$, $\mathcal{V}^q$ are the node sets and edges respectively for graphs $\mathcal{G}^s$ and $\mathcal{G}^q$ with different superscripts. The dummy node can be assigned with any feature and is not connected to any node in the query graph. Thus the *target* is obtained as

$$\mathcal{G}^t \overset{\text{def.}}{=} (\mathcal{V}^q \cup \{dummy\}, \mathcal{E}^q)\,. \tag{4.1}$$

Suppose $|\mathcal{V}^s| = n$ and $|\mathcal{V}^q| = m$, with $n \geq m$. The source graph and target graph can be denoted as two probability measures,

$$\mu_s = \sum_{i=1}^{n} \boldsymbol{p}_i \delta_{(x_i, a_i)} \quad \text{and} \quad \mu_t = \sum_{j=1}^{m+1} \boldsymbol{q}_j \delta_{(y_j, b_j)}. \tag{4.2}$$

The parameters have the same meanings as in (3.30): probability vectors $\boldsymbol{p}$ and $\boldsymbol{q}$ denotes the distribution of mass on nodes; a_i and b_j are elements in the same feature space (Ω, d_Ω); x_i and y_j are elements in the structures spaces $(\mathcal{X}, d_\mathcal{X})$ and $(\mathcal{Y}, d_\mathcal{Y})$ respectively. Note that in μ_t, the dummy node is indexed by $j = m + 1$.

We aim to obtain the transport matrix via computing the FGW distance between the source and target,

$$\mathcal{FGW}_\alpha(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{M}, \boldsymbol{C}^s, \boldsymbol{C}^t) \overset{\text{def.}}{=} \min_{\boldsymbol{T} \in \mathcal{T}(\boldsymbol{p}, \boldsymbol{q})} (1-\alpha)\langle \boldsymbol{T}, \boldsymbol{M} \rangle_F + \alpha \langle \boldsymbol{L} \otimes \boldsymbol{T}, \boldsymbol{T} \rangle_F \tag{4.3}$$

To obtain the proper transportation of mass to both the query graph and the dummy node, we have the specific settings of $\boldsymbol{p}$, $\boldsymbol{q}$, and $\boldsymbol{M}$ and $\boldsymbol{L}$ as follows. We assume the nodes in the test graph have equal mass as $1/n$. Thus the total mass of the subgraph is m/n. The total mass of the query should be the same as that of the subgraph, so each node of the query is also set with equal mass as $1/n$. The remaining mass in the test graph should be fully transported to the dummy node. To sum up, the probability vectors $\boldsymbol{p} \in \mathbb{R}^n$ and $\boldsymbol{q} \in \mathbb{R}^{m+1}$ are set to be

$$\boldsymbol{p}_i = 1/n \quad \text{for} \quad i = 1, ..., n \tag{4.4}$$

$$\boldsymbol{q}_j = \begin{cases} 1/n, & \text{for} \quad j = 1, ..., m \\ 1 - m/n, & \text{for} \quad j = m+1. \end{cases} \tag{4.5}$$

We allow any mass of the test graph to be transported to the dummy node "for free", while the transportation cost to the query graph remains unchanged. The feature cost matrix $\boldsymbol{M} \in \mathbb{R}^{n \times (m+1)}$ is set to be

$$\boldsymbol{M}_{i,j} = \begin{cases} d_\Omega(a_i, b_j), & \text{for} \quad j = 1, ..., m \\ 0, & \text{for} \quad j = m+1. \end{cases} \tag{4.6}$$

To compute the structure tensor $\boldsymbol{L} = \mathcal{L}(\boldsymbol{C}^s, \boldsymbol{C}^t) \in \mathbb{R}^{n \times n \times (m+1) \times (m+1)}$, the structure cost matrices $\boldsymbol{C}^s \in \mathbb{R}^{n \times n}$ and $\boldsymbol{C}^t \in \mathbb{R}^{(m+1) \times (m+1)}$ are defined within the structure space $(\mathcal{X}, d_\mathcal{X})$ and $(\mathcal{Y}, d_\mathcal{Y})$ respectively. Then the computation of $\boldsymbol{L}$ is set to be

$$\boldsymbol{L}_{i,i',j,j'} = \begin{cases} \mathcal{L}\left(\boldsymbol{C}^s_{i,i'}, \boldsymbol{C}^t_{j,j'}\right), & \text{for} \quad j = 1, ..., m \quad \text{and} \quad j' = 1, ..., m \\ 0, & \text{for} \quad j = m+1 \quad \text{or} \quad j' = m+1 \end{cases} \tag{4.7}$$

By computing the FGW distance (4.3), we are also able to obtain an optimal transport matrix $\boldsymbol{T}^* \in \mathbb{R}^{n \times (m+1)}$ that indicates how the nodes in the test graph are matched with the query graph and dummy node. The last column of $\boldsymbol{T}^*$ (indexed by $j = m+1$) shows which nodes are matched to the dummy node.

For this subgraph matching problem, we do not restrict the **feature cost function** to be a metric d_Ω, but can be assigned with a general cost function d. The structure matrices are not restricted either. As default, we would like the cost functions d and $\mathcal{L}$ to use normalized versions (as discussed in Section 3.4 and 3.5). For d, we can possibly pick the Dirac function (3.16) or normalized square L^2 norm distance (3.17). For structure matrices, we can possibly pick the adjacency matrices or shortest-path distance matrices as discussed below.

Structure cost matrices. Here we present two choices of structure cost matrices, the adjacency matrix $\boldsymbol{A}$ and the shortest-path distance matrix $\boldsymbol{D}$.

For the **adjacency matrices** $\boldsymbol{A}^s$ and $\boldsymbol{A}^t$, the loss function $\mathcal{L}$ can be simply set as square L^2 norm as

$$\mathcal{L}\left(\boldsymbol{A}^s_{i,i'}, \boldsymbol{A}^t_{j,j'}\right) = |\boldsymbol{A}^s_{i,i'} - \boldsymbol{A}^t_{j,j'}|^2 \tag{4.8}$$

Then the entries of $\boldsymbol{L}$ are only discrete values of either 0 or 1.

With this setting, the FGW distance will be **zero** if an *exact* induced subgraph is successfully found in the test graph. A zero FGW distance also indicates whether an exact matching is found. Thus we can use a zero threshold of FGW distance to decide whether an exact induced subgraph matching is found.

The **shortest-path distance matrices** $\boldsymbol{D}^s$ and $\boldsymbol{D}^t$ are natural choices for FGW distance from a theoretical point of view, but we need to adopt special loss function $\mathcal{L}$ for adaptation to this subgraph matching problem. Assuming we adopt the square L^2 norm analogously, it can happen that the loss value is nonzero even if the induced subgraph is properly found. The reason is that the shortest-path distance between the same pair of nodes may not remain the same in the query graph as in the test graph. A simple example is shown in Figure 4.2. The graph in blue is the induced subgraph. The red node is outside of the subgraph but belongs to the test graph. Thus, in the induced subgraph, the distance between node 1 and node 4 is 3, while this distance in the test graph is 2. This difference will create a nonzero cost when we further compute with the $\mathcal{L}$ function.

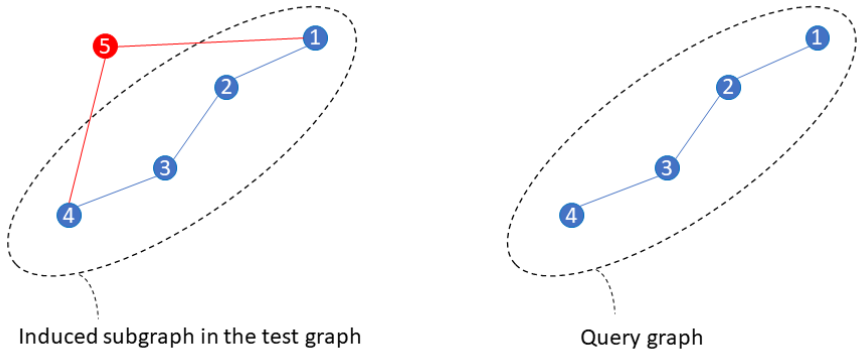


Figure 4.2: Issue with the shortest-path distance matrix

A remedy for this issue is to adopt the *proximity* and function $\Delta_+(x, y)$ presented in NeMa [23] [1]. Take $\boldsymbol{D}^s$ for an example, the proximity between two node i and i' is defined as $\alpha^{\boldsymbol{D}^s_{i,i'}}$, where α is an real value between $[0, 1]$. The function $\Delta_+(x, y)$ is defined as

$$\Delta_+(x, y) = \begin{cases} x - y, & \text{if } x > y \\ 0 & \text{otherwise} \end{cases} . \tag{4.9}$$

[1] The shortest-path distance matrix is also adopted in SAGA [19] but without special adaptation. Thus the cost value will not always be zero even if the exact matching is found, which is not appropriate.

Then we define

$$\mathcal{L}\left(\boldsymbol{D}^s_{i,i'}, \boldsymbol{D}^t_{j,j'}\right) = \Delta_+\left(\alpha^{\boldsymbol{D}^t_{j,j'}}, \alpha^{\boldsymbol{D}^s_{i,i'}}\right). \tag{4.10}$$

Thus only the cases when $\boldsymbol{D}^s_{i,i'} > \boldsymbol{D}^t_{j,j'}$, the loss is non-zero. If there are additional nodes in the test graph that act as intermediates between the two nodes, the shortest path in the test graph can possibly be shorter. Thus we do not want this case to contribute to the total cost.

Unfortunately, though the proposed remedy offers potential benefits, it comes at the cost of a significant increase in computational complexity. As we will present in Section 4.3.2, the function $\Delta_+(x, y)$ does not meet the conditions for adopting the computational trick. Consequently, the computational complexity will be much higher than the adjacency matrix approach with square L^2 norm, combined with the aforementioned computational trick.

Another potential issue is that the zero FGW distance does not exactly indicate whether an exact matching is found, though the converse side holds (if an exact matching is found, the FGW distance will be zero) [23].

4.3 Optimization and computation

The FGW distance inherently involves an optimization problem. Compared with the Wasserstein distance, it requires heavy additional computation due to the tensor-matrix product. This section presents the optimization algorithm and techniques for computation reduction.

To begin with, the FGW objective function and its gradient are shown as follows.

Proposition 1. [43, 14] The gradient of the FGW objective function,

$$\mathcal{J}(\boldsymbol{T}) = (1-\alpha)\langle \boldsymbol{T}, \boldsymbol{M}\rangle_F + \alpha\langle \boldsymbol{L}\otimes\boldsymbol{T}, \boldsymbol{T}\rangle_F, \quad \in \mathbb{R}. \tag{4.11}$$

is given by

$$\nabla_{\boldsymbol{T}}\mathcal{J}(\boldsymbol{T}) = (1-\alpha)\boldsymbol{M} + \alpha\cdot 2\cdot(\boldsymbol{L}\otimes\boldsymbol{T}), \quad \in \mathbb{R}^{n\times(m+1)}. \tag{4.12}$$

Proof of Proposition 1. The first parts of the functions are linear. The gradient of $\langle \boldsymbol{T}, \boldsymbol{M}\rangle_F$ is simply the matrix $\boldsymbol{M}$. Now we show that the second parts have the following relationship. For the function

$$\mathcal{C}(\boldsymbol{T}) = \langle \boldsymbol{L}\otimes\boldsymbol{T}, \boldsymbol{T}\rangle_F \quad \in \mathbb{R}, \tag{4.13}$$

its gradient is

$$\nabla_{\boldsymbol{T}}\mathcal{C}(\boldsymbol{T}) = 2\cdot(\boldsymbol{L}\otimes\boldsymbol{T}) \quad \in \mathbb{R}^{n\times(m+1)}. \tag{4.14}$$

The problem is to take the derivative of a scalar function with respect to the variable matrix $\boldsymbol{T}$. In the following, we take the derivative with respect to each entry $\boldsymbol{T}_{k,l}$ to obtain each entry of the gradient $(\nabla_{\boldsymbol{T}}\mathcal{C}(\boldsymbol{T}))_{k,l}$.

From definition (3.24), we expand the objective function as

$$\mathcal{C}(\boldsymbol{T}) = \langle \boldsymbol{L}\otimes\boldsymbol{T}, \boldsymbol{T}\rangle_F = \sum_{i,j}\left(\sum_{i',j'}\boldsymbol{L}_{i,i',j,j'}\boldsymbol{T}_{i',j'}\right)\cdot\boldsymbol{T}_{i,j} \tag{4.15}$$

Rewrite (4.15) as

$$\sum_{i,j}\left(\sum_{i',j'}\boldsymbol{L}_{i,i',j,j'}\boldsymbol{T}_{i',j'}\right)\cdot\boldsymbol{T}_{i,j} \tag{4.16}$$

$$=\left(\sum_{i',j'}\boldsymbol{L}_{k,i',l,j'}\boldsymbol{T}_{i',j'}\right)\cdot\boldsymbol{T}_{k,l}+\sum_{i\neq k,j\neq l}\left(\sum_{i',j'}\boldsymbol{L}_{i,i',j,j'}\boldsymbol{T}_{i',j'}\right)\cdot\boldsymbol{T}_{i,j} \tag{4.17}$$

Take the derivative of the first term with respect to $\boldsymbol{T}_{k,l}$ and obtain

$$\sum_{i',j'}\boldsymbol{L}_{k,i',l,j'}\boldsymbol{T}_{i',j'}+\left(\frac{\partial}{\partial\boldsymbol{T}_{k,l}}\sum_{i',j'}\boldsymbol{L}_{k,i',l,j'}\boldsymbol{T}_{i',j'}\right)\cdot\boldsymbol{T}_{k,l} \tag{4.18}$$

$$=(\boldsymbol{L}\otimes\boldsymbol{T})_{k,l}+\left(\frac{\partial}{\partial\boldsymbol{T}_{k,l}}\left(\boldsymbol{L}_{k,k,l,l}\cdot\boldsymbol{T}_{k,l}+\sum_{i'\neq k,j'\neq l}\boldsymbol{L}_{k,i',l,j'}\boldsymbol{T}_{i',j'}\right)\right)\cdot\boldsymbol{T}_{k,l} \tag{4.19}$$

$$=(\boldsymbol{L}\otimes\boldsymbol{T})_{k,l}+\boldsymbol{L}_{k,k,l,l}\cdot\boldsymbol{T}_{k,l} \tag{4.20}$$

Similarly, the second term can be split into

$$\sum_{i\neq k,j\neq l}(\boldsymbol{L}_{i,k,j,l}\boldsymbol{T}_{k,l})\cdot\boldsymbol{T}_{i,j}+\sum_{i\neq k,j\neq l}\left(\sum_{i'\neq k,j'\neq l}\boldsymbol{L}_{i,i',j,j'}\boldsymbol{T}_{i',j'}\right)\cdot\boldsymbol{T}_{i,j}, \tag{4.21}$$

in which the second composition is irrelevant of $\boldsymbol{T}_{k,l}$. Take the derivative of the first composition,

$$\frac{\partial}{\partial\boldsymbol{T}_{k,l}}\sum_{i\neq k,j\neq l}(\boldsymbol{L}_{i,k,j,l}\boldsymbol{T}_{k,l})\cdot\boldsymbol{T}_{i,j}=\sum_{i\neq k,j\neq l}\boldsymbol{L}_{i,k,j,l}\cdot\boldsymbol{T}_{i,j} \tag{4.22}$$

Combine (4.20) and (4.22) we have

$$(\nabla_{\boldsymbol{T}}\mathcal{C}(\boldsymbol{T}))_{k,l}=(\boldsymbol{L}\otimes\boldsymbol{T})_{k,l}+\boldsymbol{L}_{k,k,l,l}\cdot\boldsymbol{T}_{k,l}+\sum_{i\neq k,j\neq l}\boldsymbol{L}_{i,k,j,l}\cdot\boldsymbol{T}_{i,j} \tag{4.23}$$

$$=(\boldsymbol{L}\otimes\boldsymbol{T})_{k,l}+\sum_{i,j}\boldsymbol{L}_{i,k,j,l}\cdot\boldsymbol{T}_{i,j} \tag{4.24}$$

$$=(\boldsymbol{L}\otimes\boldsymbol{T})_{k,l}+\sum_{i,j}\boldsymbol{L}_{k,i,l,j}\cdot\boldsymbol{T}_{i,j} \tag{4.25}$$

$$=2\cdot(\boldsymbol{L}\otimes\boldsymbol{T})_{k,l} \tag{4.26}$$

where $\boldsymbol{L}_{i,k,j,l}=\boldsymbol{L}_{k,i,l,j}$ is defined by the symmetry of $\boldsymbol{C}^s$ and $\boldsymbol{C}^t$.

Thus the gradient of $\mathcal{C}(\boldsymbol{T})$ is

$$\nabla_{\boldsymbol{T}}\mathcal{C}(\boldsymbol{T})=2\cdot(\boldsymbol{L}\otimes\boldsymbol{T})\quad\in\mathbb{R}^{n\times(m+1)}. \tag{4.27}$$

Then we can conclude that the gradient of $\mathcal{J}(\boldsymbol{T})$ is

$$\nabla_{\boldsymbol{T}}\mathcal{J}(\boldsymbol{T})=(1-\alpha)\boldsymbol{M}+\alpha\cdot2\cdot(\boldsymbol{L}\otimes\boldsymbol{T})\quad\in\mathbb{R}^{n\times(m+1)}. \tag{4.28}$$

□

With the implementation of the FGW objective function and its gradient, multiple optimization methods can be adopted. Here we present the prevalent Frank-Wolfe algorithm [14, 9, 22, 12, 53].

4.3.1 Frank-Wolfe algorithm

Frank-Wolfe method is also called conditional gradient descent algorithm, which is a projection-free method for constrained optimization problem [54, 55]. The objective function is not required to be convex, but it should be "smooth" enough (i.e., differentiable with L-Lipschitz gradient). The feasible set is however needed to be convex and compact. It relies on solving a so-called *linear minimization oracle* (See (4.31) below). The steps in our problem are summarized as follows. The algorithm is presented in Algorithm 1.

At each iteration k, consider the first-order Taylor approximation of $\mathcal{J}(\boldsymbol{T})$ at $\boldsymbol{T}^{(k)}$,

$$\mathcal{J}(\boldsymbol{T}) \approx \mathcal{J}\left(\boldsymbol{T}^{(k)}\right) + \operatorname{tr}\left(\nabla\mathcal{J}\left(\boldsymbol{T}^{(k)}\right)^T \cdot \left(\boldsymbol{T} - \boldsymbol{T}^{(k)}\right)\right) \tag{4.29}$$

$$= \mathcal{J}\left(\boldsymbol{T}^{(k)}\right) + \left\langle \nabla\mathcal{J}\left(\boldsymbol{T}^{(k)}\right), \left(\boldsymbol{T} - \boldsymbol{T}^{(k)}\right)\right\rangle_F. \tag{4.30}$$

To minimize $\mathcal{J}(\boldsymbol{T})$ within the feasible set $\mathcal{T}(\boldsymbol{p}, \boldsymbol{q})$, it is only required to minimize the linear part $\left\langle \nabla\mathcal{J}\left(\boldsymbol{T}^{(k)}\right), \boldsymbol{T}\right\rangle_F$. We then obtain a new point $\hat{\boldsymbol{T}}^{(k)}$ by solving this minimization problem,

$$\hat{\boldsymbol{T}}^{(k)} \in \underset{\boldsymbol{T}\in\mathcal{T}(\boldsymbol{p},\boldsymbol{q})}{\arg\min} \left\langle \nabla\mathcal{J}\left(\boldsymbol{T}^{(k)}\right), \boldsymbol{T}\right\rangle_F. \tag{4.31}$$

The linear minimization oracle shares the same structure as the EMD problem (or Wasserstein distance). The notation $\in$ is used since the $\hat{\boldsymbol{T}}^{(k)}$ obtained is not always unique, which can be learned from the property of Wasserstein distance in Section 3.2.2. Then the descent direction is defined as

$$\boldsymbol{d}^{(k)} = \hat{\boldsymbol{T}}^{(k)} - \boldsymbol{T}^{(k)}. \tag{4.32}$$

$\boldsymbol{T}^{(k)}$ is updated with $\boldsymbol{d}^{(k)}$ and a step size $\gamma^{(k)} \in [0, 1]$ as

$$\boldsymbol{T}^{(k+1)} = \boldsymbol{T}^{(k)} + \gamma^{(k)}\boldsymbol{d}^{(k)} \tag{4.33}$$

$$= (1 - \gamma^{(k)})\boldsymbol{T}^{(k)} + \gamma^{(k)}\hat{\boldsymbol{T}}^{(k)} \tag{4.34}$$

where $\gamma^{(k)}$ is usually found by Armijo backtracking line search,

$$\gamma^{(k)} = \underset{\gamma\in[0,1]}{\arg\min}\ \mathcal{J}\left((1 - \gamma^{(k)})\boldsymbol{T}^{(k)} + \gamma^{(k)}\hat{\boldsymbol{T}}^{(k)}\right). \tag{4.35}$$

or can also be set as $\gamma^{(k)} = \frac{2}{k+2}$ [55].

From (4.34) we know that $\boldsymbol{T}^{(k+1)}$ is a convex combination of $\boldsymbol{T}^{(k)}$ and $\hat{\boldsymbol{T}}^{(k)}$. Since the feasible set is required to be convex, if the initial $\boldsymbol{T}^{(0)}$ is set to be feasible, $\boldsymbol{T}^{(k)}$ will be feasible in every step. The algorithm stops when the difference between the current objective value and that of the previous iteration falls below a certain tolerance Δ, or when the total number of iterations reaches the maximum N.

Algorithm 1: Frank-Wolfe Algorithm

Input: $\boldsymbol{p}$, $\boldsymbol{q}$, $\mathcal{J}(\boldsymbol{T})$, $\nabla\mathcal{J}(\boldsymbol{T})$, Tolerance Δ, MaxIter N
Output: $\boldsymbol{T}^*$

Initialization: $\boldsymbol{T}_0 = \boldsymbol{p}\boldsymbol{q}^T$, $n = 0$;
while $\delta < \Delta$ and $n < N$ **do**
  $\hat{\boldsymbol{T}}^{(k)} \in \underset{\boldsymbol{T}\in\mathcal{T}(\boldsymbol{p},\boldsymbol{q})}{\arg\min} \left\langle \nabla\mathcal{J}\left(\boldsymbol{T}^{(k)}\right), \boldsymbol{T}\right\rangle_F$; `/* Linear minimization oracle */`
  $\boldsymbol{d}^{(k)} = \hat{\boldsymbol{T}}^{(k)} - \boldsymbol{T}^{(k)}$;
  $\gamma^{(k)} = \underset{\gamma\in[0,1]}{\arg\min} \mathcal{J}\left(\boldsymbol{T}^{(k)} + \gamma\boldsymbol{d}^{(k)}\right)$; `/* Set step size` $\gamma^{(k)}$ `by linesearch */`
  $\boldsymbol{T}^{(k+1)} = \boldsymbol{T}^{(k)} + \gamma^{(k)}\boldsymbol{d}^{(k)}$;
  $\delta = \left|\mathcal{J}\left(\boldsymbol{T}^{(k+1)}\right) - \mathcal{J}\left(\boldsymbol{T}^{(k)}\right)\right|$;
  $n = n + 1$;
end
$\boldsymbol{T}^* = \boldsymbol{T}^{(k)}$;

4.3.2 Peyré's trick

Note that it is expensive to compute the tensor-matrix product $(\boldsymbol{L}\otimes\boldsymbol{T})$ directly by definition (3.24),

$$(\boldsymbol{L}\otimes\boldsymbol{T})_{i,j} \overset{\text{def.}}{=} \sum_{i',j'} \boldsymbol{L}_{i,i',j,j'}\boldsymbol{T}_{i',j'}. \tag{4.36}$$

In our case, it requires $\mathcal{O}(n^2(m+1)^2) = \mathcal{O}(n^2m^2)$ operations. The computation can be reduced by modifying the trick introduced by Peyré et al [43]. Based on this, the computation is reduced from $\mathcal{O}(n^2m^2)$ to $\mathcal{O}(n^2m + m^2n)$, where m is often much smaller than n. In the following sections, we first show the Peyré's trick, and then introduce how to modify it into our subgraph matching algorithm.

Peyré et al [43] proposed a computational method for computing $\boldsymbol{L}\otimes\boldsymbol{T}$, subject to specific conditions on the loss function $\mathcal{L}$.

Proposition 2 (Peyré's trick)**.** [43, Proposition 1] If the loss between the source graph $\mathcal{G}^s$ and target graph $\mathcal{G}^t$ can be written as

$$\mathcal{L}(a,b) = f_1(a) + f_2(b) - h_1(a)h_2(b) \tag{4.37}$$

for functions (f_1, f_2, h_1, h_2), then for any transport matrix $\boldsymbol{T}\in\mathcal{T}(\boldsymbol{p},\boldsymbol{q})$,

$$\mathcal{L}(\boldsymbol{C}^s,\boldsymbol{C}^t)\otimes\boldsymbol{T} = c_{\boldsymbol{C}^s,\boldsymbol{C}^t} - h_1(\boldsymbol{C}^s)\boldsymbol{T}h_2(\boldsymbol{C}^t)^\top. \tag{4.38}$$

where

$$c_{\boldsymbol{C}^s,\boldsymbol{C}^t} \overset{\text{def.}}{=} f_1(\boldsymbol{C}^s)\boldsymbol{p}\mathbf{1}_{m+1}^\top + \mathbf{1}_n\boldsymbol{q}^\top f_2(\boldsymbol{C}^t)^\top \tag{4.39}$$

is independent of $\boldsymbol{T}$.

Special cases. The square L^2 loss $\mathcal{L}(a,b) \overset{\text{def.}}{=} |a-b|^2$ satisfies (4.37) for $f_1(a) = a^2, f_2(b) = b^2, h_1(a) = a$ and $h_2(b) = 2b$. The KL loss $\mathcal{L}(a,b) = KL(a|b) \overset{\text{def.}}{=} a\log(a/b) - a + b$ satisfies (4.37) for $f_1(a) = a\log(a) - a, f_2(b) = b, h_1(a) = a$ and $h_2(b) = \log(b)$.

Proof of Proposition 2. We continue with the dimensions $\boldsymbol{L} \in \mathbb{R}^{n\times n\times(m+1)\times(m+1)}$ and $\boldsymbol{T} \in \mathbb{R}^{n\times(m+1)}$.

To substitute $\boldsymbol{p} = \boldsymbol{T}\mathbf{1}_{m+1}$ and $\boldsymbol{q} = \boldsymbol{T}^\top\mathbf{1}_{\boldsymbol{n}}$ into the calculation of tensor-matrix product,

$$\boldsymbol{L}\otimes\boldsymbol{T} = f_1(\boldsymbol{C}^s)\boldsymbol{p}\mathbf{1}_{m+1}^\top + \mathbf{1}_n\boldsymbol{q}^\top f_2(\boldsymbol{C}^t)^\top - h_1(\boldsymbol{C}^s)\boldsymbol{T}h_2(\boldsymbol{C}^t)^\top \tag{4.40}$$

$$= f_1(\boldsymbol{C}^s)\boldsymbol{T}\mathbf{1}_{\boldsymbol{m+1}}\mathbf{1}_{m+1}^\top + \mathbf{1}_n(\boldsymbol{T}^\top\mathbf{1}_{\boldsymbol{n}})^\top f_2(\boldsymbol{C}^t)^\top - h_1(\boldsymbol{C}^s)\boldsymbol{T}h_2(\boldsymbol{C}^t)^\top \tag{4.41}$$

Then for each entry,

$$(\boldsymbol{L}\otimes\boldsymbol{T})_{i,j} = \sum_{i',j'}\left[f_1(\boldsymbol{C}^s_{i,i'})\boldsymbol{T}_{i',j'}(\mathbf{1}_{m+1}\mathbf{1}_{m+1}^\top)_{j',j} + (\mathbf{1}_n\mathbf{1}_n)_{i,i'}\boldsymbol{T}_{i',j'}f_2(\boldsymbol{C}^t_{j',j}) - h_1(\boldsymbol{C}^s_{i,i'})\boldsymbol{T}_{i',j'}h_2(\boldsymbol{C}^t_{j',j})\right] \tag{4.42}$$

$$= \sum_{i',j'}\left[f_1(\boldsymbol{C}^s_{i,i'}) + f_2(\boldsymbol{C}^t_{j',j}) - h_1(\boldsymbol{C}^s_{i,i'})h_2(\boldsymbol{C}^t_{j',j})\right]\boldsymbol{T}_{i',j'} \tag{4.43}$$

If the loss can be written as

$$\mathcal{L}(a,b) = f_1(a) + f_2(b) - h_1(a)h_2(b), \tag{4.44}$$

we can continue equation (4.43) as

$$(\boldsymbol{L}\otimes\boldsymbol{T})_{i,j} = \sum_{i',j'}\mathcal{L}(\boldsymbol{C}^s_{i,i'}, \boldsymbol{C}^t_{j',j})\boldsymbol{T}_{i',j'} = \sum_{i',j'}\boldsymbol{L}_{i,i',j,j'}\boldsymbol{T}_{i',j'} \tag{4.45}$$

Thus the computation method is valid since it is equivalent to the definition of the tensor-matrix product. □

Peyré's trick in subgraph matching. We are unable to use Proposition 2 directly for our subgraph matching problem since our loss function (4.7) can not be written in form (4.37). However, note that if we first leave out the dummy node, the loss function related to the query nodes indeed satisfies this requirement. Let us define

$$\tilde{\boldsymbol{L}} = \mathcal{L}(\boldsymbol{C}^s, \boldsymbol{C}^q) \in \mathbb{R}^{n\times n\times m\times m}, \tag{4.46}$$

where $\boldsymbol{C}^s \in \mathbb{R}^{n\times n}$ is the structure cost matrix of the source graph, and $\boldsymbol{C}^q \in \mathbb{R}^{m\times m}$ is the structure cost matrix of the query graph. Then the original tensor $\boldsymbol{L} = \mathcal{L}(\boldsymbol{C}^s, \boldsymbol{C}^q) \in \mathbb{R}^{n\times n\times(m+1)\times(m+1)}$ can be obtained by adding zero elements to $\tilde{\boldsymbol{L}}$ in the dimensions of $j = m+1$ and $j' = m+1$. Further, the following proposition shows that we can directly implement the idea with subsequent computation of the tensor-matrix product, without directly modifying the tensor $\tilde{\boldsymbol{L}}$.

Proposition 3 (Tensor-matrix product with isolation of the dummy node)**.** The tensor matrix product $\boldsymbol{L} \otimes \boldsymbol{T} \in \mathbb{R}^{n \times (m+1)}$ can be obtained by $\tilde{\boldsymbol{L}} \otimes \tilde{\boldsymbol{T}} \in \mathbb{R}^{n \times m}$ by add a zero-column at the end of the matrix as

$$(\boldsymbol{L} \otimes \boldsymbol{T})_{i,j} = \begin{cases} \left(\tilde{\boldsymbol{L}} \otimes \tilde{\boldsymbol{T}}\right)_{i,j}, & \text{for} \quad j = 1, ..., m \\ 0, & \text{for} \quad j = m+1 \end{cases} \tag{4.47}$$

where $\tilde{\boldsymbol{T}} \in \mathbb{R}^{n \times m}$ is defined with the transport matrix $\boldsymbol{T}$ without the last column.

Figure 4.3 shows the relationship between the matrices with and without isolation of the dummy node.

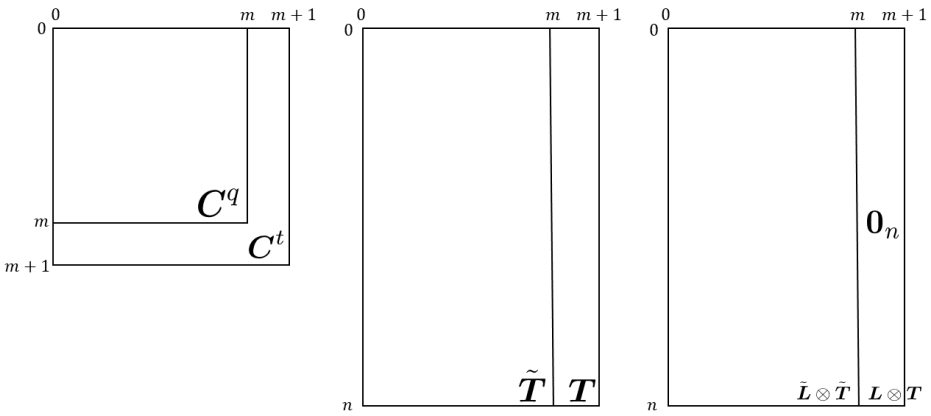


Figure 4.3: **New matrices and original ones.**

Proof of Proposition 3. The intuitive idea is the following. Since $\boldsymbol{L}_{i,i',j,j'}$ will be zero if $j = m+1$ or $j' = m+1$, by definition, all the entries in the last column of $\boldsymbol{L} \otimes \boldsymbol{T} \in \mathbb{R}^{n \times (m+1)}$ will also be zeros. Specific details are as follows.

The last column of this tensor-product matrix (when $j = m+1$) can be directly computed as

$$(\boldsymbol{L} \otimes \boldsymbol{T})_{i,j=m+1} = \sum_{i',j'} \boldsymbol{L}_{i,i',j=m+1,j'} \boldsymbol{T}_{i',j'} = 0 \tag{4.48}$$

The first m columns (when $j = 1, ..., m$) can be computed as follows

$$\begin{aligned}
(\boldsymbol{L} \otimes \boldsymbol{T})_{i,j} &= \sum_{i',j'=1,\ldots,m+1} \boldsymbol{L}_{i,i',j,j'} \boldsymbol{T}_{i',j'} \\
&= \sum_{i',j'=1,\ldots,m} \boldsymbol{L}_{i,i',j,j'} \boldsymbol{T}_{i',j'} + \boldsymbol{L}_{i,i',j,j'=m+1} \boldsymbol{T}_{i',j'=m+1} \\
&= \sum_{i',j'=1,\ldots,m} \boldsymbol{L}_{i,i',j,j'} \boldsymbol{T}_{i',j'} \\
&= \sum_{i',j'=1,\ldots,m} \tilde{\boldsymbol{L}}_{i,i',j,j'} \tilde{\boldsymbol{T}}_{i',j'} \\
&= \left(\tilde{\boldsymbol{L}} \otimes \tilde{\boldsymbol{T}}\right)_{i,j}
\end{aligned} \tag{4.49}$$

where $\boldsymbol{L}_{i,i',j,j'} = \mathcal{L}(\boldsymbol{C}^s_{i,i'}, \boldsymbol{C}^t_{j,j'})$ and $\tilde{\boldsymbol{L}}_{i,i',j,j'} = \mathcal{L}(\boldsymbol{C}^s_{i,i'}, \boldsymbol{C}^q_{j,j'})$ are equal when $j = 1, ..., m$ and $j' = 1, ..., m$.

□

Modification of Peyré's trick. Another issue appears subgraph matching with Peyré's trick comes down to matrix $\tilde{\boldsymbol{T}} \in \mathbb{R}^{n \times m}$ since it is an "incomplete" matrix compared with $\boldsymbol{T}$. From the definition in (4.4) let us define $\tilde{\boldsymbol{q}}_j = 1/n$ for $j = 1, ..., m$. From the definition of matrix $\tilde{\boldsymbol{T}}$ we know that its column marginal is $\tilde{\boldsymbol{T}}^\top \mathbf{1}_n = \tilde{\boldsymbol{q}}$, while its row marginal is $\tilde{\boldsymbol{T}}\mathbf{1}_m$ varies along the optimization process and longer equal to $\boldsymbol{p}$. Thus the matrix $\tilde{\boldsymbol{T}}$ does not belong to the feasible set $\mathcal{T}(\boldsymbol{p}, \tilde{\boldsymbol{q}})$.

Nevertheless, the transportation from the source graph to the query graph can be viewed as a special case *partial* optimal transport problem [14]. That is, only a partial part of the total mass is transported to the query graph. We can adopt the method presented in [14] to modify the current trick as below.

Proposition 4 (Peyré's trick for subgraph matching)**.** [14, 56] If the loss between the source graph $\mathcal{G}^s$ and the query graph $\mathcal{G}^q$ can be written as

$$\mathcal{L}(a, b) = f_1(a) + f_2(b) - h_1(a)h_2(b) \tag{4.50}$$

for functions (f_1, f_2, h_1, h_2), then for any transport matrix $\boldsymbol{T} \in \mathcal{T}(\boldsymbol{p}, \boldsymbol{q})$, define a partial transport $\tilde{\boldsymbol{T}} \in \mathbb{R}^{n \times m}$ by leaving out the last column of $\boldsymbol{T}$,

$$\mathcal{L}(\boldsymbol{C}^s, \boldsymbol{C}^q) \otimes \tilde{\boldsymbol{T}} = f_1(\boldsymbol{C}^s)\tilde{\boldsymbol{T}}\mathbf{1}_m\mathbf{1}_m^\top + \mathbf{1}_n(\tilde{\boldsymbol{T}}^\top \mathbf{1}_n)^\top f_2(\boldsymbol{C}^q)^\top - h_1(\boldsymbol{C}^s)\tilde{\boldsymbol{T}}h_2(\boldsymbol{C}^q)^\top. \tag{4.51}$$

Compared with Proposition 2, matrix $\boldsymbol{C}^t$ is substituted by $\boldsymbol{C}^q$, and $\boldsymbol{T}$ is substituted by $\tilde{\boldsymbol{T}}$. Since the marginal $\tilde{\boldsymbol{T}}\mathbf{1}_m$ is no longer $\boldsymbol{p}$ and varies during the optimization process, the positions of marginals $\boldsymbol{p}$ and $\boldsymbol{q}$ are replaced by $\tilde{\boldsymbol{T}}\mathbf{1}_m$ and $\tilde{\boldsymbol{T}}^\top \mathbf{l}_n$ respectively.

Proof of Proposition 4. The proof follows a similar approach to that of Proposition 2, with substitutions of the relevant matrices and vectors. Starting from step (4.41) with

the substitutions and proceeding with the subsequent steps, we can derive the desired results as

$$\left(\tilde{\boldsymbol{L}} \otimes \tilde{\boldsymbol{T}}\right)_{i,j} = \sum_{i',j'} \mathcal{L}(\boldsymbol{C}^s_{i,i'}, \boldsymbol{C}^q_{j',j}) \boldsymbol{T}_{i',j'} = \sum_{i',j'} \tilde{\boldsymbol{L}}_{i,i',j,j'} \tilde{\boldsymbol{T}}_{i',j'} \tag{4.52}$$

□

Complexity. By Proposition 3 and 4, the number of operations required is reduced to $\mathcal{O}\left(n^2 m + m^2 n\right)$. Algorithm 2 shows the process of using these propositions to calculate the objective function and its gradient. The cost matrix $\boldsymbol{C}^q \in \mathbb{R}^{m \times m}$ is obtained from $\boldsymbol{C}^t$ beforehand. At each iteration during the optimization progress, $\tilde{\boldsymbol{T}}$ is obtained by deleting the last column of $\boldsymbol{T}$. With Proposition 4 we are able to compute $\tilde{\boldsymbol{L}} \otimes \tilde{\boldsymbol{T}}$. We then add a zero column to obtain $\boldsymbol{L} \otimes \boldsymbol{T}$. Finally, we substitute $\boldsymbol{L} \otimes \boldsymbol{T}$ into the objective function and its gradient to finish the computation of these two functions.

Algorithm 2: Objective function $\mathcal{J}(\boldsymbol{T})$ and its gradient $\nabla \mathcal{J}(\boldsymbol{T})$

Input: $\alpha, \boldsymbol{M}, \boldsymbol{C}^s, \boldsymbol{C}^t, \boldsymbol{T}$
Output: $\mathcal{J}(\boldsymbol{T})$, $\nabla \mathcal{J}(\boldsymbol{T})$

1 $\boldsymbol{C}^q \leftarrow \boldsymbol{C}^t$ without the last row and last column;
2 $\tilde{\boldsymbol{T}} \leftarrow \boldsymbol{T}$ without the last column;
3 Compute $\left(\tilde{\boldsymbol{L}} \otimes \tilde{\boldsymbol{T}}\right)$ with Proposition 4;
4 $(\boldsymbol{L} \otimes \boldsymbol{T}) = \left[\left(\tilde{\boldsymbol{L}} \otimes \tilde{\boldsymbol{T}}\right), \boldsymbol{0}_n\right]$; `/* Add a zero column to` $\left(\tilde{\boldsymbol{L}} \otimes \tilde{\boldsymbol{T}}\right)$ `*/`
5 $\mathcal{J}(\boldsymbol{T}) = (1-\alpha)\langle \boldsymbol{T}, \boldsymbol{M} \rangle_F + \alpha \langle \boldsymbol{L} \otimes \boldsymbol{T}, \boldsymbol{T} \rangle_F$;
6 $\nabla \mathcal{J}(\boldsymbol{T}) = (1-\alpha)\boldsymbol{M} + \alpha \cdot 2 \cdot (\boldsymbol{L} \otimes \boldsymbol{T})$;

4.4 Normalized FGW distance

From the original definition of FGW distance, the distance would be between $[0, 1]$ as long as the outcomes of loss functions d and $\mathcal{L}$ are set to be within $[0, 1]$. However, due to the existence of zero elements in matrix $\boldsymbol{M}$ (4.6) and tensor $\boldsymbol{L}$ (4.7), there is always a part of the loss that automatically "vanish". From the idea shown in Figure 4.1 we know that only a fraction of m/n of mass contributes to the total cost. Therefore the maximum value of the FGW distance is no longer 1, but a value varying with sizes n and m. The minimum distance remains the same as 0. This creates difficulty in evaluating the distances between different sizes of graphs. Here we propose the normalized version of FGW distance for our formulation of this subgraph matching problem.

Definition 5 (Normalized FGW distance for subgraph matching)**.** Suppose we have a test graph of size n and a query graph of size m, with $n \geq m$. When the total cost is summarized only over the query graph, the normalized Wasserstein (nW), GW (nGW) and FGW (nFGW) distances are defined as

$$\mathrm{n}\mathcal{W}(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{M}) = \min_{\boldsymbol{T} \in \mathcal{T}(\boldsymbol{p}, \boldsymbol{q})} \langle \boldsymbol{T}, \boldsymbol{M} \rangle_F / \frac{m}{n} \tag{4.53}$$

$$\mathrm{n}\mathcal{GW}(\boldsymbol{p},\boldsymbol{q},\boldsymbol{C}^s,\boldsymbol{C}^t)=\min_{\boldsymbol{T}\in\mathcal{T}(\boldsymbol{p},\boldsymbol{q})}\langle\boldsymbol{L}\otimes\boldsymbol{T},\boldsymbol{T}\rangle_F/\frac{m^2}{n^2} \tag{4.54}$$

$$\mathrm{n}\mathcal{FGW}_\alpha(\boldsymbol{p},\boldsymbol{q},\boldsymbol{M},\boldsymbol{C}^s,\boldsymbol{C}^t)=\min_{\boldsymbol{T}\in\mathcal{T}(\boldsymbol{p},\boldsymbol{q})}(1-\alpha)\langle\boldsymbol{T},\boldsymbol{M}\rangle_F/\frac{m}{n}+\alpha\langle\boldsymbol{L}\otimes\boldsymbol{T},\boldsymbol{T}\rangle_F/\frac{m^2}{n^2} \tag{4.55}$$

where all the relevant arguments $\boldsymbol{p}\in\mathbb{R}^n$, $\boldsymbol{q}\in\mathbb{R}^{m+1}$, $\boldsymbol{T}\in\mathbb{R}^{n\times(m+1)}$, $\boldsymbol{M}\in\mathbb{R}^{n\times(m+1)}$, and $\boldsymbol{L}\in\mathbb{R}^{n\times n\times(m+1)\times(m+1)}$, are defined in (4.4)-(4.7).

Proposition 5. nW distance, nGW distance, and nFGW distance are all valued between $[0,1]$, as long as α is chosen within $[0,1]$.

Proof of Proposition 5. The intuitive ideas are as follows. Since the dummy node has no contribution to the total cost, we only need to work with dimensions n and m. Intuitively, only a fraction of m/n nodes contribute to the feature cost and a fraction of m^2/n^2 edges contribute to the structure cost. By dividing the original distances by these two scalars, we are able to obtain the "cost per node" and "cost per edge". It can be shown that the maximum of the original Wasserstein distance is m/n, and the maximum of the original GW distance is m^2/n^2.

More specifically, in the extreme case we have a query that is "totally different" from the subgraph being matched. Then $\boldsymbol{M}$ is an all-one matrix except the last column, and $\boldsymbol{L}$ is an all-one tensor except when $j=m+1$ or $j'=m+1$. The total feature cost is summed up by all m query nodes, with each weighted by $1/n$,

$$\sum_{\substack{i=1,\dots,n\\ j=1,\dots,m+1}}\boldsymbol{M}_{i,j}\boldsymbol{T}_{i,j}=\sum_{\substack{i=1,\dots,n\\ j=1,\dots,m}}\boldsymbol{T}_{i,j}=\sum_{j=1,\dots,m}\boldsymbol{q}_j=\frac{m}{n} \tag{4.56}$$

Similarly, the total structure cost is summed up by all m^2 pairs of query nodes, with each weighted by $1/n^2$,

$$\begin{aligned}\sum_{\substack{i,i'=1,\dots,n\\ j,j'=1,\dots,m+1}}\boldsymbol{L}_{i,i',j,j'}\boldsymbol{T}_{i,j}\boldsymbol{T}_{i',j'}&=\sum_{\substack{i,i'=1,\dots,n\\ j,j'=1,\dots,m}}\boldsymbol{T}_{i,j}\boldsymbol{T}_{i',j'}\\ &=\sum_{\substack{i=1,\dots,n\\ j=1,\dots,m}}(\boldsymbol{T}_{i,j}\sum_{\substack{i'=1,\dots,n\\ j'=1,\dots,m}}\boldsymbol{T}_{i',j'})=\frac{m}{n}\sum_{\substack{i=1,\dots,n\\ j=1,\dots,m}}\boldsymbol{T}_{i,j}=\frac{m^2}{n^2}\end{aligned} \tag{4.57}$$

□

This nFGW distance is specifically for this subgraph matching problem and has the following benefits. To start with, since the two compositions are both within $[0,1]$, if we set $\alpha=0.5$, it means that we assign equal weights to the feature cost and structure cost. Besides, we ensure a fair similarity comparison across different nFGW distances with varying α, given that the distance values are constrained to the range $[0,1]$. Further, it sets a fair similarity comparison across different sizes of the test graph and the query graph. This standardization helps us directly evaluate the similarity between the query graph and the found subgraph, and facilitates the search for ε-suboptimal matching.

Compared with the original FGW distance (4.3), the scaling m/n and m^2/n^2 can be absorbed in the cost tensor $\boldsymbol{L}$ and matrix $\boldsymbol{M}$. In this way, the normalized FGW distance can still be a special case of FGW distance.

4.5 Sliding window and Wasserstein pruning

In practice, the use of FGW distance is limited by larger graph sizes. Firstly, the computational complexity of the tensor-matrix product scales polynomially with the size of the test graph, following an order of $\mathcal{O}(n^2m+m^2n)$, where n is the size of the test graph and m is the size of the query graph. Secondly, a larger graph typically contains more local minima, which causes the algorithm to become more susceptible and easily get trapped into one of the suboptimal solutions [57]. Given that in subgraph matching tasks the query graph is usually much smaller than the test graph, we can use this size discrepancy strategically to mitigate two negative effects.

We introduce two additional components, sliding window and Wasserstein pruning. Instead of directly comparing the query graph with the test graph, we iteratively compare the query graph with different subgraphs of the test graph. The subgraphs are built to ensure that at least one of all the subgraphs contains the optimal matching. The dummy node strategy is still adopted for the comparison with subgraphs. In each iteration, before calculating the nFGW distance, we first calculate the nW distance and use a feature cost threshold to filter out the subgraphs that do not meet the feature requirement.

Built on the SOT method, the new framework with these two additional components significantly improves the performance on sparse graphs, in terms of matching accuracy and query time. We refer to this new framework as the Sliding Subgraph Optimal Transport (SSOT). Ideally, the time complexity scales linearly in the test graph size n. We present the two components in detail in the following subsections.

4.5.1 Sliding window

The sliding window framework is shown in Figure 4.4 and Algorithm 3. We first define a *center* node of the query graph of size m. For each node $v_i \in \mathcal{V}^q, i = 1, ..., m$, we calculate the shortest path distances to every node in the graph (including node v_i itself), denoted as $d(v_i, v_j), j = 1, ..., m$. Then for each node v_i, we set $l_i = \max_j d(v_i, v_j)$ as the "maximum distance" of this node. The node with the smallest "maximum distance" is defined as the *center* node. Its corresponding l_i value is h, given by

$$h = \min_i \{\max_j d(v_i, v_j)\}. \tag{4.58}$$

Then the other nodes in the query graph have a maximum length of h from this center.

We iteratively pick every node in the test graph to be the *center* node. Around this center node, its h-hop neighborhood is created as a *sliding subgraph* of size n_s. The intention behind the definition (4.58) of h is to ensure that at least one of the sliding subgraphs "includes" the optimal matching of the query graph. At the same time, the size of this sliding subgraph is kept as small as possible. Ultimately, we build n sliding subgraphs to be compared with the query graph. In each iteration, we directly skip the cases that the sliding subgraph does not have enough nodes to include a matching of the query graph. When the sliding subgraph contains more nodes of the query graph, we add a dummy node to the query graph, and compute the nFGW distance as introduced in Section 4.2.

In the final step, we select the minimum nFGW distance computed between the query graph and the n sliding subgraphs to obtain final matching. We are also able to heuristically find the top-k subgraphs by ordering all the nFGW distances. The top-k results are not always guaranteed since it can happen that two highly similar subgraphs are within the same sliding subgraph.

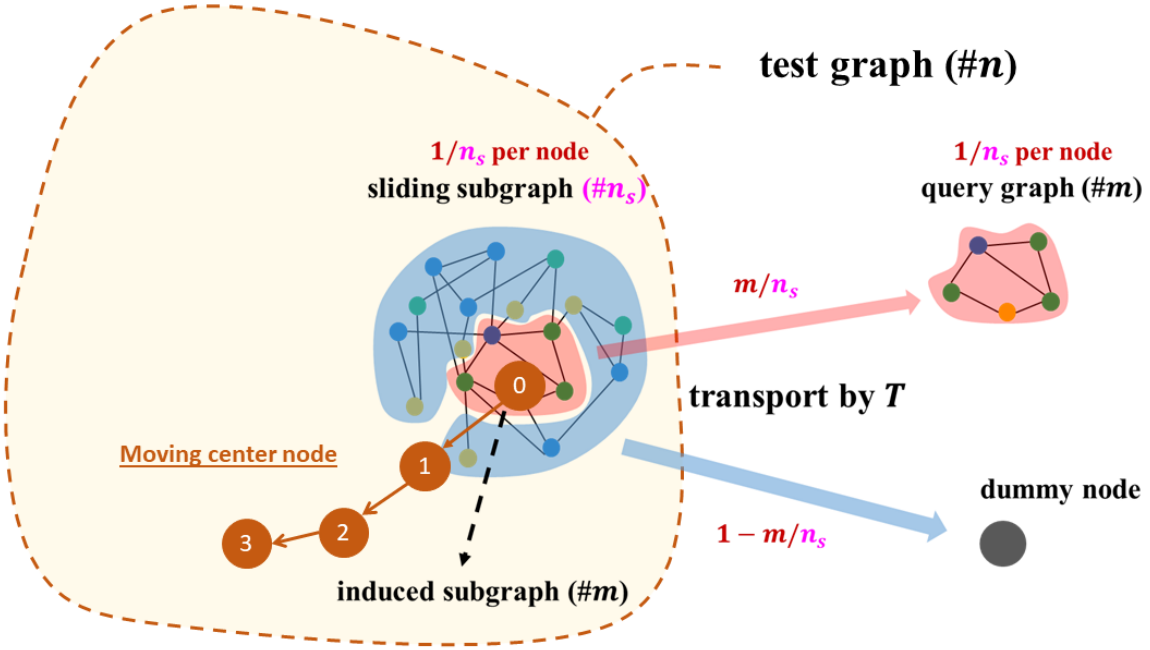


Figure 4.4: **Sliding Subgraph Optimal Transport (SSOT) - sliding window:** In each iteration, we compare the query graph with a sliding subgraph of size n_s. The subgraph (in red) should be matched to the query graph. The irrelevant part (in blue) should be matched to the dummy node.

4.5.2 Wasserstein pruning

The Wasserstein pruning is shown within the Algorithm 3. Before computing the nFGW distance between the query graph and each sliding subgraph, we compute the nW distance between them. If the nW distance is above the feature cost threshold $T^{\mathcal{W}}$, then this sliding subgraph is discarded and does not continue with the nFGW distance computation. This helps us filter out the sliding subgraph that does not meet the feature requirement in the first place. Thus it reduces the number of candidates in for nFGW distance and thus increases the chance of finding the optimal solution. Additionally, nW distance is a linear program, so the results of this feature filtering are always guaranteed. Moreover, it reduces the computational cost for richly-attributed graphs, since it reduces the chance of computing nFGW distance, which is more computationally expensive than nW distance. For exact matching, the threshold $T^{\mathcal{W}}$ is often set to a value close to zero. For noisy datasets and inexact matching, a positive threshold value ranging within $[0, 1]$ is determined based on specific requirements. The setting $T^{\mathcal{W}} = 1$ essentially means there is no Wasserstein pruning.

Algorithm 3: Sliding window and Wasserstein pruning

Input: $\mathcal{G}^s = (\mathcal{V}^s, \mathcal{E}^s)$, $\mathcal{G}^q = (\mathcal{V}^q, \mathcal{E}^q)$, k
Output: $\boldsymbol{T}^*$

for *node* v_i in $\mathcal{V}^q$ **do**
  Compute $l_i = \max_j d(v_i, v_j)$ as the "maximum distance";
end
Define $h = \min_i\{l_i\}$ as the shortest "maximum distance" among all l_i;
for *node* in $\mathcal{V}^s$ **do**
  Define $\mathcal{G}^{sub} = (\mathcal{V}^{sub}, \mathcal{E}^{sub})$ as the h-hop neighborhood of *node*;
  if $|\mathcal{V}^{sub}| < |\mathcal{V}^q|$ **then**
    continue
  end
  Add a dummy node to $\mathcal{G}^q$ and become $\mathcal{G}^t$;
  if $n\mathcal{W}(\mathcal{G}^{sub}, \mathcal{G}^t) > T^{\mathcal{W}}$ **then**
    continue
  end
  Compute $\mathrm{n}\mathcal{FGW}_\alpha(\mathcal{G}^{sub}, \mathcal{G}^t)$ using Algorithm 1 and Algorithm 2;
end
Return the top-k minimum nFGW distance and $\boldsymbol{T}^*$.

4.5.3 Complexity

The time complexity of our algorithm is closely tied to the test graph size n and its density. Specifically, for **sparse** graphs, the algorithm SSOT scales linearly at a low rate with the test graph size n. To obtain this result, we present the following steps. Before entering the sliding window loop, we need to first determine the parameter h **within the query graph** of size m. By using Dijkstra's algorithm, it requires a time complexity of $\mathcal{O}(m(m\log(m) + |\mathcal{E}^q|))$.

Within the sliding window framework, let us define D to be the maximum node degree of the test graph. **In each sliding iteration**, searching for the h-hop neighborhood sliding subgraph requires time and space complexity of $\mathcal{O}(D^h)$. The size of a sliding subgraph is also $n_s = \mathcal{O}(D^h)$. Suppose we have the threshold for Wasserstein pruning as $T^{\mathcal{W}} = 1$, then in each iteration, both the nW distance and nFGW distance are computed. The computation of nW and nFGW distances depends on the query graph size m, and the sliding subgraph size $n_s = \mathcal{O}(D^h)$. Let $K(n_s, m)$ describe the number of operations for solving the nFGW problem between a sliding subgraph of size n_s and a query graph of size m. Since the nW distance does not need to compute the tensor-matrix product and requires less complexity, the total time complexity to compute both distances is also $K(n_s, m)$.

To sum up all iterations, the overall time complexity is

$$\mathcal{O}\left(m(m\log(m) + |\mathcal{E}^q|) + n \cdot (D^h + K(D^h, m))\right).$$

This complexity goes linearly with size n. For a query graph of small size m and small h, and a sparse test graph with a small maximum node degree of D, the complexity increases at a slow rate.

To trace back, the complexity of SOT requires a time complexity of $\mathcal{O}(K(n,m))$, depending on the test graph size n and query graph size m. Since the tensor-matrix product with nFGW distance demands $\mathcal{O}(n^2m + nm^2)$, the total complexity will at least go quadratically with n, no matter what the density of the test graph is.

Additionally, for **richly-attributed** test graphs, the nFGW distance is computed only when the sliding subgraph satisfies the feature requirement set by the Wasserstein pruning. Therefore, we only need to compute the nFGW distance for a few iterations, which will further reduce the computational complexity.

4.6 Related work: partial optimal transport

The partial optimal transport problem considers a case when only a fraction of the mass is transported to the graph [14]. The objective function has the same formulation as the Wasserstein distance, the GW distance, or the FGW distance. However, the constraints for the transport matrix $\boldsymbol{T}$ is different,

$$\mathcal{T}(\boldsymbol{p},\boldsymbol{q}) = \left\{\boldsymbol{T} \in \mathbb{R}_+^{n\times m} \mid \boldsymbol{T}\mathbf{1}_m \leq \boldsymbol{p}, \boldsymbol{T}^\top \mathbf{1}_n \leq \boldsymbol{q}, \mathbf{1}_n^\top \boldsymbol{T}\mathbf{1}_m = s\right\}, \tag{4.59}$$

with a predefined fraction $0 \leq s \leq \min(\|\boldsymbol{p}\|_1, \|\boldsymbol{q}\|_1)$. For our subgraph matching problem, the transportation from the source graph to the query graph can be viewed as a special case of this partial optimal transport problem. That is, only a partial part of the total mass is transported to the query graph. The difference is that we have a fixed requirement of the column sums of the partial transport matrix $\tilde{\boldsymbol{T}}$ defined in Proposition 3. The constraints for the subgraph matching problem are

$$\tilde{\mathcal{T}}(\boldsymbol{p},\boldsymbol{q}) = \left\{\tilde{\boldsymbol{T}} \in \mathbb{R}_+^{n\times m} \mid \tilde{\boldsymbol{T}}\mathbf{1}_m \leq \boldsymbol{p}, \tilde{\boldsymbol{T}}^\top \mathbf{1}_n = \tilde{\boldsymbol{q}}, \mathbf{1}_n^\top \tilde{\boldsymbol{T}}\mathbf{1}_m = s\right\} \tag{4.60}$$

For our subgraph matching problem, $\boldsymbol{p}_i = 1/n$ for $i = 1, ..., n$, $\tilde{\boldsymbol{q}}_j = 1/n$ for $j = 1, ..., m$ and $s = m/n$.

Similar to what has been discussed in Section 2.2.3, the partial optimal transport problem can be solved directly with the feasible set $\mathcal{T}(\boldsymbol{p},\boldsymbol{q})$, or be solved by adding dummy nodes to remove the inequalities in the constraints. The study [14] on partial optimal transport used the dummy node method to compute the partial-Wasserstein distance by expending the cost matrix $\boldsymbol{M}$ with some positive or nul scalar ξ. However, the settings of their formulation do not allow them to use the same trick for expending the structure cost matrices $\boldsymbol{C}^s$ and $\boldsymbol{C}^q$ for partial-GW distance. Thus they solved partial-GW distance directly using the feasible set (4.59) by Frank-Wolfe algorithm. Within this algorithm, the linear minimization oracle is formulated as a partial-Wasserstein distance which is solved by adding dummy nodes.

Compared with the settings in our methods, the "trick" that the work on partial optimal transport [14] did not adopt is setting all the costs involved with the dummy node to zero, for both feature and structure parts. Based on what we discussed so far, our problem can still be properly solved by adding a dummy node by allocating zero costs to it. By adopting Propositions 3 and 4, we are also able to conduct computation reduction. The practice of assigning zero costs to the dummy node is also adopted in [12, 15].

5 Experiments

5.1 Introduction

This chapter details all the experiments conducted with the frameworks SOT and SSOT. In response to the questions we raised in Section 1.1, we begin with multiple real-world examples of chemical compounds and biomedical pathways in Section 5.3. Afterward, in Sections 5.5 and 5.6, we conduct the performance evaluations with synthetic datasets and real-world datasets. We investigate the performance of our algorithms on exact matching, approximate matching, and inexact matching. We also compare the performance of methods with existing works NeMa [23] and G-Finder [24].

For exact and approximate matching, we assume the datasets are clean and complete, with the ultimate goal of finding the exact matching. While the optimal solution is not guaranteed, we are also interested in whether the algorithms are able to find a satisfying ε-suboptimal solution. In Section 5.5, We use the Erdős–Rényi random graph model to evaluate the effects of different parameters of graphs, including the trade-off parameter, number of distinct feature values, sizes of test graph and query graph, and average node degree of the test graph. In Section 5.6, we use real-world datasets to evaluate the performance of different methods.

For inexact matching, there is no standard way for evaluation, since the problem is more application-specific. One may want to find a pattern more similar in terms of feature, or in terms of structure. The evaluation of the performance can also be highly customized by adopting various graph similarity measures. Based on these facts, we only consider the following settings for evaluation in Section 5.6. We conduct the inexact matching in noisy environments (in which some node feature values are in a noisy version) and assume that we know the ground truth of exact subgraph matching from the clean environment. We aim to investigate if the method can still find the original exact matching in noisy environments. In other words, we evaluate the robustness of the methods against noise.

5.2 Algorithmic settings

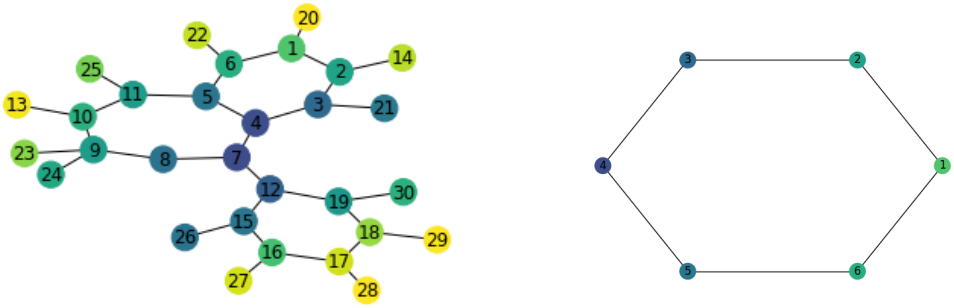
The Frank-Wolfe algorithm (Algorithm 1) is implemented with the Python Optimal Transport (POT) toolbox [48]. Within this toolbox, the linear minimization oracle (4.31) is solved by the well-developed EMD solver [58]. In our experiments, we primarily employ line search to find the step size $\gamma^{(k)}$. However, in rare cases where the line search default in POT fails to return an optimal $\gamma^{(k)}$, we default to $\gamma^{(k)} = 0.99$, aligning with the initial step size set in the POT toolbox.

5.3 Practical examples

This section shows examples of subgraph matching in chemical compounds and biological pathways. We illustrate the potential contributions of our algorithms to chemical and biomedical fields. We mainly focus on the **frequent subgraph matching** problem. Without a prior information, we assume the datasets are clean and complete. For simplicity, the experiments in this subsection are mainly conducted with the SOT framework unless specially indicated. The trade-off parameter α in nFGW distance is set to 0.5 as default.

5.3.1 Chemical Compound

In this section, we use the BZR dataset [1]. The dataset consists of 405 graphs of chemical compounds, with the average number of nodes as 35.75, and an average number of edges as 38.36. Each graph is endowed with a category label of "-1" or "1". The node feature values are 3-dimensional real-valued vectors. Features are assigned with real-valued vectors and the feature cost function is set to be the square L^2 norm distance. We are interested in detecting underlying common patterns among the graphs in this dataset. The query graph is constructed by taking out the induced subgraph form by the first six nodes of the first graph (with index 0 and label "-1") in the dataset (shown in Figure 5.1). We then use this query graph to query all the graphs in the dataset.



(a) The first graph in the BZR dataset (b) Query graph

Figure 5.1: **BZR dataset:** The first graph with label "-1" in the dataset (with index 0) and the query graph we obtain from this graph.

The experimental results inform that the exact matching only exists in the first graph (just the original graph we use to form the query graph) in the dataset. Nevertheless, we are able to obtain ε-suboptimal subgraph within other graphs. With ε =1e-03, we can find 9 graphs with label "-1" that contain a ε-suboptimal subgraph, while none of the graphs with label "1" contains such a subgraph. For the 9 graphs being found with a suboptimal subgraph, we have their nFGW distances with the query graph as follows

8.015e-13, 6.710e-4, 1.881e-4,
9.026e-4, 4.206e-4, 7.964e-5,

6.135e-4, 5.798e-4, 9.752e-4,

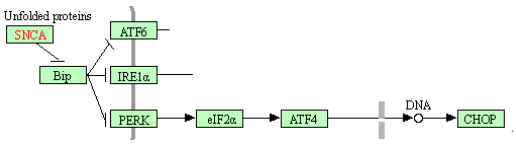
in which the first value 8.015e-13 belongs to the case when the exact matching is found in the frst graph (with index 0).

5.3.2 Biomedical pathways

With the same idea, biomedical diseases in the same category may share similar characteristics. By analyzing biomedical pathways with small signaling pathway structures, we are able to learn about similar evolution processes involved in different diseases. We present two categories of biomedical diseases, neurodegenerative disease and cancers.

We use the biological data from Kyoto Encyclopedia of Genes and Genome (KEGG) database [59]. The implementation is built on Python API KEGGutils [60]. This API converts the information of a pathway in KEGG into a graph in NetworkX format. The feature cost function is set as the Dirac function.

Parkinson and Alzheimer. To explore the similarities between Parkinson disease and Alzheimer disease, we query the unfolded protein response (UPR) signaling pathway (as shown in Figure 5.2) in both pathways. Parkinson disease yields with an exact matching, while Alzheimer disease can be found with a suboptimal matching. The matching results are shown in Figure 5.3, with circled red dashed lines indicating the matched subgraphs. The subgraph indicated in Figure 5.3a is exactly the UPR signaling pathway.

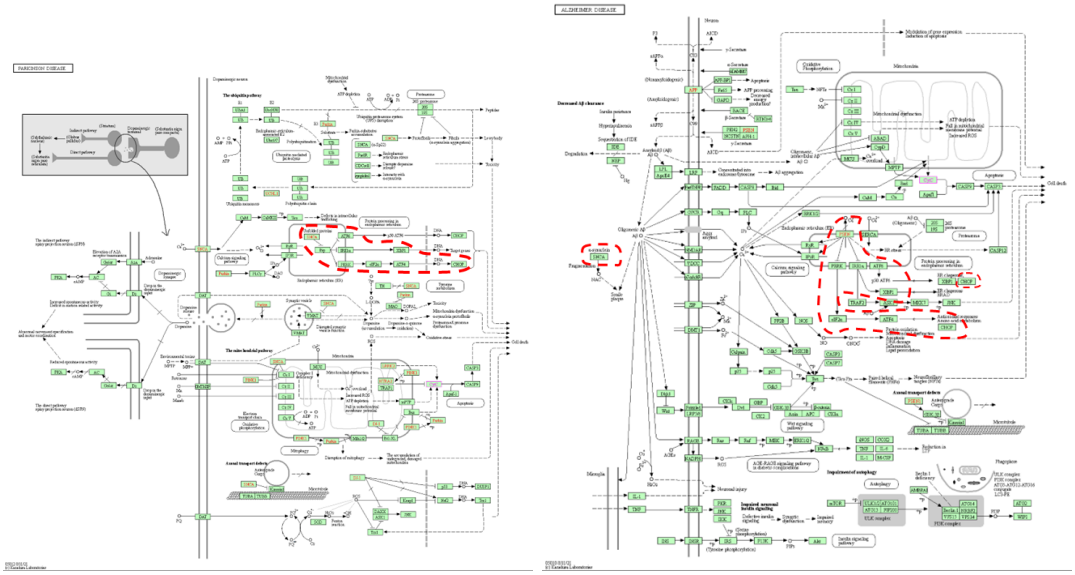


(a)

Figure 5.2: Unfolded protein response (UPR) signaling pathway

Cancers. The Ras-Raf-MEK-ERK signaling pathway (also known as MAPK/ERK pathway) is a fundamental signal pathway that exists in many disease pathways, especially cancers. The activation of the ERK pathway has been shown to increase the expression of HER2 in breast cancer cells, leading to increased signaling through the ERBB pathway and promoting cancer cell growth and survival.

We use the ERK pathway as the query graph to query the breast cancer pathway. With the SSOT framework and top-3 subgraph matching, three Ras-Raf-MEK-ERK pathways are able to be found in the breast cancer pathway. The matching results are shown in Figure 5.4, with circled red dashed lines indicating the subgraphs.



(a) Parkinson (b) Alzheimer

Figure 5.3: Neurodegenerative disease pathways with their found subgraphs.

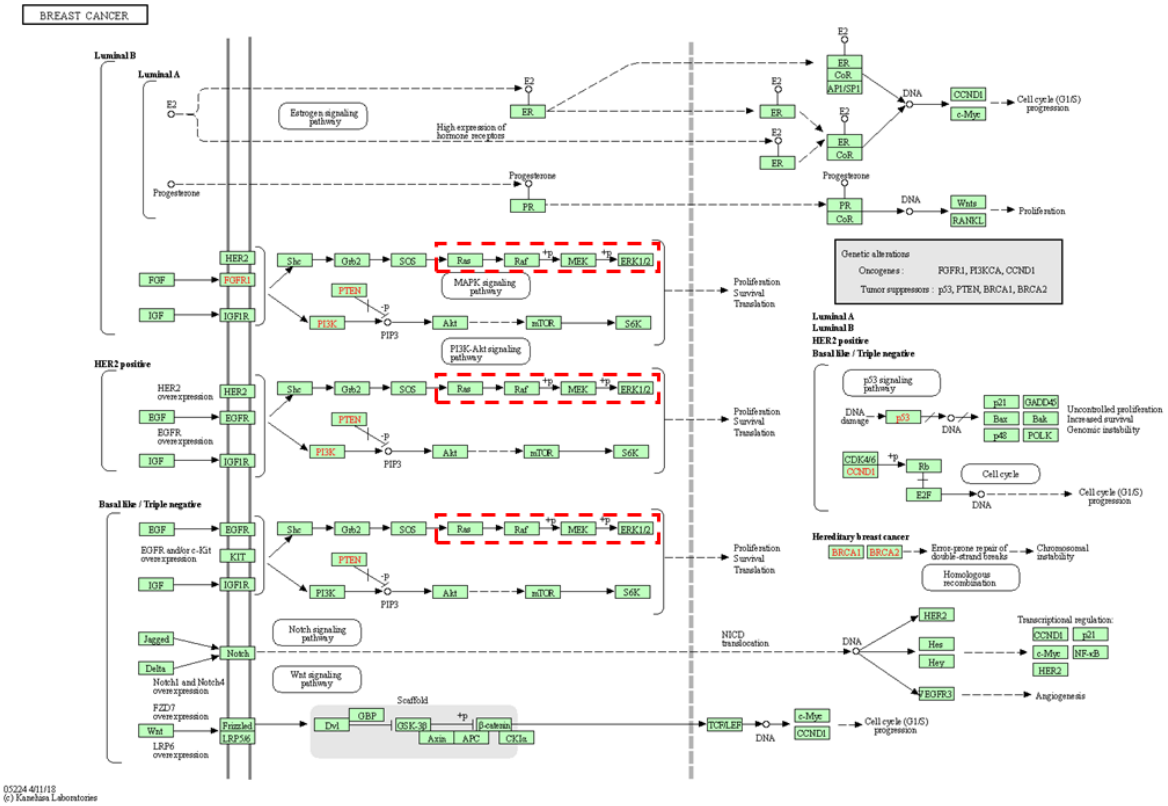


Figure 5.4: Breast cancer pathway with ERK signaling pathways.

5.4 Parameters of interests

In the remaining sections of this chapter, we aim to evaluate our algorithms on both synthetic datasets and real-world datasets. The following parameters may affect the performance of our methods:

- size of query graph m
- size of test graph n
- average node degree of the query graph d_Q
- average node degree of the test graph d_T
- number of distinct node feature values N^F
- trade-off parameter α in nFGW distance
- feature cost threshold $T^{\mathcal{W}}$ for Wasserstein pruning

5.5 Synthetic datasets

In this section, we create synthetic datasets to evaluate our own subgraph matching methods for exact matching and approximate matching. We aim to investigate how the parameters listed above affect the performance of the algorithms, in terms of query time and accuracy.

5.5.1 Dataset introduction

The synthetic dataset is created with Erdős–Rényi random graph model without self-loops. An (N, p) graph refers to a random graph of N nodes, and each pair of nodes is connected with probability p. The synthetic dataset consists of 1000 pairs of randomly created test and query graphs. In order to conduct exact matching, we ensure that there is at least one exact induced subgraph matching in the test graph. When building the graphs, we first create the query graph of size m, in which each pair of nodes is randomly connected with probability p_Q. Then the test graph is built by adding additional $n-m$ nodes to the periphery of this query graph. Each pair of nodes is randomly connected with probability p_T. To ensure the query graph is an induced subgraph, there are no additional edges between the m subgraph within the test graph compared with the original query graph.

In the following experiments, we want to evaluate the performance on different average node degrees d_Q and d_T, instead of the edge probability p_Q and p_T. The average node degree for a random graph (N, p) is $d = \frac{2 \cdot \#edges}{N} = (N-1)p$. If we want to investigate different sizes of graphs of the same degree d, the edge connection probability is set to be $p = d/(N-1)$.

5.5.2 Baseline

In the following, unless otherwise specified, graph parameters are set as default without further declaration: $n = 45$, $m = 5$, $N^F = 4$, $d_Q = 2$, $d_T = 10$. The node features are discrete-valued. The feature cost function is set as the Dirac function. The structure cost matrices $\boldsymbol{C}^s$ and $\boldsymbol{C}^t$ are assigned with adjacency matrices. The trade-off parameter is set as $\alpha = 0.5$ in nFGW distance to set equal importance for feature cost and structure cost.

5.5.3 Performance measures

The following measures are evaluated:

1. mean and 95% confidence interval of the nFGW distances
2. success rates of finding *an* exact matching; success rates that the objective values are zero; success rates of finding *an* ε-suboptimal matching
3. time of query (in seconds)

5.5.4 Parameter settings

The zero thresholds are set as 1×10^{-9} in practice, aligning with the stopping Tolerance $\Delta = 1 \times 10^{-9}$ in Frank-Wolfe algorithm (Algorithm 1).

In the SSOT framework, to obtain the **exact matching**, we first prune the graph features with zero Wasserstein distance, by setting $T^{\mathcal{W}} = 1 \times 10^{-9}$. The zero threshold for nFGW distance is also set as $\varepsilon = 1 \times 10^{-9}$. The actual exact matching rate is calculated by checking the found optimal transport matrix for each pair of query and test graphs. For **inexact matching**, we set $\varepsilon = 0.05$ to decide if an 0.05-suboptimal matching is found.

The mean and confidence interval are calculated by 1000 nFGW distances. The confidence level is calculated by the Bootstrap method.

5.5.5 Trade-off parameter α

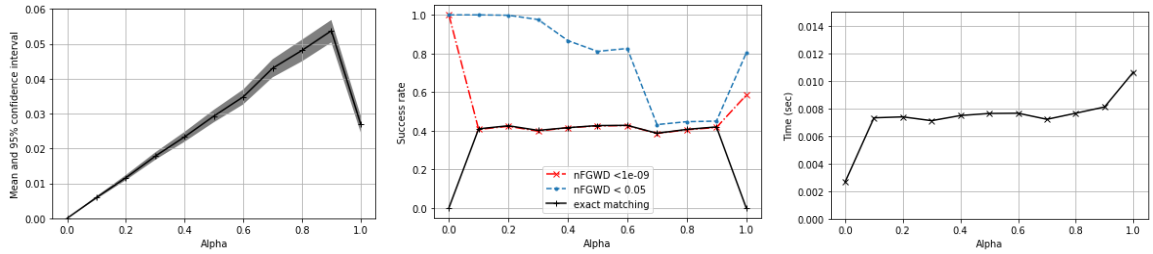
The performance versus the trade-off parameter α is shown in Figure 5.5 and Figure 5.6. The trade-off is varied with $\alpha \in \{0, 0.1, ...1.0\}$ in the experiments. In practice, the choice of α may largely depend on the goals of the applications. For a (sub)graph matching problem, it can be obtained via experiments on a "training dataset" [39], or by empirical experiments [23].

From Figures (b) we can learn that the nW ($\alpha = 0$) or the nGW distance ($\alpha = 1$) itself is not able to find the subgraph. When α is neither 0 nor 1, the choice of α does not affect the success rate of exact matching. Nevertheless, from the nFGW distances in Figures (a) and the "nFGWD < 0.05" suboptimal lines in Figures (b), we can still see the effect of α on both frameworks. The worse performance happens when α is set to be approximately $0.6 \sim 0.9$, among which the objective values are relatively high, and the 0.05-suboptimal success rates are relatively low. When α is a small value around $0.1 \sim 0.5$, more suboptimal solutions can be found. The performance reaches

the worst when α is around 0.9. Since when α is large, we put more emphasis on the nonconvex structure part, the algorithm can easily get trapped in a "bad" local minimum of relatively large value. We can also notice that larger α results in more fluctuations in the objective values. On the contrary, when α is small, we put more emphasis on the linear feature part, and then a "good" local minimum of relatively small value can be easily found.

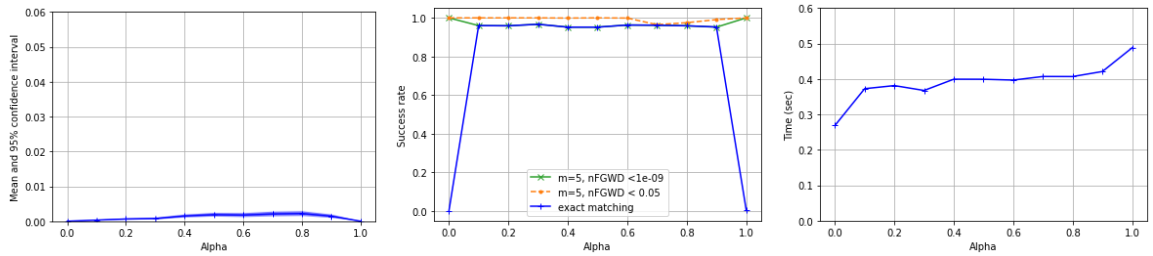
Additionally, during the range of $\alpha \in \{0.1, ...0.9\}$, the black line of exact matching matches exactly with the red line of "nFGWD $<$ 1e-09". This shows that for this synthetic dataset, the zero threshold for the nFGW distance is indeed a good indication of exact matching.

In terms of time, there is no big difference when α is neither 0 nor 1. When $\alpha = 0$, it takes the least time; when $\alpha = 1$, it takes the longest time. Compared with the SOT framework, the SSOT framework achieves lower objective values, higher success rates, and also longer query time in all experiments.



(a) (b) (c)

Figure 5.5: Objective values, success rates, and query times versus trade-off parameter with SOT.



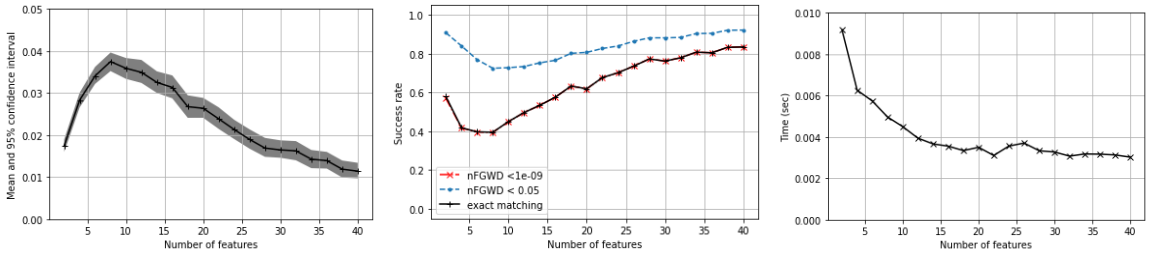
(a) (b) (c)

Figure 5.6: Objective values, success rates, and query times versus trade-off parameter with SSOT.

5.5.6 Number of distinct feature values

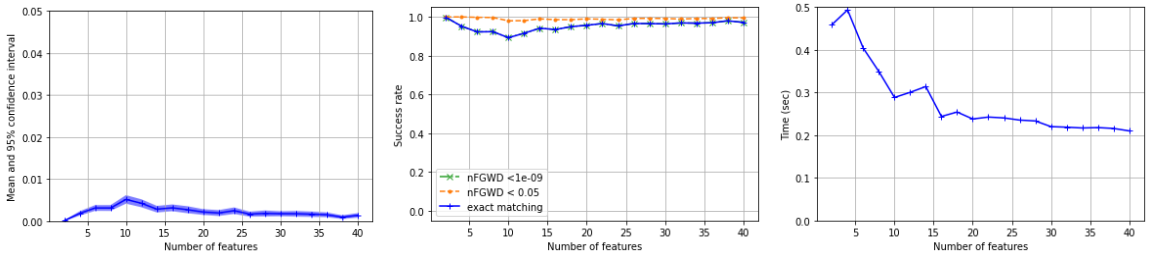
We compare the performance for $N^F \in \{2, 4, ..., 38, 40\}$. Results are shown in Figure 5.7 and Figure 5.8.

For both frameworks, the success rates first decrease and then gradually increase. The objective values have the corresponding reversed evolution. When the number of features is very small (smaller than the size of the query graph), there exist more exact matches and ε-suboptimal matches in the test graph, so the success rate is relatively high. When there are slightly more distinct features (around the size of the query graph), the number of optimal solutions becomes less. Afterwards, when there are much more distinct features (much larger than the query size), the nodes in the test graph become more distinct and create fewer local minima. In extreme cases, the query graph can be detected in the test graph only with feature information and without structure information. The time gradually decreases when there are more distinct feature values. Thus increasing the variety of features helps accelerate the optimization process.



(a) (b) (c)

Figure 5.7: Objective values, success rates, and query times versus number of features with SOT.



(a) (b) (c)

Figure 5.8: Objective values, success rates, and query times versus number of features with SSOT.

5.5.7 Sizes

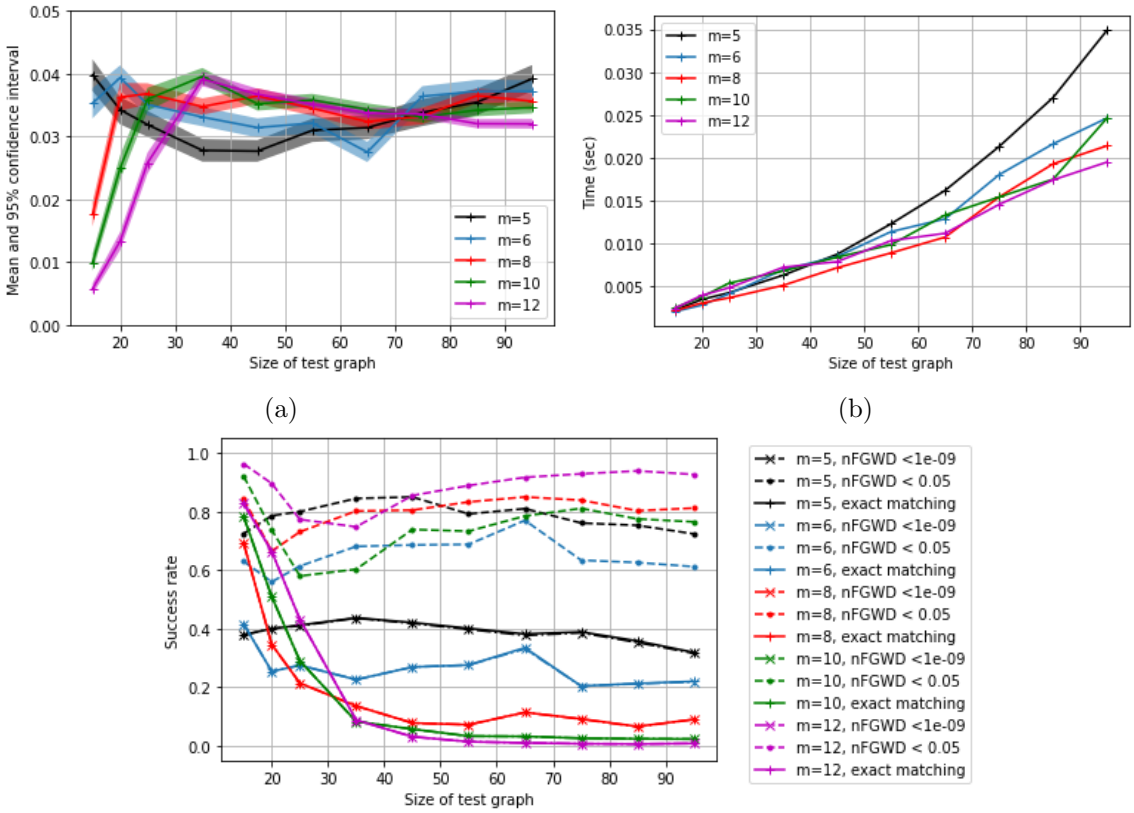
We research how the sizes of test graph n and query graph m will affect the performance. For each m, we consider $n \in \{15, 20, 25, 35, 45, 55, 65, 75, 85, 95\}$. The experiments are repeated with $m \in \{5, 6, 8, 10, 12\}$. For each pair of n and m, we set $p_T = d_T/(n-1)$ and $p_Q = d_Q/(m-1)$ to construct the graphs. Results are shown in Figure 5.9 and Figure 5.10.

For the **SOT** results in Figure 5.9, one may first notice the appearance of "tops" in Figure 5.9a, and "bottoms" (in dashed lines) in Figure 5.9c. The "top" and "bottom" of the line $m = 5$ are not visible in the figure due to space limitation, but one can deduce that they should appear somewhere when $0 < n < 15$. With the increases of size n, the objective values first increase and then decrease. The success rates of the 0.05-suboptimal solutions go in the opposite direction correspondingly. It finally gets close to 1.0 when n is large enough. While the success rates of exact matching (in solid line) decrease continuously.

In terms of the query time, it increases with size n. Besides, the smaller the size of the query m, the longer the query will take. We also notice from the mean and 95% confidence level that with smaller query size m, the cost values have more fluctuations. With a larger query size, the cost values are more concentrated.

Several possible reasons are as follows. When the size of the test graph n continuously increases, there exists more subgraphs that are similar to the query. Therefore for exact matching, it becomes easier to be trapped in these local minimums without finding the optimal solution. This reason applies similarly to the early stage of finding the 0.05-suboptimal solutions. When the test graph is much larger, there exists more local minimums that are satisfying enough. However, this is not the case for exact matching, since the number of exact matches is generally small even when the test graph is much larger.

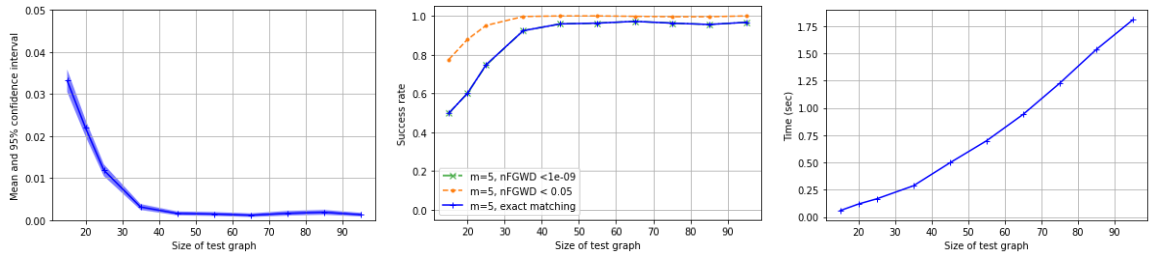
The results for **SSOT** with $m = 5$ is shown in Figure 5.10. Compared with the $m = 5$ lines in the SOT, it achieves lower objective values, and higher success rates, at the expense of longer query times.



(a) (b)

(c)

Figure 5.9: Objective values, success rates, and query times versus graph sizes with SOT.



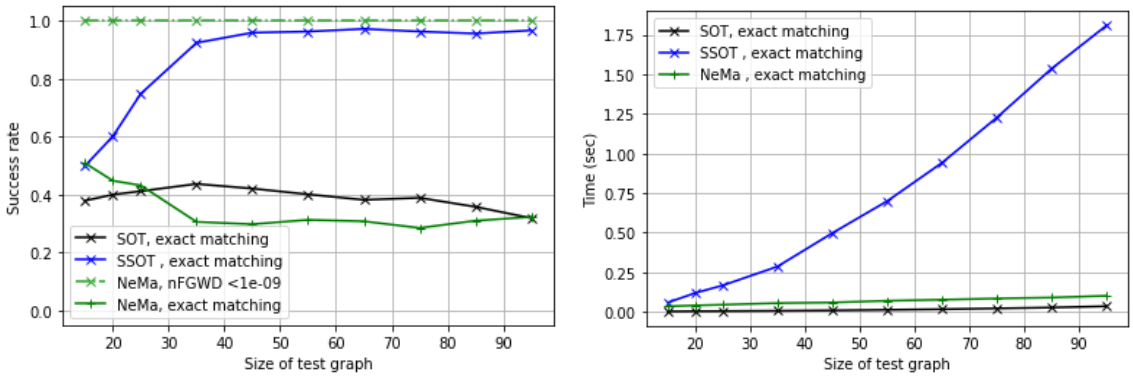
(a) (b) (c)

Figure 5.10: Objective values, success rates, and query times versus graph sizes with SSOT.

Comparison with NeMa. We compare the performance of NeMa with SOT and SSOT in terms of success rate and query time. The threshold of the individual feature cost of NeMa is also set to "zero" as $T^{\mathcal{F}} = T^{\mathcal{W}} = 1 \times 10^{-9}$.

For NeMa, as we can see from the green lines (both solid and dashed ones), whether the objective value is zero is not a good indication of whether an induced subgraph is found (This is also illustrated in the original paper [23]). The induced subgraph is not always found, but the objective value of NeMa is always zero. In terms of success rate and time, NeMa is slower than the SOT but faster than the SSOT. However, the success rate is even lower than SOT.

Two situations make the NeMa achieve lower success rate. 1) the mapping is not always a one-to-one mapping, even though the cost value is zero; 2) NeMa is not designed for searching induced subgraph. NeMa can potentially fall into a general subgraph without finding the induced subgraph. While for our methods, general subgraphs do not disturb the search of induced sugraphs.



(a) (b)

Figure 5.11: Success rates, and query times versus graph sizes with SOT, SSOT, and NeMa.

Performance for large sparse richly-attributed graphs. In this part, we evaluate the performance of graphs with much larger sizes. We assign the size of the test graph with $n \in \{50, 100, 1000, 3000, 5000, 7000, 10000\}$, and the number of distinct feature values as $N^F = 20$. The query size is set to $m = 5$. The degree values are set to $d_Q = 2$ and $d_T = 3$. The results are shown in Figure 5.12. The experiments are only run with 10 repetitions due to the large graph size and long running time.

The time complexity of SSOT for sparse graphs goes linearly with n as discussed in Section 4.5.3. The algorithms SOT and SSOT spend the same time at around $n = 3000$. Afterward, the SSOT is much faster than SOT.

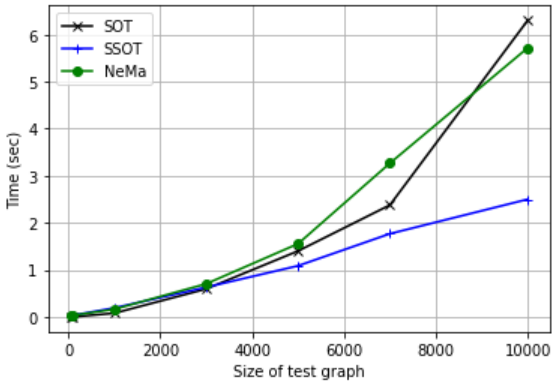


Figure 5.12: Time comparison versus size of test graph.

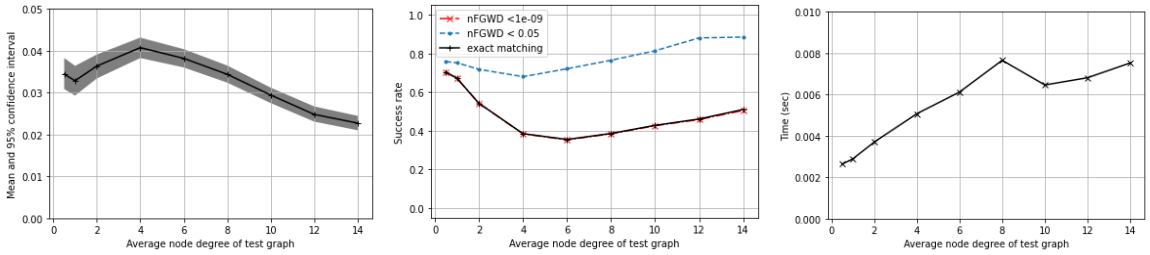
5.5.8 Average node degree of test graph

We investigate the performance of different average node degree $d_T \in \{0.5, 1, 2, 4, 6, ..., 14\}$ for the test graph.

The evolution through the increase of degree is similar to the increase of the size of the test graph, except for the success rates of exact matching. When the graph is extremely sparse, it is easy to find the solution, since there are few exact matches and suboptimal solutions. When it becomes a bit denser, it becomes easier to be trapped in a local minimum. However, when the graph becomes much denser, there exists more exact matching and satisfying suboptimal solutions. Thus the success rate goes up again.

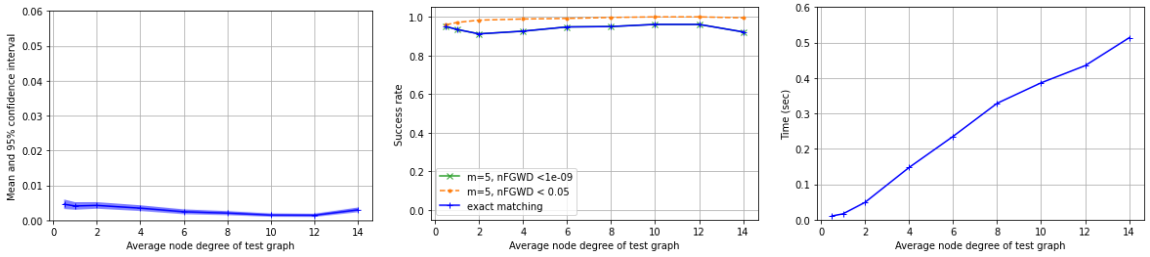
In the prior section, we observed that increasing the size n continuously decreased the success rates of exact matching. However, this trend doesn't hold when we increase the average node degree d_T of the test graph. When the node degree is relatively large, increasing the node degree increases the number of exact matchings, while simply increasing the size n does not have the same effect.

The query time increases continuously. In each iteration, the time is mainly for constructing sliding subgraphs and computing the nFGW distance. The time for constructing the sliding subgraphs is theoretically polynomial to d_T and thus the total time increases with d_T.



(a) (b) (c)

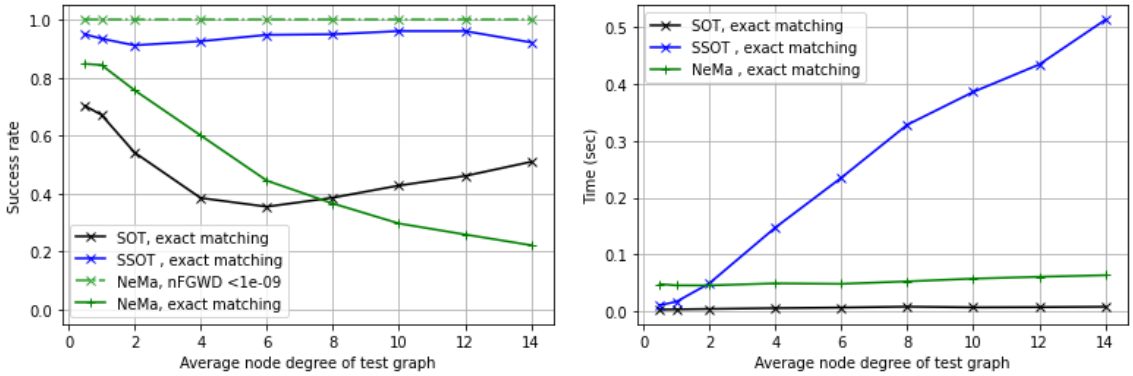
Figure 5.13: Objective values, success rates, and query times versus average node degree of test graph with SOT.



(a) (b) (c)

Figure 5.14: Objective values, success rates, and query times versus average node degree of the test graph with SSOT.

Comparison with NeMa. The performance of NeMa can be approximately considered as in between SOT and SSOT, with a better performance with small node degrees than large node degrees.



(a) (b)

Figure 5.15: Success rates, and query times versus average node degrees with SOT, SSOT, and NeMa.

5.6 Real-world datasets

In this section, we examine the performance of our methods on real-world datasets for top-1 matching, in both clean and noisy environments. We also compare the performance of our methods with the existing methods NeMa [23] and G-Finder [24]. We explore the performance of these methods in terms of accuracy and time, and compare the different performances in clean and noisy environments.

5.6.1 Dataset introduction

Test graphs. We use the datasets BZR and FIRSTMM_DB. The BZR dataset has been introduced in Section 5.3.1. The FIRSTMM_DB dataset consists of 41 graphs of 3D point cloud data, with the average number of nodes as 1377.27, and an average number of edges as 3074.10 [1]. The node feature values are real-valued scalars.

Query graphs. For each test graph in the dataset, we generate 10 connected induced subgraphs of 6 nodes by the Breadth-First Search (BFS) algorithm. To create a subgraph, a start node is first randomly picked within the test graph. Then the subgraph is expanded by adding the neighbors of the current visiting node in a bread-first manner until the subgraph reaches the desired size. These subgraphs are used as the query graphs for the corresponding test graphs.

5.6.2 NeMa and G-Finder

NeMa and G-Finder are indexed-based methods which have been briefly introduced in Section 2.2.6.3. In NeMa, the indexing step pre-computes the neighborhood vectors and stores the vectors in the index. Before the optimization process begins, it first computes a "candidate list" for each query node with a feature cost threshold $T^{\mathcal{F}}$. This threshold ensures that any individual node in the test graph that is matched to the query node falls within a specified similarity range. Then the query process greedily searches all the possible matches within the candidate sets with iterative algorithms, and finds the match that minimizes the cost function. In G-Finder, only nodes with the same feature value are allowed to be matched with a query node. A node that is matched with a query node with a different feature value is considered as a "node mismatch" in the cost function.

On the contrary, in our SSOT method, we use the Wasserstein distance to consider the overall label difference of all node labels of the query. We do not require every node feature in the found subgraph to be exactly the same or similar to one of the query nodes. We instead consider a "candidate list" for the whole query graph with a threshold $T^{\mathcal{W}}$ (as defined in Section 4.5.2).

Regarding the time comparison with these index-based methods, we calculate the total time, including both the indexing phase and the query phase.

5.6.3 Implementation details

Matching ratio. We consider the ratio of correctly found nodes within all the query nodes. The ground truths are known by the node indices of the query graphs taken out from the original test graph. Since the graphs in datasets BZR and FIRSTMM_DB are all richly-attributed with real values/vectors, two different node feature values can rarely be the same. Therefore it is rare that there exist multiple exact matchings in a test graph. There are some rare cases in FIRSTMM_DB dataset that the exact matching is not unique, but these rare cases only have little effect on the overall performance. Therefore we assume that there is only one exact matching in each graph.

Noisy environments. To introduce the noisy environments, we randomly pick one node in the query graph and add zero-mean Gaussian noise with a standard deviation of 0.1 to its node feature value. Since we add a Gaussian noise without being too strong and only to one of the query nodes, we still assume that there is only one optimal (inexact) matching in the test graph (not an exact matching in noisy cases). We compare the found subgraph with the ground truth to calculate the ratio of correctly matched query nodes. We are interested in whether the method is still able to find the original optimal matching in the noisy environments.

Feature costs. The feature cost function is set to be normalized square L^2 norm distance for our methods and NeMa.

NeMa. We use the Python implementation fornax to conduct our experiments. Since fornax is implemented with the backup of an SQLite database, the time of database manipulation should not be included. The total time is measured by the summation of time of the following three steps: candidate selection, indexing (compute the neighborhood vectors for all nodes in the test graph; eliminate isolated candidates for query nodes), and query (optimization).

Note that the cost values in fornax are defined "reversely" from the original definition. When an exact matching is found, the cost value is 1 instead of 0. This applies to the same for its feature cost and structure cost. For a fair comparison with other methods, the trade-off parameter is set to be $\lambda = 0.5$. Within the definition of "neighborhood vectorization", which encodes the neighborhood information of each node, the hopping distance is set to 1, to be matched to the adjacency matrix we used in our algorithms. The propagation factor α is set to be 0.3 as the default in fornax. Self-loops are not allowed in fornax, in which the program is not able to proceed. We mark the matching ratio to be zero if we come across this situation.

G-Finder. Since G-Finder is implemented in C++, we do not compare its processing time with other methods. We only compare its performance in terms of accuracy.

One drawback of G-Finder is that it does not consider the cases when the query graph is a line graph, where the program is not able to find a root node within the query and will fail completely (see details below). Before the official comparison with G-Finder, we modified its "root selection step" to include this case. The details are as below.

- The original algorithm first decomposes the query graph into a core structure and a forest structure. The core structure is built by iteratively removing all degree-one nodes from the query graph and the remaining subgraph is the core structure [24, 61]. Thus in the core structure, each node has at least 2 neighbors. G-Finder requires that the root of the query should be chosen from a node that belongs to the core structure of the query. This constraint is not suitable for line graphs, since the core structure of a line graph is empty by definition. Therefore, in order to find a root for a line query graph, we modified the algorithm to include these special cases. If the query graph is a line graph, the root node is searched among *all* query nodes with the original metric $\frac{|C(u)|}{\deg(u)}$ defined in the paper, where $|C(u)|$ is the size of the candidate set of query node u, and $\deg(u)$ is the degree of u in the query graph.

In the original paper, the cost function incorporates tolerances for discrepancies within the matching process. The weights for the number of missing query nodes (w_1), missing query edges (w_2), and intermediate nodes (w_3) are all set equally to 1, which are the same as the default in the paper.

5.6.4 Results and discussions

This section presents the performance of all the methods on BZR and FIRSTMM_DB datasets under both clean and noisy environments. The results are shown by the ratio of correctly matched nodes (**Matching ratio**) and query time (**Time**), in Tables 5.1, 5.2, 5.3, and 5.4.

For the SSOT and NeMa, the query time and accuracy are correspondingly changed by assigning different thresholds $T^{\mathcal{W}}$ and $T^{\mathcal{F}}$ to Wasserstein pruning and label cost function (as introduced in Sections 4.5.2 and 5.6.2), respectively. These accuracy-time trade-offs in noisy environments are graphically represented in Figure 5.16. For SOT and G-Finder, they do not have tunable parameters and we only have one experiment result for each clean/noisy dataset. In our noisy settings, G-Finder is not able to perform normally due to its inherent limitations when dealing with noise. Specifically, when noise is introduced to a query node, it results in a new feature value that does not exist in the test graph, which is not permitted in G-Finder.

From the results of SSOT and NeMa in all tables, the query time increases gradually with the rise of thresholds $T^{\mathcal{W}}$ or $T^{\mathcal{F}}$. All the methods achieve satisfying results on BZR dataset in both clean and noisy environments. However, when handling the larger FIRSTMM_DB graphs, all methods experienced longer query time and reduced accuracy.

In clean environments (Table 5.1 and 5.2), the SOT method stands out with the BZR dataset. Conversely, the SSOT shows the best performance with the FIRSTMM_DB dataset, when the threshold is set as $T^{\mathcal{W}} = 1 \times 10^{-9}$. This suggests the benefits of SSOT over SOT on large graphs. From FIRSTMM_DB, we can

Table 5.1: Clean dataset of BZR

SOT							
Matching ratio	1.0						
Time	0.005						
G-Finder							
Matching ratio	0.9995						
	$T^{\mathcal{W}}$ or $T^{\mathcal{F}}$						
	1×10^{-9}	1×10^{-5}	1×10^{-2}	5×10^{-2}	1×10^{-1}	5×10^{-1}	1
SSOT ($T^{\mathcal{W}}$)							
Matching ratio	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Time	0.008	0.008	0.008	0.009	0.008	0.026	0.088
NeMa ($T^{\mathcal{F}}$)							
Matching ratio	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Time	0.028	0.027	0.027	0.031	0.030	0.028	0.132

Table 5.2: Clean dataset of FIRSTMM_DB

SOT					
Matching ratio	0.835				
Time	4.296				
G-Finder					
Matching ratio	0.872				
	$T^{\mathcal{W}}$ or $T^{\mathcal{F}}$				
	1×10^{-9}	1×10^{-5}	1×10^{-2}	5×10^{-2}	1×10^{-1}
SSOT ($T^{\mathcal{W}}$)					
Matching ratio	0.836	0.827	0.821	0.821	0.821
Time	0.252	1.923	6.669	7.834	8.354
NeMa ($T^{\mathcal{F}}$)					
Matching ratio	0.772	0.719	0.719	0.719	0.719
Time	0.384	1.430	5.455	7.023	8.585

also observe that both SSOT and NeMa decrease the query time at lower threshold settings of $T^{\mathcal{W}}$ and $T^{\mathcal{F}}$. These results indicate that pre-computing the candidate sets can both enhance the matching precision and reduce the processing time. The effects of the pre-computing are mitigated when $T^{\mathcal{W}}$ and $T^{\mathcal{F}}$ are set to be relatively high.

In noisy environments (Tables 5.3 and 5.4), regarding our results, we can first observe that SOT achieves the best performance among all the methods. For SSOT and NeMa, the choice of the thresholds $T^{\mathcal{W}}$ and $T^{\mathcal{F}}$ becomes more complex. On one hand, the thresholds should be sufficiently low to minimize the candidate set size, and thus reduce the query time and improve the matching accuracy. On the other hand, the thresholds should also be high enough to ensure the inclusion of the objective subgraph within the candidate set, given that the noise introduces a positive, non-zero

nW distance even for the correct subgraphs. Therefore, higher thresholds always bring longer query time, but whether it will bring higher or lower accuracy depends on specific situations.

For **BZR** dataset, both SSOT and NeMa achieve higher accuracy with increased thresholds within the range from 1×10^{-9} to 1. This improvement suggests that a larger candidate set is more likely to contain the correct subgraph. Even with a threshold of 1, these methods can accurately identify the subgraph, implying that the computation of candidate sets is less critical for smaller graphs in the presence of noise.

With the **FIRSTMM_DB** dataset, a more complex pattern emerges. The SSOT's matching ratio improves with the threshold up to a point (around $T^{\mathcal{W}} = 1 \times 10^{-2}$) before a slight decline. This implies a threshold around 1×10^{-2} may be large enough to include the objective subgraph, and also small enough to have a small candidate set size. In contrast, the accuracy for NeMa shows a slight but continuous decline. The behaviors of NeMa can be potentially explained as follows. We can notice the overall low matching ratios for the FIRSTMM_DB dataset (lower than the ratio of clean nodes in the query as 5/6). Setting a lower threshold can be potentially beneficial for NeMa, since a smaller candidate set for the clean nodes in the query graph makes it easier to pinpoint the correct matches without being overly influenced by the overall cost associated with all the query nodes. Thus the accuracy of NeMa can be potentially increased by setting a lower threshold if accuracy is already low. However, this is not the case for SSOT, since the nodes are not assessed separately in the first place, setting an extremely low threshold will only filter out the objective subgraph that we intend for.

The ratio-time trade-off plots for noisy cases of two datasets are shown in Figure 5.16. From the plot of BZR dataset, we can conclude that SSOT achieves better rate-time performance than NeMa. The performance of SOT is even better. For FIRSTMM_DB dataset, two methods have their own benefits in different situations. With a high feature cost threshold, either $T^{\mathcal{W}}$ or $T^{\mathcal{F}}$, the programs require a relatively high query time, and SSOT achieves higher accuracy. With a low feature cost threshold, the programs require a relatively low query time, and NeMa achieves higher accuracy.

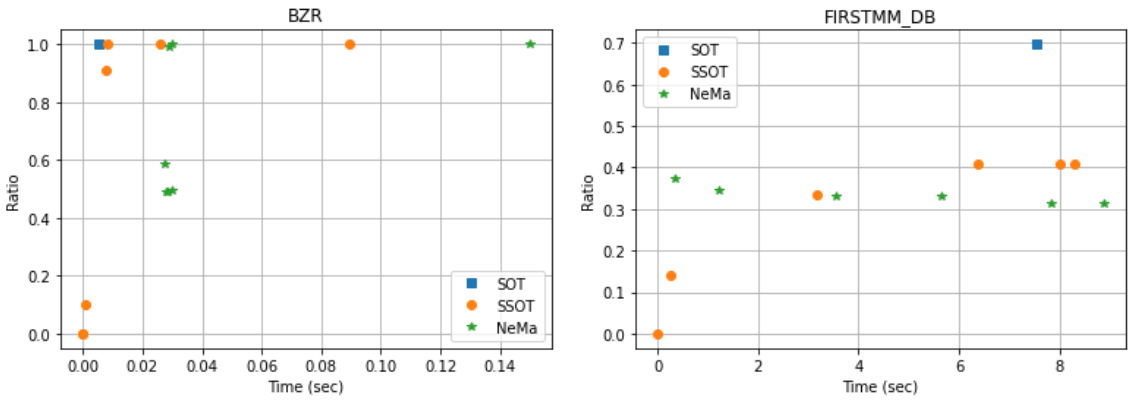


Figure 5.16: Matching ratio versus Time of different methods with noise std=0.1 for BZR and FIRSTMM_DB .

Table 5.3: Feature Noise std = 0.1 for BZR

SOT							
Matching ratio	1.0						
Time	0.005						
G-Finder							
Matching ratio	Fail						
	$T^{\mathcal{W}}$ or $T^{\mathcal{F}}$						
	1×10^{-9}	1×10^{-5}	1×10^{-3}	1×10^{-2}	1×10^{-1}	5×10^{-1}	1
SSOT ($T^{\mathcal{W}}$)							
Matching ratio	0	0	0.101	0.908	1.0	1.0	1.0
Time	2.404e-05	2.876e-05	0.001	0.008	0.008	0.026	0.089
NeMa ($T^{\mathcal{F}}$)							
Matching ratio	0.492	0.492	0.495	0.588	0.993	1.0	1.0
Time	0.028	0.028	0.030	0.027	0.029	0.030	0.150

Table 5.4: Feature Noise std = 0.1 for FIRSTMM_DB

SOT						
Matching ratio	0.697					
Time	7.530					
G-Finder						
Matching ratio	Fail					
	$T^{\mathcal{W}}$ or $T^{\mathcal{F}}$					
	1×10^{-9}	1×10^{-5}	1×10^{-3}	1×10^{-2}	5×10^{-2}	1×10^{-1}
SSOT ($T^{\mathcal{W}}$)						
Matching ratio	0	0.140	0.334	0.409	0.407	0.407
Time	11.071e-05	0.245	3.162	6.363	8.023	8.314
NeMa ($T^{\mathcal{F}}$)						
Matching ratio	0.372	0.345	0.331	0.332	0.313	0.313
Time	0.328	1.230	3.555	5.646	7.825	8.874

5.7 Discussions

The results of the performance evaluations on synthetic datasets and real-world datasets lead to several key observations as follows.

The trade-off parameter is always set to $\alpha = 0.5$ as default in our experiments, which can be generally adopted in various applications if no a prior information is available. Nevertheless, if one only wants to conduct exact matching, as we can learn from Section 5.5.5, this parameter can be selected arbitrarily from the open interval $(0, 1)$, excluding the values of 0 and 1. For approximate matching and inexact matching, this parameter can be tailored to specific practical interests within $[0, 1]$, allowing for a preference towards either feature or structure information. However, due to the non-convex nature of the structure component, an emphasis on structural information may

encounter more difficulties in finding a satisfying subgraph, as compared to an emphasis on feature information.

Our frameworks jointly consider feature information and structure information. It is also worth mentioning that endowing a greater number of distinct feature values generally tends to enhance performance. The SSOT method stands out with its Wasserstein pruning, or alternatively, the SOT method can be employed with a potential bias towards feature information by setting α within $[0, 0.5]$.

The SOT framework consistently yields satisfying results for small graphs. The SSOT framework, on the other hand, significantly bolsters performance for large graphs. Also in SSOT, if the dataset is clean and complete, one can always use a lower feature cost threshold to improve the matching accuracy and reduce the query time. We can also conclude that with small noise in the node features, the SOT method demonstrates greater resilience compared to SSOT, NeMa, and G-Finder. The selection of thresholds in noisy environments should generally be positive, although the ideal value may differ across datasets. Pre-experimental tuning is recommended to achieve the best possible matching outcomes in practical settings.

Conclusion and Future Directions

6

6.1 Conclusion

In this thesis, we have presented two subgraph matching frameworks: SOT and SSOT, both founded on the principles of optimal transport and utilizing the Fused Gromov-Wasserstein (FGW) distance. Our work is built at the intersection of different domains with newly developed techniques.

- We formulated the subgraph matching problem with the idea of optimal transport and adopted the FGW distance. With the adoption of Frank-Wolfe algorithm, the program is computationally efficient and is easy to implement with existing optimization solvers. Moreover, the FGW distance is supported by solid mathematical foundations.
- We refined the classical graph matching QAP's by introducing a dummy node with zero-cost allocations to tailor for this subgraph matching problem. This innovation led us to propose the normalized FGW distance to adapt to various real-world applications and enhance performance evaluation. We also achieved a substantial computation reduction by isolating the dummy node and directly modifying the matrix of the tensor-matrix product.
- We improved our algorithms with insights from index-based methods in graph database queries and introduced the sliding window framework and Wasserstein pruning. These two components significantly improve query efficiency and accuracy, offering practical advantages in an engineering context.

The experimental results validate the efficacy of both the SOT and the SSOT frameworks. The SOT framework is particularly effective for smaller graphs, while the SSOT framework is beneficial for larger graphs. Furthermore, both methods exhibit robust adaptability in noisy environments, outperforming existing methods such as NeMa and G-Finder. Lastly, practical applications presented in the chemical and biomedical fields highlight the potential contributions of our work to these areas.

6.2 Future directions

6.2.1 Matching for subgraphs of different sizes

The problem we solved in this thesis is about matching a subgraph of the same size as the query. Note that using the FGW distance allows us to search for a subgraph of a different size m', such that $m' \neq m$. To solve this problem, we can simply modify the mass of each query node to be $\frac{m'}{nm}$. The modified version of the "dummy node strategy"

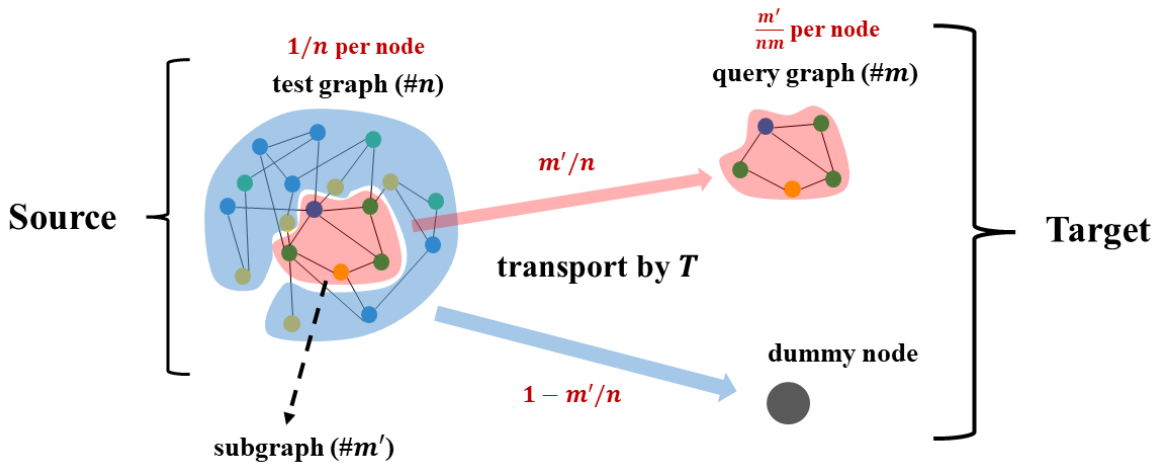
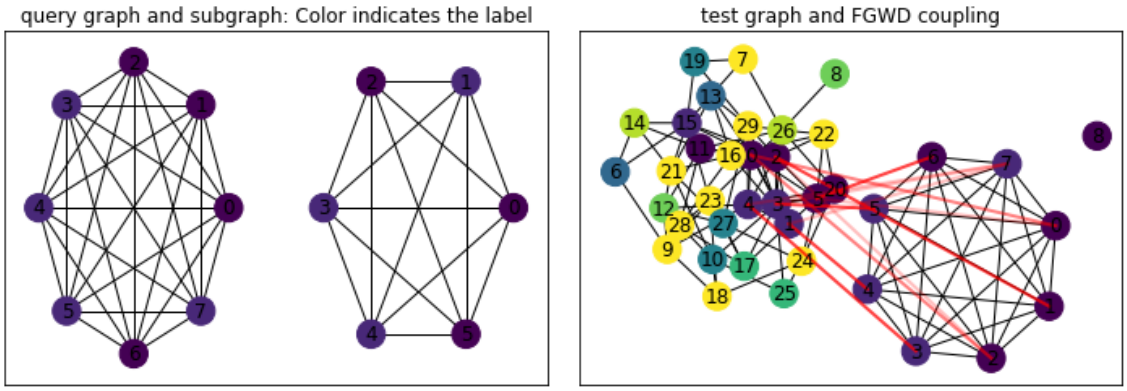


Figure 6.1: **Subgraph matching algorithm (modified):** The subgraph (in red) should be matched to the query graph. The irrelevant part (in blue) should be matched to the dummy node.

algorithm is illustrated in Figure 6.1. Here we present an example of this problem. We have a query graph and a subgraph shown in Figure 6.2a. The query is of size 8 and the subgraph is of size 6. Both graphs are attributed with two distinct feature values on nodes. The optimal transport matrix $\boldsymbol{T}^*$ is still sparse, while the matching between the subgraph and query graph is not a one-to-one matching.



(a) (b)

Figure 6.2: Example of matching a subgraph of a different size.

6.2.2 Entropic regularization for large graphs

Entropic regularization can be developed for FGW objective by leveraging the existing techniques developed for the Wasserstein objective and the GW objective. The main advantage is to relieve the computational burden for much larger test graphs by adopt-

ing numerical methods such as Sinkhorn iterations [38, 62]. The modified objectives are no longer referred to as distances since an entropy term is added.

The entropy-regularized FGW objective can be formulated as follows. The discrete Shannon entropy of the transport matrix is defined as

$$\mathcal{H}(\boldsymbol{T}) := -\sum_{i,j} \boldsymbol{T}_{i,j} \log\left(\boldsymbol{T}_{i,j}\right) \tag{6.1}$$

$$= -\langle \boldsymbol{T}, \log \boldsymbol{T} \rangle_F \tag{6.2}$$

where log refers to the natural logarithm. $\mathcal{H}(\boldsymbol{T})$ is a strictly concave function. Consider the constraints on $\boldsymbol{T}$, $\mathcal{H}(\boldsymbol{T})$ is zero if and only if $\boldsymbol{T}$ is a permutation matrix (one entry of 1 in each row and each column and 0s elsewhere, which is a strongly sparse matrix). Conversely, $\mathcal{H}(\boldsymbol{T})$ is large when $\boldsymbol{T}$ has many nonzero entries (not sparse). The gradient of $\mathcal{H}(\boldsymbol{T})$ with respect to $\boldsymbol{T}$

$$\nabla_{\boldsymbol{T}} \mathcal{H}(\boldsymbol{T}) = -\log(\boldsymbol{T}) - \mathbf{1}_{n \times m} \tag{6.3}$$

We can regularize the original objective function for the FGW distance by the negative entropy as

$$\text{minimize } (1-\alpha)\langle \boldsymbol{T}, \boldsymbol{M} \rangle_F + \alpha \langle \boldsymbol{L} \otimes \boldsymbol{T}, \boldsymbol{T} \rangle_F - \epsilon \mathcal{H}(\boldsymbol{T}) \tag{6.4}$$

$$\text{s.t. } \boldsymbol{T}\mathbf{1}_m = \boldsymbol{p}; \boldsymbol{T}^\top \mathbf{1}_n = \boldsymbol{q}; \boldsymbol{T} \geq \mathbf{0} \tag{6.5}$$

where ϵ is a small positive value. The regularization term $-\mathcal{H}(\boldsymbol{T})$ is strictly convex, which can effectively soften the base FGW objective function, and push the optimal solution with a higher entropy.

6.2.3 Incorporating new features or data types

The edge feature information can be easily incorporated into the adjacency matrix. This also includes the possibility of directed graphs. Related works can be found in [23, 12]. Further, the nature of optimal transport allows us to apply our method across incomparable spaces [62], like graphs, shapes, and images. Possible research problems can be: query a shape by using a graph, or query an image by using a graph.

Appendix A

A.1 General formulation of optimal transport

Metric space. A **metric space** is a pair $(\mathcal{M}, d_{\mathcal{M}})$, where $\mathcal{M}$ is a set and $d_{\mathcal{M}} : \mathcal{M} \times \mathcal{M} \to \mathbb{R}$ is a **metric** on $\mathcal{M}$ [63]. The metric $d_{\mathcal{M}}$ is also known as the distance function, measuring the distances between each pair of points in the set $\mathcal{M}$. For all points $x, y, z \in \mathcal{M}$, the metric $d_{\mathcal{M}}$ satisfies the following atoms:

1. Positivity: $d_{\mathcal{M}}(x, y) \geq 0$, with equality if and only if $x = y$,
2. Symmetry: $d_{\mathcal{M}}(x, y) = d_{\mathcal{M}}(y, x)$,
3. Triangle inequality: $d_{\mathcal{M}}(x, y) \leq d_{\mathcal{M}}(x, z) + d_{\mathcal{M}}(z, y)$.

For two metric spaces $(\mathcal{X}, d_{\mathcal{X}})$ and $(\mathcal{Y}, d_{\mathcal{Y}})$, we define the distance $d_{\mathcal{X}} \oplus d_{\mathcal{Y}}$ on the product space $\mathcal{X} \times \mathcal{Y}$ such that, for $(x, y), (x', y') \in \mathcal{X} \times \mathcal{Y}$, $d_{\mathcal{X}} \oplus d_{\mathcal{Y}}\left((x, y), (x', y')\right) = d_{\mathcal{X}}\left(x, x'\right) + d_{\mathcal{Y}}\left(y, y'\right)$.

A.1.1 Wasserstein distance

Definition 6 (Wasserstein distance)**.** [40] Consider two probability measures μ_s and μ_t with $\text{supp}[\mu_s] = \mathcal{A}$ and $\text{supp}[\mu_t] = \mathcal{B}$, suppose they live in the same metric space (Ω, d_Ω), i.e., $\mathcal{A} \subset \Omega$ and $\mathcal{B} \subset \Omega$. The Wasserstein distance between μ_s and μ_t is defined as

$$\mathcal{W}_p^{\Omega}(\mu_s, \mu_t) = \left(\inf_{\pi \in \Pi(\mu_s, \mu_t)} \int_{\mathcal{A} \times \mathcal{B}} d_\Omega(a, b)^p d\pi(a, b) \right)^{\frac{1}{p}}. \tag{A.1}$$

where $\Pi(\mu_s, \mu_t)$ is the set of all possible joint probability measure $\pi(a, b)$ on $\mathcal{A} \times \mathcal{B}$ with marginals μ_s and μ_t, i.e.,

$$\Pi(\mu_s, \mu_t) = \left\{ \pi(a, b) \geq 0, \int_{\mathcal{B}} \pi(a, b) db = \mu_s, \int_{\mathcal{A}} \pi(a, b) da = \mu_{\mathrm{t}} \right\}. \tag{A.2}$$

Compared with the discrete formulation shown in Definition 2, $\pi(a, b)$ acts as the same role as the transport matrix $\boldsymbol{T}$, while $c(a, b)$ acts the same as the cost matrix $\boldsymbol{M}$.

Kantorovitch and Monge. The formulation of Wasserstein distance is also referred to be the Kantorovitch formulation. A more historical formulation dates back to Monge. The Monge formulation seeks for a deterministic *mapping* $T : \mathcal{A} \to \mathcal{B}$ that minimizes the total cost of moving μ_s to μ_t [44],

$$\mathcal{M}^{\Omega}(\mu_s, \mu_t) = \inf_{T \# \mu_s = \mu_t} \int_{\mathcal{A}} d_\Omega(a, T(a)) d\mu_s(a) \tag{A.3}$$

where $T\#\mu_s = \mu_t$ constrains that μ_t is the push-forward measure of μ_s through a function T. The Monge problem is difficult to solve since the constraint $T\#\mu_s = \mu_t$ is non-convex. Furthermore, the solution may not exist [38]. One example is when μ_s is a Dirac measure while μ_t is not.

The Kantorovitch formulation acts as a relaxation of the Monge problem, seeking for a probabilistic *coupling* $\pi(a, b)$ to minimize the cost. It is a Linear Program and there is always a solution.

A.1.2 Gromov-Wasserstein Distance

[40] The Wasserstein distance provides a way to measure the distance between two probability measures living in the same metric space. However, it is not suitable for two measures living in different spaces, since we are not able to define a well-behaved distance function across two different spaces. For example, we are not able to compare a 2-dimensional probability measure and a 3-dimensional one with the Wasserstein distance. This issue can be solved with the Gromov-Wasserstein (GW) distance.

Definition 7 (Gromov-Wasserstein distance)**.** Consider two probability measures μ_s and μ_t with supports $\text{supp}[\mu_s] = \mathcal{X}$ and $\text{supp}[\mu_t] = \mathcal{Y}$, on metrics spaces $(\mathcal{X}, d_{\mathcal{X}})$ and $(\mathcal{Y}, d_{\mathcal{Y}})$. The spaces do not necessarily belong to a same metric space. The Gromov-Wasserstein distance is defined as

$$\mathcal{GW}_p^{\mathcal{X},\mathcal{Y}}(\mu_s, \mu_t) = \left(\inf_{\pi \in \Pi(\mu_s, \mu_t)} \int_{\mathcal{X}^2 \times \mathcal{Y}^2} \mathcal{L}(x, x', y, y')^p \, d\pi(x, y) d\pi(x', y') \right)^{\frac{1}{p}}. \tag{A.4}$$

where

$$\Pi(\mu_s, \mu_t) = \left\{ \pi(x, y) \geq 0, \int_{\mathcal{Y}} \pi(x, y) dy = \mu_s, \int_{\mathcal{X}} \pi(x, y) dx = \mu_t \right\} \tag{A.5}$$

$$\mathcal{L}(x, x', y, y') \overset{\text{def.}}{=} \mathcal{L}(d_{\mathcal{X}}(x, x'), d_{\mathcal{Y}}(y, y')) \tag{A.6}$$

The GW distance defines two distances $d_{\mathcal{X}}(x, x')$ and $d_{\mathcal{Y}}(y, y')$ *within* different metric spaces, and then evaluate the cost built on these distances. Compared with the discrete formulation shown in Definition 3, $\pi(x, y)$ acts as the same role as the transport matrix $\boldsymbol{T}$, while $d_{\mathcal{X}}(x, x')$ and $d_{\mathcal{Y}}(y, y')$ act the same as the cost matrix $\boldsymbol{C}^s$ and $\boldsymbol{C}^t$ respectively.

Gromov-Hausdorff distance. From a mathematical point of view, the GW distance (with $p = 1$) is a relaxation of the Gromov–Hausdorff distance. The Gromov–Hausdorff distance is given by [38, 64],

$$\mathcal{GH}^{\mathcal{X},\mathcal{Y}}(d_{\mathcal{X}}, d_{\mathcal{Y}}) = \frac{1}{2} \inf_{R \in \mathcal{R}(\mathcal{X},\mathcal{Y})} \sup_{((x,y),(x',y')) \in R^2} \mathcal{L}(d_{\mathcal{X}}(x, x'), d_{\mathcal{Y}}(y, y')), \tag{A.7}$$

where $R \in \mathcal{X} \times \mathcal{Y}$ is a set coupling between sets $\mathcal{X}$ and $\mathcal{Y}$, and $\mathcal{R}(\mathcal{X}, \mathcal{Y})$ consists of all the possible couplings. The Gromov–Hausdorff distance is a way to measure the distance between two metric spaces $(\mathcal{X}, d_{\mathcal{X}})$ and $(\mathcal{Y}, d_{\mathcal{Y}})$. Note that Gromov–Hausdorff

distance does not endow with the idea of optimal transport itself and does not specify a probability measure, thus cannot endow relative importance [26]. When the two metric spaces are discrete, $\mathcal{X} = \{x_i\}_{i=1}^n$, and $\mathcal{Y} = \{y_j\}_{j=1}^m$, define $\boldsymbol{C}^s_{i,i'} = d_{\mathcal{X}}(x_i, x_{i'})$, and $\boldsymbol{C}^t_{j,j'} = d_{\mathcal{Y}}(y_j, y_{j'})$, then

$$\mathcal{GH}\left(\boldsymbol{C}^s, \boldsymbol{C}^t\right) = \frac{1}{2} \inf_{\boldsymbol{R}\mathbf{1}_m > 0, \boldsymbol{R}^\top \mathbf{1}_n > 0} \max_{(i,i',j,j')} \mathcal{L}\left(\boldsymbol{C}^s_{i,i'}, \boldsymbol{C}^t_{j,j'}\right) \boldsymbol{R}_{i,j} \boldsymbol{R}_{i',j'}. \tag{A.8}$$

where $\boldsymbol{R} \in \{0,1\}^{n \times m}$ is a binary matrix. The function value is nonzero if and only if both $\boldsymbol{R}_{i,j}$ and $\boldsymbol{R}_{i',j'}$ are equal to one. The GW distance (Definition 3) relaxes the Gromov–Hausdorff problem to be a continuous problem [43]. Then the "soft" matching between edge (i, i') source graph, and edge (j, j') of target graph is formulated by $\boldsymbol{T}_{i,j}\boldsymbol{T}_{i',j'}$ [45].

A.1.3 Probabilistic formulation

The Wasserstein distance and Gromov-Wasserstein distance can be equivalently written in probabilistic forms.

Wasserstein distance.[38]

$$\mathcal{W}_p^{\Omega}(\mu_s, \mu_t) = \left(\inf_{\pi \in \Pi(\mu_s, \mu_t)} \mathbb{E}_{\pi(a,b)} \left[d_{\Omega}(a,b)^p \right] \right)^{\frac{1}{p}}. \tag{A.9}$$

Gromov-Wasserstein distance.

$$\mathcal{GW}_p^{\mathcal{X},\mathcal{Y}}(\mu_s, \mu_t) = \left(\inf_{\pi \in \Pi(\mu_s, \mu_t)} \mathbb{E}_{\pi(x,y)} \mathbb{E}_{\pi(x',y')} \left[\mathcal{L}(x, x', y, y')^p \right] \right)^{\frac{1}{p}}. \tag{A.10}$$

A.1.4 Structured objects

Structured data is a broader term that includes *graph*. Any kind of data composed with some specific structure or organization can be referred to as structured data. Data elements are connected with each other with well-defined relationships. Besides structure, structured data also has an emphasis on the characteristics/features of data points. In this way, structured data can be defined as *generalized labeled graph* [44]. A labeled graph can be viewed as an ensemble of features or attributes, associated with structural information. Other structured data includes 3D shape or object, time-series signal, structured image, hierarchical database, etc [38].

In Section 3.3, we introduced the concept of modeling labeled graphs as probability measures. Expanding on this, we now introduce the definition of structured objects using the term "metric measure space".

Metric measure space. A metric measure space (denoted as mm-space) is a triplet $(\mathcal{M}, d_{\mathcal{M}}, \mu)$ where μ is a probability measure on $\mathcal{M}$ [40]. Metric measure space is built on metric space and equipped with a notion of probability.

Definition 8 (Structured objects)**.** [40] A structured object over a metric space (Ω, d_Ω) is the triplet $(\mathcal{X} \times \mathcal{A}, d_\mathcal{X}, \mu)$, where $(\mathcal{X}, d_\mathcal{X})$ is a compact metric space, $\mathcal{A}$ is a compact set of Ω, and μ is a fully supported probability measure over $\mathcal{X} \times \mathcal{A}$. The space (Ω, d_Ω) is denoted as the feature space. The set $\mathcal{A}$ itself is the feature information of the structured object and the space $(\mathcal{X}, d_\mathcal{X})$ is its structure information.

For graph, it naturally has an edge set in definition, and x is the neighborhood information of each node; For point cloud data, x is the coordinate of one point, though it is the relative distance that indicates the structure of the data [40, 44]; For time series signals [44], x is a timestamp (or temporal position). A simple example of 2-dimensional point cloud data and its probability measure are shown in Figure A.1. We can observe that μ acts as a joint distribution of μ^F and μ^S.

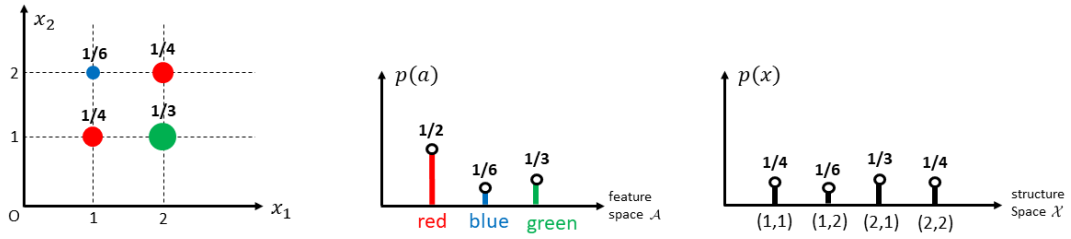


Figure A.1: μ **is the joint distribution of** μ^F **and** μ^S**:** The point cloud data (Left) consists of four data points. Each point has a color for its feature and a 2-dimensional position (x_1, x_2) in the coordinate system. The relative importance is presented as proportional to the radius. The measure μ can be written as $\mu = \frac{1}{6}\delta\{(1,1), \text{red}\} + \frac{1}{4}\delta\{(1,2), \text{blue}\} + \frac{1}{3}\delta\{(2,1), \text{yellow}\} + \frac{1}{4}\delta\{(2,2), \text{red}\}$. The feature distribution μ^F (Middle) is one marginal, $\mu^F = \frac{1}{2}\delta\{\text{red}\} + \frac{1}{6}\delta\{\text{blue}\} + \frac{1}{3}\delta\{\text{yellow}\}$. Similarly, the structural distribution μ^S (Right) is $\mu^S = \frac{1}{4}\delta\{(1,1)\} + \frac{1}{6}\delta\{(1,2)\} + \frac{1}{3}\delta\{(2,1)\} + \frac{1}{4}\delta\{(2,2)\}$.

The equivalence of structured objects, named as isomorphism, can be viewed as a generalization of Definition 1. However, we have to first define the isometry between two metric spaces. The two definitions are as follows.

Definition 9 (Isometry of metric spaces)**.** [40] Two metric spaces $(\mathcal{X}, d_\mathcal{X})$ and $(\mathcal{Y}, d_\mathcal{Y})$ are isometric if there exists a map $f : \mathcal{X} \to \mathcal{Y}$ that preserves the distances,

$$\forall x, x' \in \mathcal{X}, d_\mathcal{Y}\left(f(x), f\left(x'\right)\right) = d_\mathcal{X}\left(x, x'\right). \tag{A.11}$$

The map f is called the *isometry* of the two metric spaces.

Definition 10 (Isomorphism of structured objects)**.** [40] Two structured objects $(\mathcal{X} \times \mathcal{A}, d_\mathcal{X}, \mu_s)$ and $(\mathcal{Y} \times \mathcal{B}, d_\mathcal{Y}, \mu_t)$ are equivalent if there exists an *isometry* f such that f is measure-preserving, i.e. $f\#\mu_s = \mu_t$.

The notion of isometry itself can be used to compare the structures between two structured objects, while isomorphism also includes the features. The push-forward notation means that only the positions of supp$[\mu_s]$ and supp$[\mu_t]$ are replaced, while the values of the measure are preserved.

A.1.5 Fused Gromov-Wasserstein Distance

[40] To jointly consider features and structure for two objects, one in space $(\mathcal{X} \times \mathcal{A})$ and the other in $(\mathcal{Y} \times \mathcal{B})$. The formulation of Wasserstein distance and Gromov-Wasserstein distance is combined into Fused Gromov-Wasserstein distance. The metrics of the spaces are still $d_\Omega(a, b)$, $d_\mathcal{X}(x, x')$, and $d_\mathcal{Y}(y, y')$.

Definition 11 (Fused Gromov-Wasserstein distance)**.** For $\alpha \in [0, 1]$, the Fused Gromov-Wasserstein distance between μ_s and μ_t is defined as

$$\mathcal{FGW}_{\alpha,p,q}^{\Omega,\mathcal{X},\mathcal{Y}}(\mu_s, \mu_t) = \left(\inf_{\pi \in \Pi(\mu_s, \mu_t)} \int_{(\mathcal{X}\times\mathcal{A})^2 \times (\mathcal{Y}\times\mathcal{B})^2} \left((1-\alpha) d_\Omega(a, b)^q + \alpha \mathcal{L}(x, y, x', y')^q\right)^p \right.$$
$$\left. d\pi((x, a), (y, b)) d\pi((x', a'), (y', b')) \right)^{\frac{1}{p}}. \tag{A.12}$$

Actually, we can also define an adapted Gromov-Wasserstein distance on metric space $(\mathcal{X} \times \mathcal{A}, d_\mathcal{X} \oplus d_\mathcal{A})$ and $(\mathcal{Y} \times \mathcal{B}, d_\mathcal{Y} \oplus d_\mathcal{B})$. However, this formulation can not always distinguish different graphs, and the FGW distance benefits from better mathematical properties [40].

Bibliography

[1] K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann, "Benchmark data sets for graph kernels," 2016.

[2] S.-M. Hsieh, C.-C. Hsu, and L.-F. Hsu, "Efficient Method to Perform Isomorphism Testing of Labeled Graphs," in *Computational Science and Its Applications - ICCSA 2006* (M. L. Gavrilova, O. Gervasi, V. Kumar, C. J. K. Tan, D. Taniar, A. Laganá, Y. Mun, and H. Choo, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 422–431, Springer, 2006.

[3] L. Livi and A. Rizzi, "The graph matching problem," *Pattern Analysis and Applications*, vol. 16, pp. 253–283, Aug. 2013.

[4] S. Bhadange, A. Arora, and A. Bhattacharya, "GARUDA: a system for large-scale mining of statistically significant connected subgraphs," *Proceedings of the VLDB Endowment*, vol. 9, pp. 1449–1452, Sept. 2016.

[5] F. Katsarou, N. Ntarmos, and P. Triantafillou, "Performance and scalability of indexed subgraph query processing methods," *Proceedings of the VLDB Endowment*, vol. 8, pp. 1566–1577, Aug. 2015. Number: 12 Publisher: VLDB Endowment Inc.

[6] M. Kuramochi and G. Karypis, "Frequent subgraph discovery," in *Proceedings 2001 IEEE International Conference on Data Mining*, pp. 313–320, Nov. 2001.

[7] E. L. Goodman and D. Grunwald, "Streaming Temporal Graphs: Subgraph Matching," in *2019 IEEE International Conference on Big Data (Big Data)*, pp. 4977–4986, Dec. 2019. arXiv:2004.00215 [cs].

[8] D. Koutra, A. Parikh, A. Ramdas, and J. Xiang, "Algorithms for Graph Similarity and Subgraph Matching,"

[9] V. Lyzinski, D. E. Fishkind, M. Fiori, J. T. Vogelstein, C. E. Priebe, and G. Sapiro, "Graph Matching: Relax at Your Own Risk," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, pp. 60–73, Jan. 2016. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[10] E. L. Lawler, "The Quadratic Assignment Problem," *Management Science*, vol. 9, pp. 586–599, July 1963. Publisher: INFORMS.

[11] 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, p. e0121002, Apr. 2015.

[12] Z. Zhang, Y. Xiang, L. Wu, B. Xue, and A. Nehorai, "KerGM: Kernelized Graph Matching," Nov. 2019. arXiv:1911.11120 [cs, stat].

[13] 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*, ICMR '16, (New York, NY, USA), pp. 167–174, Association for Computing Machinery, 2016.

[14] L. Chapel, M. Z. Alaya, and G. Gasso, "Partial Optimal Tranport with applications on Positive-Unlabeled Learning," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 2903–2913, Curran Associates, Inc., 2020.

[15] S. Medasani, R. Krishnapuram, and Y. Choi, "Graph matching by relaxation of fuzzy assignments," *IEEE Transactions on Fuzzy Systems*, vol. 9, pp. 173–182, Feb. 2001. Conference Name: IEEE Transactions on Fuzzy Systems.

[16] C. Schellewald and C. Schnörr, "Probabilistic Subgraph Matching Based on Convex Relaxation," in *Energy Minimization Methods in Computer Vision and Pattern Recognition* (A. Rangarajan, B. Vemuri, and A. L. Yuille, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 171–186, Springer, 2005.

[17] J. Yan, H. Xu, H. Zha, X. Yang, H. Liu, and S. Chu, "A Matrix Decomposition Perspective to Multiple Graph Matching," in *2015 IEEE International Conference on Computer Vision (ICCV)*, (Santiago, Chile), pp. 199–207, IEEE, Dec. 2015.

[18] V. Carletti, *Exact and Inexact Methods for Graph Similarity in Structural Pattern Recognition*. PhD thesis, Université de Caen; Universita degli studi di Salerno, May 2016.

[19] Y. Tian, R. C. McEachin, C. Santos, D. J. States, and J. M. Patel, "SAGA: a subgraph matching tool for biological graphs," *Bioinformatics*, vol. 23, pp. 232–239, Jan. 2007.

[20] O. Duchenne, F. Bach, I.-S. Kweon, and J. Ponce, "A Tensor-Based Algorithm for High-Order Graph Matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, pp. 2383–2395, Dec. 2011. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[21] Z. Wang, Y. Wu, and F. Liu, "Tensor Affinity Learning for Hyperorder Graph Matching," *Mathematics*, vol. 10, p. 3806, Jan. 2022. Number: 20 Publisher: Multidisciplinary Digital Publishing Institute.

[22] X. Yang, Z.-Y. Liu, and H. Qiao, "A Continuation Method for Graph Matching Based Feature Correspondence," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, pp. 1809–1822, Aug. 2020. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[23] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan, "NeMa: fast graph search with label similarity," *Proceedings of the VLDB Endowment*, vol. 6, pp. 181–192, Jan. 2013.

[24] L. Liu, B. Du, J. xu, and H. Tong, "G-Finder: Approximate Attributed Subgraph Matching," in *2019 IEEE International Conference on Big Data (Big Data)*, pp. 513–522, Dec. 2019.

[25] D. Eppstein, "Subgraph Isomorphism in Planar Graphs and Related Problems," 1999.

[26] F. Mémoli, "Gromov–Wasserstein Distances and the Metric Approach to Object Matching," *Foundations of Computational Mathematics*, vol. 11, pp. 417–487, Aug. 2011.

[27] P. Foggia, G. Percannella, and M. Vento, "Graph matching and learning in pattern recognition in the last 10 years," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 28, p. 1450001, Feb. 2014. Publisher: World Scientific Publishing Co.

[28] 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, pp. 265–298, May 2004.

[29] T. K. Tu, J. D. Moorman, D. Yang, Q. Chen, and A. L. Bertozzi, "Inexact Attributed Subgraph Matching," in *2020 IEEE International Conference on Big Data (Big Data)*, (Atlanta, GA, USA), pp. 2575–2582, IEEE, Dec. 2020.

[30] K. Riesen and H. Bunke, "Approximate graph edit distance computation by means of bipartite graph matching," *Image and Vision Computing*, vol. 27, pp. 950–959, June 2009.

[31] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou, "Comparing stars: on approximating graph edit distance," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 25–36, 2009.

[32] N. M. Kriege, F. D. Johansson, and C. Morris, "A survey on graph kernels," *Applied Network Science*, vol. 5, pp. 1–42, Dec. 2020. Number: 1 Publisher: SpringerOpen.

[33] N. Kriege and P. Mutzel, "Subgraph Matching Kernels for Attributed Graphs,"

[34] S. Dutta, P. Nayek, and A. Bhattacharya, "Neighbor-Aware Search for Approximate Labeled Graph Matching using the Chi-Square Statistics," in *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, (Republic and Canton of Geneva, CHE), pp. 1281–1290, International World Wide Web Conferences Steering Committee, 2017.

[35] S. Agarwal, S. Dutta, and A. Bhattacharya, "VerSaChI: Finding Statistically Significant Subgraph Matches using Chebyshev's Inequality," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, (Virtual Event Queensland Australia), pp. 2812–2816, ACM, Oct. 2021.

[36] C. Clark, *Multiobjective Optimization for the Alignment of Protein Networks*. PhD thesis.

[37] Rex, Ying, Z. Lou, J. You, C. Wen, A. Canedo, and J. Leskovec, "Neural Subgraph Matching," Oct. 2020. arXiv:2007.03092 [cs, stat].

[38] G. Peyré and M. Cuturi, "Computational Optimal Transport," Mar. 2020. arXiv:1803.00567 [stat].

[39] V. Titouan, N. Courty, R. Tavenard, C. Laetitia, and R. Flamary, "Optimal Transport for structured data with application on graphs," in *Proceedings of the 36th International Conference on Machine Learning*, pp. 6275–6284, PMLR, May 2019. ISSN: 2640-3498.

[40] T. Vayer, L. Chapel, R. Flamary, R. Tavenard, and N. Courty, "Fused Gromov-Wasserstein distance for structured objects: theoretical foundations and mathematical properties," Nov. 2018. arXiv:1811.02834 [cs, stat].

[41] A. Barbe, M. Sebban, P. Gonçalves, P. Borgnat, and R. Gribonval, "Graph Diffusion Wasserstein Distances," in *Machine Learning and Knowledge Discovery in Databases* (F. Hutter, K. Kersting, J. Lijffijt, and I. Valera, eds.), vol. 12458, pp. 577–592, Cham: Springer International Publishing, 2021. Series Title: Lecture Notes in Computer Science.

[42] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao, "Neighborhood based fast graph search in large networks," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, (Athens Greece), pp. 901–912, ACM, June 2011.

[43] G. Peyré, M. Cuturi, and J. Solomon, "Gromov-Wasserstein Averaging of Kernel and Distance Matrices," in *Proceedings of The 33rd International Conference on Machine Learning*, pp. 2664–2672, PMLR, June 2016. ISSN: 1938-7228.

[44] T. Vayer, "A contribution to Optimal Transport on incomparable spaces," Nov. 2020. arXiv:2011.04447 [cs, stat].

[45] S. Chowdhury and T. Needham, "Gromov-Wasserstein Averaging in a Riemannian Framework," Apr. 2020. arXiv:1910.04308 [cs, math].

[46] H. Jin, Z. Yu, and X. Zhang, "Orthogonal Gromov-Wasserstein discrepancy with efficient lower bound," in *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*, pp. 917–927, PMLR, Aug. 2022. ISSN: 2640-3498.

[47] R. E. Burkard and L. S. Pitsoulis, "The Quadratic Assignment Problem,"

[48] R. Flamary, N. Courty, A. Gramfort, M. Z. Alaya, A. Boisbunon, S. Chambon, L. Chapel, A. Corenflos, K. Fatras, N. Fournier, L. Gautheron, N. T. Gayraud, H. Janati, A. Rakotomamonjy, I. Redko, A. Rolet, A. Schutz, V. Seguy, D. J. Sutherland, R. Tavenard, A. Tong, and T. Vayer, "Pot: Python optimal transport," *Journal of Machine Learning Research*, vol. 22, no. 78, pp. 1–8, 2021.

[49] C. Vincent-Cuaz, *Optimal transport for graph representation learning*. PhD thesis, Mar. 2023.

[50] X. Liu, J. Cheng, Y. Song, and X. Jiang, "Boosting Graph Structure Learning with Dummy Nodes," June 2022. arXiv:2206.08561 [cs].

[51] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, "SuperGlue: Learning Feature Matching with Graph Neural Networks," Mar. 2020. arXiv:1911.11763 [cs].

[52] I. Roy, V. S. B. R. Velugoti, S. Chakrabarti, and A. De, "Interpretable Neural Subgraph Matching for Graph Retrieval," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 8115–8123, June 2022. Number: 7.

[53] H. Maron and Y. Lipman, "(Probably) Concave Graph Matching," Dec. 2018. arXiv:1807.09722 [math].

[54] M. Frank and P. Wolfe, "An algorithm for quadratic programming," *Naval Research Logistics Quarterly*, vol. 3, no. 1-2, pp. 95–110, 1956. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800030109.

[55] M. Jaggi, "Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization,"

[56] X. Liu, R. Zeira, and B. J. Raphael, "PASTE2: Partial Alignment of Multi-slice Spatially Resolved Transcriptomics Data," Jan. 2023. Pages: 2023.01.08.523162 Section: New Results.

[57] C. Vincent-Cuaz, R. Flamary, M. Corneli, T. Vayer, and N. Courty, "Semi-relaxed Gromov-Wasserstein divergence with applications on graphs," 2022.

[58] N. Bonneel, "Displacement Interpolation Using Lagrangian Mass Transport,"

[59] M. Kanehisa and S. Goto, "KEGG: Kyoto Encyclopedia of Genes and Genomes," *Nucleic Acids Research*, vol. 28, pp. 27–30, Jan. 2000.

[60] F. M. Castelli, "Keggutils," Dec. 2022.

[61] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang, "Efficient Subgraph Matching by Postponing Cartesian Products," in *Proceedings of the 2016 International Conference on Management of Data*, (San Francisco California USA), pp. 1199–1214, ACM, June 2016.

[62] J. Solomon, G. Peyré, V. G. Kim, and S. Sra, "Entropic metric alignment for correspondence problems," *ACM Transactions on Graphics*, vol. 35, no. 4, pp. 72:1–72:13, 2016.

[63] D. Burago, Y. Burago, and S. Ivanov, *A Course in Metric Geometry*, vol. 33 of *Graduate Studies in Mathematics*. Providence, Rhode Island: American Mathematical Society, June 2001.

[64] V. Oles and K. R. Vixie, "Lipschitz (non-)equivalence of the Gromov–Hausdorff distances, including on ultrametric spaces," Sept. 2022. arXiv:2204.10250 [cs, math].