DISSERTATION
submitted
to the
Combined Faculty for the Natural Sciences and Mathematics
of
Heidelberg University, Germany
for the degree of
Doctor of Natural Sciences

Put forward by

M.Sc. Ayser Armiti

Born in: Nablus, Palestine

Oral examination:.....

Geometric Graphs: Matching, Similarity, and Indexing

Advisor: Prof. Dr. Michael Gertz

Abstract

For many applications, such as drug discovery, road network analysis, and image processing, it is critical to study spatial properties of objects in addition to object relationships. Geometric graphs provide a suitable modeling framework for such applications, where vertices are located in some 2D space. As a result, searching for similar objects is tackled by estimating the similarity of the structure of different graphs. In this case, inexact graph matching approaches are typically employed. However, computing the optimal solution to the graph matching problem is proved to be a very complex task. In addition to this, approximate approaches face many problems such as poor scalability with respect to graph size and less tolerance to changes in graph structure or labels.

In this thesis, we propose a framework to tackle the inexact graph matching problem for geometric graphs in 2D space. It consists of a pipeline of three components that we design to cope with the requirements of several application domains. The first component of our framework is an approach to estimate the similarity of vertices. It is based on the string edit distance and handles any labeling information assigned to the vertices and edges. Based on this, we build the second component of our framework. It consists of two algorithms to tackle the inexact graph matching problem. The first algorithm adopts a probabilistic scheme, where we propose a density function that estimates the probability of the correspondences between vertices of different graphs. Then, a match between the two graphs is computed utilizing the expectation maximization technique. The second graph matching algorithm follows a continuous optimization scheme to iteratively improve the match between two graphs. For this, we propose a vertex embedding approach so that the similarity of different vertices can be easily estimated by the Euclidean distance. The third component of our framework is a graph indexing structure, which helps to efficiently search a graph database for similar graphs. We propose several lower bound graph distances that are used to prune non-similar graphs and reduce the response time.

Using representative geometric graphs extracted from a variety of applications domains, such as chemoinformatics, character recognition, road network analysis, and image processing, we show that our approach outperforms existing graph matching approaches in terms of matching quality, classification accuracy, and runtime.

Zusammenfassung

Für viele Anwendungen wie beispielsweise die Arzneimittelforschung, Straßennetzwerkanalyse und Bildverarbeitung ist die Analyse räumlicher Eigenschaften von Objekten zusätzlich zu den Beziehungen zwischen den Objekten von zentraler Bedeutung. Für solche Anwendungen bieten geometrische Graphen einen geeigneten Modellierungsrahmen, in dem Knoten im zweidimensionalen Raum dargestellt werden. Hierdurch können Ähnlichkeiten zwischen Objekten durch die Abschätzung der Ähnlichkeiten ihrer Graphen bestimmt werden. Dafür werden typischerweise inexakte Graph-Matching-Verfahren verwendet. Allerdings wurde gezeigt, dass die Berechnung einer optimalen Lösungen für das Graph-Matching-Problem eine sehr komplexe Aufgabe darstellt. Außerdem sind die Skalierbarkeit in Bezug auf die Größe der Graphen und die Toleranz gegenüber Änderungen in der Graphstruktur weitere Schwächen inexakter Ansätze.

In dieser Arbeit stellen wir ein neues Framework vor, um das inexakte Graph-Matching-Problem für geometrische Graphen im zweidimensionalen Raum zu lösen. Dieses besteht aus einer dreiteiligen Pipeline, die wir so entworfen haben, dass die Anforderungen zahlreicher Anwendungsgebiete berücksichtigt werden. Die erste Komponente ist ein Ansatz zur Abschätzung von Knotenähnlichkeiten, die auf der String-Edit-Distance basiert und jegliche Knoten- und Kanteneigenschaften berücksichtigt. Darauf aufbauend besteht die zweite Komponente aus zwei Algorithmen zur Berechnung des Graph-Matching-Problems. Der erste Algorithmus basiert auf einer Wahrscheinlichkeitsschätzung, für die wir eine Dichtefunktion zur Berchnung der Übereinstimmungswahrscheinlichkeiten zwischen Knoten verschiedener Graphen entwickelt haben. Danach wird die Übereinstimmung zwischen zwei Graphen mithilfe von Expectation Maximization errechnet. Der zweite Graph-Matching-Algorithmus wendet dagegen ein kontinuierliches Optimierungsschema an, um die Übereinstimmungen iterativ zu verbessern. Hierfür schlagen wir einen Ansatz zur Einbettung der Konten vor, so dass die Ähnlichkeit verschiedener Knoten schlicht anhand der Euklidischen Distanz abgeschätzt werden kann. Die letzte Komponente des Frameworks bildet schließlich eine Graph-Indexstruktur, die das effiziente Durchsuchen einer Graphdatenbank nach ähnlichen Graphen ermöglicht. Zusätzlich stellen wir mehrere Graphabstandsfunktionen zum Ausschließen unähnlicher Graphen vor, um die Laufzeit zu optimieren.

Anhand einer repräsentativen Auswahl geometrischer Graphen aus unterschiedlichen Anwendungsbereichen zeigen wir, dass unser Ansatz bessere Ergebnisse

bezüglich Matching-Qualität, Klassifikationsgenauigkeit und Laufzeit erzielt als existierende Ansätze.

Acknowledgements

I would like to express my deepest appreciation to all who supported me during my PhD study. First and foremost, I would like to express my sincere gratitude to Prof. Dr. Michael Gertz who gave me the golden opportunity to join his research group. On countless occasions, his excellent advice has been a tremendous support to learn a great deal about research, academic writing, and presentation skills. I would also like to thank the committee members JProf. Dr. Heike Leitte, Prof. Dr. Gerhard Reinelt, and PD. Dr. Wolfgang Merkle for their time and feedback.

Many friends and colleagues contributed in many different ways to the completion of this thesis. Dr. Canh Tran Van, Florian Flatow, Hamed Abdelhaq, Jannik Strötgen, Katarina Gavrić, Dr. Mohammed AbuJarour, Osama Samoudi, and Dr. Wesam Herbawi — thank you all very much. I would also like to acknowledge my other friends and colleagues Dr. Anh Le Van Quoc, Christian Sengstock, Hui Li, and Thomas Bögel for their interaction during my four years of study. Thanks Florian, Christian, and Jannik for the valuable discussions that helped me to integrate in the German culture.

I am thankful and proud to be a graduate student of Heidelberg University. My deepest gratitude goes to the "Deutsche Akademische Austauschdienst Dienst" (DAAD) for their financial support.

Alaa Juber, you believed in me long before I did. It is impossible to put into words what your faith in me has done for me. Our last five years were tremendously hard. Transferring to Germany with two kids, moving to four different apartments, you and me doing the PhD at two far apart universities, all these and many more strength the taste of this accomplishment.

In a faraway place, where life is like a fairy tale, a young Palestinian couple dreamed that their child pursues his higher education. My parents, your dream is getting closer to become true as I am putting the final touches to my PhD study. This thesis is dedicated to you.

To my parents.

Contents

1 Introduction to Graph Similarity **1**

1.1 Graph Similarity 2
1.2 Labeled Geometric Graphs 3
1.3 Objectives 3
1.4 Challenges 4
1.5 Contributions 5
1.6 Structure of the Thesis 7

2 The Graph Matching Problem **9**

2.1 Preliminary Definitions 9
2.2 Graph Matching 12
2.3 Exact Graph Matching 15
2.3.1 Graph Isomorphism 15
2.3.2 Geometric Isomorphism 17
2.3.3 Subgraph Isomorphism 19
2.3.4 Maximum Common Subgraph 19
2.4 Inexact Graph Matching 20
2.4.1 Graph Edit Distance 22
2.4.2 Spectral Graph Matching 27
2.4.3 Continuous Optimization Approaches 37
2.5 Summary and Discussion 39

3 Vertex Similarity **41**

3.1 Related Work 42
3.2 Local-based Vertex Similarity 45
3.2.1 Vertex Edit Distance 47
3.3 Vertex Similarity for 2D Geometric Graphs 49
3.3.1 Optimal Solution 49

3.3.2 Approximate Solution . . . 52
3.4 Experimental Evaluation . . . 58
3.4.1 Graph Similarity and Classification . . . 60
3.4.2 Graph Matching . . . 63
3.4.3 Subgraph Matching . . . 65
3.4.4 Scalability Study . . . 67
3.5 Summary and Discussion . . . 68

4 Geometric Graph Matching: A Probabilistic Approach 73
4.1 Preliminaries . . . 75
4.1.1 Maximum Likelihood Estimation . . . 76
4.1.2 Expectation Maximization . . . 78
4.2 Graph Mixture Model . . . 79
4.2.1 MLE for Graph Matching . . . 80
4.2.2 A Graph Density Function . . . 81
4.3 Graph Matching Using the EM Technique . . . 86
4.3.1 Expectation . . . 87
4.3.2 Maximization . . . 88
4.3.3 Generalized EM . . . 89
4.3.4 Complexity Analysis . . . 92
4.4 Experimental Evaluation . . . 94
4.4.1 Matching Quality . . . 95
4.4.2 Graph Similarity and Classification . . . 98
4.4.3 Scalability Study . . . 99
4.4.4 Parameters Analysis . . . 102
4.5 Summary and Discussion . . . 104

5 Efficient Geometric Graph Matching Using Vertex Embedding 107
5.1 Embedding into Vector Spaces . . . 108
5.2 Vertex Embedding . . . 110
5.2.1 Prototype Selection . . . 113
5.3 Iterative Graph Matching . . . 116
5.3.1 Candidate Initialization . . . 117
5.3.2 Solving the Assignment Problem . . . 118
5.3.3 Similarity Refinement . . . 119
5.3.4 Candidate Selection . . . 121
5.3.5 Convergence . . . 121

5.4 Experimental Evaluation 124
5.4.1 Parameters Analysis 126
5.4.2 Matching Quality 129
5.4.3 Scalability Study 132
5.5 Summary and Discussion 133

6 Geometric Graph Similarity Search **135**
6.1 Problem Definition 135
6.2 Related Work 137
6.3 Geometric Graph Indexing 140
6.3.1 Geometric Graph Distance Measure 141
6.3.2 Indexing Structure 145
6.3.3 Vertex Range Query Problem 145
6.3.4 From Similar Vertices to Similar Graphs 147
6.3.5 Lower Bound Distances 149
6.4 Experimental Evaluation 155
6.4.1 Index Construction 156
6.4.2 Pruning Performance 157
6.4.3 Response Time 161
6.5 Summary 162

7 Conclusions and Future Work **165**
7.1 Summary 165
7.2 Future Work 167

A Datasets **169**
A.1 Road Networks 169
A.2 Chinese Characters 171
A.3 CMU dataset 171
A.4 IAM Graph Dataset 172
A.4.1 COIL-100 172
A.4.2 AIDS 173
A.4.3 GREC 174

Bibliography **175**

Chapter 1

Introduction to Graph Similarity

In recent years, commodity hardware enable the collection of huge amounts of data from a variety of application domains, such as meteorology, bioinformatics, and geoinformatics. Since there is a correlation between the observed data and the variables controlling a phenomenon, computers are used to analyze such data to extract patterns and predict future behaviors. For example, in bioinformatics, medical records are used to predict relationships between clinical tests and the possibility of having cancer; in meteorology, sensor data, such as pressure and temperature, is used for weather forecasting.

For a computer to analyze such data, first, it is critical to decide how to represent the objects and entities related to an application domain. In addition to the vector-based representation, recently, more and more applications have been using the *graph data structure* for the modeling of objects and relationships. In general, a graph consists of nodes or vertices that represent entities, and edges connecting the vertices represent the relationships between them.

There are several examples where graphs are used to represent data. Graphs have been used to represent chemical compounds where the atoms are modeled by the vertices and the covalent bonds are represented by edges. A protein is modeled as a graph such that vertices represent secondary structures and spatial relationships between them are represented by the edges. A road network is typically modeled as a graph such that cross roads are represented by vertices and road segments are modeled by edges. The World Wide Web is naively represented as a graph where vertices represent web pages and edges represent hyperlinks. With the recent emergence of social networks, graphs are used to model users and their friendships.

The graph data structure has several advantages and adapts the number of vertices and edges to the complexity of each object. Different types of relationships

can be represented by edges connecting two vertices or even more. Features can be easily assigned to the vertices and edges representing some properties related to an application domain, such as the latitudes and longitudes of elements of a road network.

1.1 Graph Similarity

For the previously mentioned applications and many more, a critical task is to compare two graphs and to determine their similarity. For example, in computer vision and pattern recognition, graph similarity is used for face recognition [128] and object tracking [23]; in chemoinformatics, it is used to extract similar chemical compounds [96, 97] and to predict bio-activities [14, 121]; in bioinformatics, graph similarity is used for the identification of protein motifs [24], metabolic pathways [133], and to understand gene regulation mechanisms [34]; in geoinformatics, graph similarity is used for route planning, street connectivity analysis [105], and spatial information matching [122].

Considering such a wide variety of application domains, the notion of similarity is often subjective and reflects the needs of each application. For example, graph clustering and classification algorithms require a *distance function* that describes the similarity of two graphs as a real-valued number. Other graph applications, like frequent substructure discovery, consider two graphs similar if one is a substructure of the other. Other applications, like motif discovery, search for *common structures* in a database of graphs. For this, two graphs are considered similar if there is a substructure from the first graph that is identical to a substructure from the second one. In addition to that, the notion of *inexact structure similarity* has been proposed to quantify the similarity of non-identical graphs based on the amount of modifications needed to make one graph identical to another.

Basically, the similarity between two graphs consists of the similarity of the labels that are assigned to the vertices and edges, in addition to the similarity of their structure. Different functions are used to estimate the similarity between labels, such as the Euclidean distance or a Dirac function that assigns a distance of 0 if the two vertices have the same label and 1 otherwise. On the other hand, estimating the structural similarity between two graphs is a very complex task. It is denoted by the *graph matching problem*, which searches for a mapping between the vertices of two graphs such that the mapped vertices have similar connectivity.

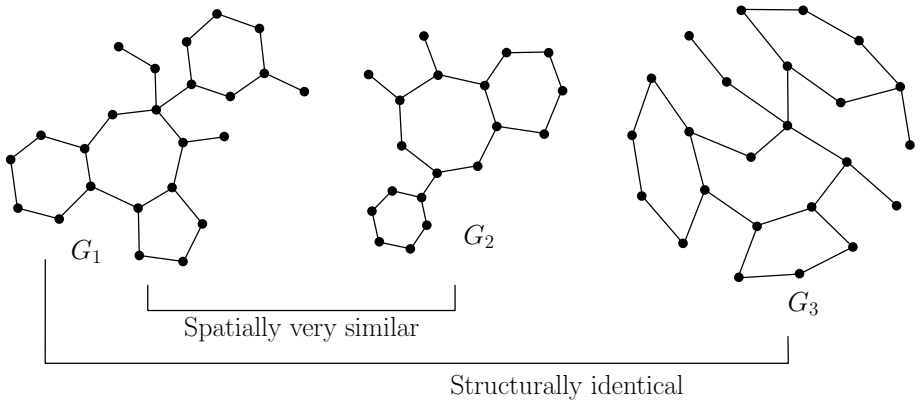


Figure 1.1: The difference between the similarity of geometric and non-geometric graphs.

1.2 Labeled Geometric Graphs

One of the biggest advantages of using graphs is the possibility to consider different features for different parts of a graph. This is possible by assigning *labeling information* to both the vertices and edges of a graph. For example, in cheminformatics, a label is used to represent an atom type, in computer vision, a label represents the color of a group of pixels. In these cases and many more, a graph is called *labeled graph*.

A common property of the above graph-based applications is that the vertices of the graphs have coordinates in some 2- or 3-dimensional space, which are used to capture the spatial properties of the objects under study. For example, the atoms of a protein or a chemical compound are labeled with their coordinates in 2D or 3D space, latitudes and longitudes are assigned to road networks, different features appearing in an image are assigned coordinates indicating their locations in 2D space. Generally, the graphs that are used to model such objects are called *geometric graphs*. Figure 1.1 gives a feeling on why geometry matters. If no spatial information is considered, graph G_1 is identical to graph G_3. However, spatial properties make G_1 more similar to G_2 than G_3, even though G_1 and G_2 are not identical in terms of the number of vertices and the structure.

1.3 Objectives

Motivated by a wide variety of application domains, in this thesis, we are interested in studying the problem of geometric graph similarity. In addition to the labeling information and graph structure, the spatial properties of geometric graphs, indicated

by the coordinates of the vertices, must be considered by a graph similarity measure or a graph matching algorithm.

The main goal of this thesis is to build a general framework for graph matching and similarity that can be used in several application domains. Our framework consists of the following three components. 1) A vertex similarity measure that estimates the similarity between vertices of different graphs. 2) Utilizing the concept of vertex similarity, we propose graph matching algorithms that estimate the similarity of different geometric graphs. 3) In the context of querying a graph database for similar graphs, our framework uses a *graph indexing structure* to efficiently search for similar graphs and prune non-similar ones.

1.4 Challenges

Graphs with their flexible modeling power are widely used for the representation of objects and relationships. However, flexibility comes with a high price. In the following, we summarize the main challenges that we faced during the study of the similarity of geometric graphs.

1. Several trivial operations of the vector-based representation have an exponential runtime complexity when applied to the graph representation. For example, computing the similarity of two vectors has a linear runtime complexity. However, computing the optimal solution to the graph matching problem has in general an exponential runtime complexity [20, 98].

2. Spatial properties of geometric graphs increase the complexity of graph matching. Since the coordinates of the vertices depend on the particular reference frame of each graph, geometric graph similarity is defined to handle invariance under geometric transformations, such as translation, rotation, and in some cases scaling. For example, in Figure 1.1, a rotation followed by a scaling is required to estimate the similarity between the two graphs G_1 and G_2.

3. Most of the graph indexing structures assume that the vertices and/or the edges are labeled with a discrete alphabet, e.g., [53, 132, 135, 138, 139]. However, for geometric graphs, the vertices are assigned coordinates as real-valued labels. In addition to this, using a spatial index structure such as an R-tree based on the coordinates of the vertices is incapable of handling similarity under geometric transformation.

4. We found it challenging and difficult to search for and study related work. Several research communities have proposed heuristics and algorithms for graph matching and similarity. The lack of standardization and communication between such communities generate similar algorithms and concepts under different naming conventions. For example, the structure of a vertex and its direct connected vertices has been widely used for graph similarity. However, different authors have been presenting such a structure under different names, such as *clique* [114], *local structure* [101], *stars* [134], and *subgraph* [95].

1.5 Contributions

In this thesis, we propose a framework for geometric graph matching and similarity that has the following advantages over the current approaches.

1. It scales well, in terms of time and space, as graph size increases.
2. It is more tolerant to changes in graph structure, labeling information, and the spatial properties of the graphs.
3. It can be used to estimate both subgraph and common subgraph matchings.
4. It can be used to efficiently search a database for similar graphs taking into consideration geometric transformations.

In the following, we outline the accomplishments of this thesis.

Vertex Similarity
The basic block of any graph matching algorithm is to estimate the similarity between the vertices of different graphs. As a result, we start our contributions and propose a vertex distance function for labeled geometric graphs in 2D space, which we partially published in "Vertex Similarity - A Basic Framework for Matching Geometric Graphs" [10]. It has two main advantages. First, it is invariant to spatial transformations, such as translation, rotation, and scaling. Second, it handles labeling information that is assigned to the vertices and edges.

Geometric Graph Matching: A Probabilistic Approach
We propose a probabilistic graph matching algorithm, which is tolerant to the differences between a two given graphs. We partially introduced this algorithm in

our paper "Geometric Graph Matching and Similarity: A Probabilistic Approach" [9]. An initial match between two graphs is estimated using our vertex distance function. Then, the expectation maximization (EM) technique is used to iteratively improve the match between two graphs. To utilize the EM technique, we formalize the graph matching problem as a maximum likelihood estimation (MLE) problem. We propose a density function that combines both the similarity of the vertices and the structure of the graphs.

Efficient Geometric Graph Matching Using Vertex Embedding

We propose a graph matching algorithm that scales well as graph size increases. First, we propose a vertex embedding approach to convert a graph into a multi-set of vectors. As a result, the distance between two vertices is efficiently estimated by the Euclidean distance. Second, based on vertex embedding, we propose a graph matching algorithm. Initially, highly similar vertices, which are called *anchors*, are selected to initialize the match between two graphs. Then, the match is improved iteratively using the structure of both graphs. For this, we propose a *probabilistic voting scheme*, where the vertices of the match vote to quantify the similarity of different vertices from both graphs. We published the general embedding framework and some preliminary results in "Efficient Geometric Graph Matching Using Vertex Embedding" [8].

Geometric Graph Similarity Search

The final contribution of this thesis is a two layers indexing structure to efficiently search a database for similar graphs. The first layer efficiently retrieves vertices highly similar to the vertices of the query graph. For this, we propose to index the vertices of the graphs by an *R-tree* after embedding them in a higher-dimensional space using our proposed embedding scheme. Then, the set of candidate similar vertices is used to generate a set of candidate similar graphs. This is accomplished by using both an *inverted index* that maps each vertex to its graph and a lower bound distance function that utilizes the set of candidate similar vertices. The second layer of our indexing structure consists of two pruning approaches to further reduce the size of the set of candidate similar graphs.

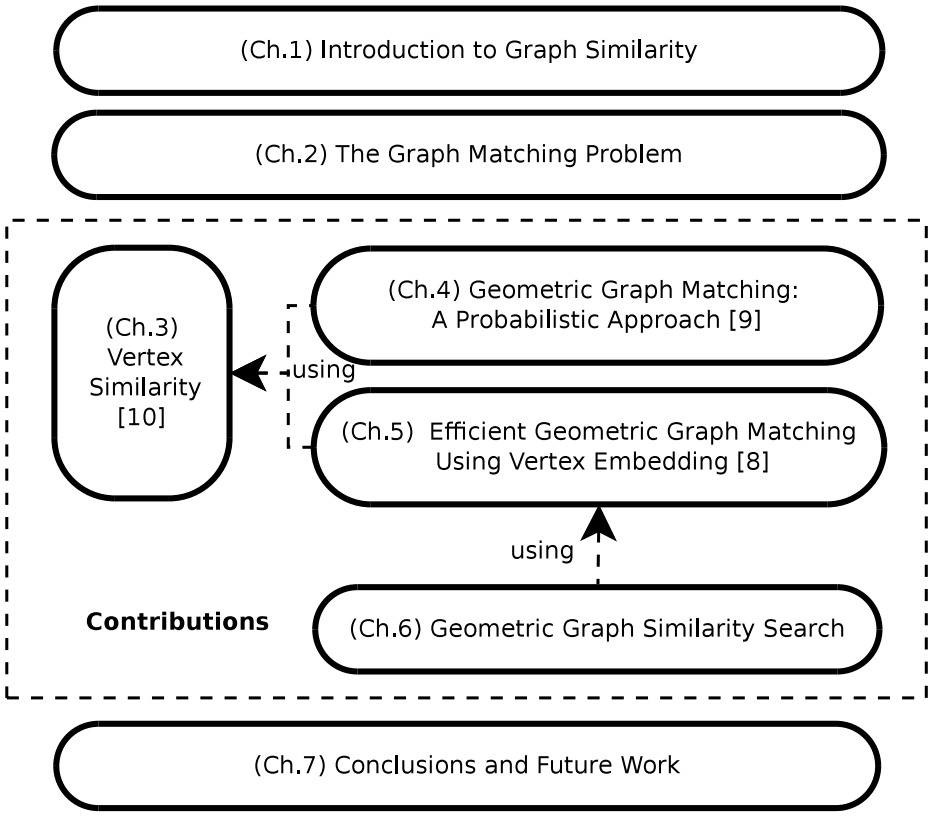


Figure 1.2: Graphical overview of the structure of this thesis. Four contributions are added to the research community, which are outlined inside the dotted rectangle.

1.6 Structure of the Thesis

In the following, we outline the structure of this thesis, which is also illustrated in Figure 1.2. Chapter 2, first, introduces general concepts related to graph theory. Then, the problem of graph matching and the different notions of graph similarity are formalized. We detail our contributions in Chapters 3, 4, 5, and 6. We propose the vertex distance metric in Chapter 3. Based on this, we propose the probabilistic graph matching algorithm in Chapter 4. After that, in Chapter 5, we propose a embedding scheme and a scalable graph matching algorithm. We utilize our approaches to vertex similarity and embedding to propose a indexing structure in Chapter 6. At the end, Chapter 7 summarizes our findings and suggests open issues for future studies.

Chapter 2

The Graph Matching Problem

Searching for similar objects is a vital task in many application domains, such as biology, chemistry, computer vision, and pattern recognition. For such applications and many more, graphs are used to model complex objects and relationships. Therefore, the problem of finding similar objects is transformed into the problem of finding similar graphs. Graph similarity, in general, captures both the similarity of the labels, which are assigned to the vertices and edges, and the structural similarity of the graphs.

The latter notion of similarity is normally formalized as the graph matching problem, which searches for correspondences between the vertices of two graphs such that the mapped vertices have similar structure. Graph matching can be classified into two main categories. The first category is the *exact graph matching problem*, where a vertex from one graph is mapped to another vertex from another graph only when the two vertices are identical in terms of structure and label. The second category is the *inexact graph matching problem*, which allows the mapping of any two vertices with a penalty cost that describes the differences in the structure and label.

In this chapter we give the foundations of the graph matching problem, which is typically used to measure the structural similarity of different graphs. We first introduce some preliminary definitions, and then we detail the graph matching problem and discuss common approaches and techniques.

2.1 Preliminary Definitions

In this section we give preliminary definitions related to the graph-based representation, which are considered the foundations of the rest of this thesis [38].

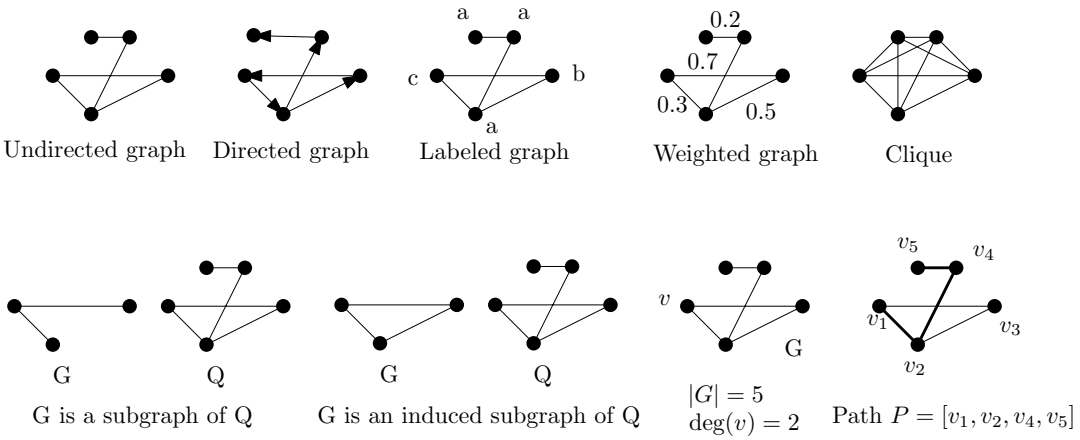


Figure 2.1: Examples for the different concepts related to the graph representation.

A graph $G = (V, E)$ is represented by a non-empty finite set of vertices V, also called nodes, and a finite set of edges E that connect the vertices. The vertex set of a graph Q is denoted by $V(Q)$. In the same way, we refer to the edge set of Q as $E(Q)$. Notice that V and E are used as functions of the graph name to denote its vertex and edge sets, respectively. A vertex v in a graph G is referred to as $v \in V(G)$ or $v \in G$. An edge e that belongs to a graph G is denoted by $e \in E(G)$ or $e \in G$. An edge $e_{ij} = (v_i, v_j)$ connects the two vertices v_i and v_j. For an *undirected graph*, both edges $e_{ij} = (v_i, v_j)$ and $e_{ji} = (v_j, v_i)$ refer to the same edge. On the other hand, an edge in a *directed graph* is represented as an ordered pairs such that edge $e_{ij} = (v_i, v_j)$ is different from $e_{ji} = (v_j, v_i)$ as shown in Figure 2.1.

The *order* of a graph G represents the number of vertices in that graph and is denoted by $|G|$. On the other side, the *size* of G represents the number of edges and is denoted by $\|G\|$. The definitions of both graph order and size are well separated in the domain of graph theory [38]. However, in the graph data mining literature, graph size is used to represent the number of vertices in a graph [103]. In this thesis, we use graph size to denote the number of vertices in a graph.

For a graph G, two vertices v_i and v_j are *adjacent* or *direct neighbors* if there is an edge $e_{ij} = (v_i, v_j) \in E(G)$ connecting or *joining* both of them. The two vertices v_i and v_j are *incident* with the edge e_{ij} and also they are its *ends*. On the other side, two edges are *adjacent* if they are connected to the same vertex. The *degree* (valency) of a vertex v_i is denoted by $deg(v_i)$ and equals the number of edges incident to that vertex. A vertex of degree zero is called *isolated vertex*. A vertex of degree one is called a *pendant vertex*.

Definition 2.1. (Undirected Labeled Graph) *An undirected labeled graph* $G = (V, E, l)$ *consists of a set of vertices* V*, a set of edges* $E \subseteq \{V \times V\}$*, and a labeling function* $l : \{V \cup E\} \rightarrow \{\Sigma \cup \varepsilon\}$ *assigning a label to every vertex and every edge from a label alphabet* Σ*.* ε *is the null label denoting unlabeled vertices or edges.*

The above definition does not restrict the label alphabet Σ. For instance, $\Sigma = \{a, b, \ldots, z\}$ assigns each vertex or edge a discrete label from the English alphabet, $\Sigma = \mathbb{N}$ assigns positive integer labels to the vertices and edges. Similarly, $\Sigma = \mathbb{R}^d$ represents labels of real-valued vectors of dimension d. A *weighted graph*, as shown in Figure 2.1, assigns labels to only its edges such that $\Sigma = \mathbb{R}$.

Definition 2.2. (Subgraph) *Let* $G = (V, E, l_g)$ *and* $Q = (U, T, l_q)$ *be two undirected labeled graphs. Graph* G *is considered a subgraph of* Q*, denoted* $G \subseteq Q$*, if* $V \subseteq U$ *and* $E \subseteq T$*.*

The subgraph notion captures the case when one graph is contained in another. If G is a subgraph of Q, then Q is called a *supergraph* of G. Another variation of the notion of subgraph is *induced subgraph*. Graph $G = (V, E, l_g)$ is an induced subgraph of another one $Q = (U, T, l_q)$ when $V \subseteq U$ and $E = T \cap V \times V$. If G is an induced subgraph of Q and there are the vertices $\{v_i, v_j\} \subseteq V$ and their corresponding vertices $\{u_k, u_l\} \subseteq U$ such that $e_{kl} = (u_k, u_l) \in T$, then edge $e_{ij} = (v_i, v_j)$ must be in E. Figure 2.1 shows the difference between the subgraph and the induced subgraph concepts.

A *clique* is a (sub)graph, such that any two vertices are connected by an edge. A *path* is a special type of (sub)graphs defined as $P = (V, E)$ such that $V = \{v_1, v_2, \ldots, v_{|P|}\}$ and $E = \{(v_1, v_2), (v_2, v_3), \ldots, (v_k, v_{k+1}), \ldots, (v_{|P|-1}, v_{|P|})\}$. Normally, a path is represented by its sequence of vertices as $P = [v_1, v_2, \ldots, v_{|P|}]$. The *length* of a path represents the number of edges it has. The vertices v_1 and $v_{|P|}$ are *linked* or *connected* by the path P and are called its *ends*. In a weighted graph, the shortest path between two vertices is the path of the minimum sum of edge weights. For unweighted graphs, a shortest path is the one of the minimum number of edges. A *self loop* is an edge that connects a vertex to itself. *Parallel edges* denote several edges connecting the same two vertices. A graph with no self loops and parallel edges is called a *simple graph*.

A product graph, or an *association graph* as called by the pattern recognition community, defines the compatibility between the structure of two graphs and is formalized as follows:

Definition 2.3. (Product Graph) *Given the two graphs* $G_1 = (V_1, E_1, l_1)$ *and* $G_2 = (V_2, E_2, l_2)$*, their product graph is* $G = (V, E, l)$ *such that:*

- $V = (v_i, v_k) \in V_1 \times V_2$ such that $l_1(v_i) = l_2(v_k)$
- $E = \{((v_i, v_k), (v_j, v_l)) \in V \times V\}$, *such that:*
  - $e_{ij} = (v_i, v_j) \in E_1 \wedge e_{kl} = (v_k, v_l) \in E_2 \wedge l_1(e_{ij}) = l_2(e_{kl}))$, *or*
  - $e_{ij} = (v_i, v_j) \notin E_1 \wedge e_{kl} = (v_k, v_l) \notin E_2$.

The vertices of the product graph represent pairs of vertices from the original graphs, and the edges represent the similarity between the edges of the original two graphs.

A graph is called geometric when its vertices have coordinates in some d-dimensional space, which is formally defined as follows:

Definition 2.4. (Geometric Graph) *A labeled undirected geometric graph* $G = (V, E, l, c)$ *consists of a finite set of vertices* V*, a finite set of edges* $E \subseteq \{V \times V\}$*, a labeling function* $l : \{V \cup E\} \rightarrow \{\Sigma \cup \varepsilon\}$*, assigning a label to every vertex and every edge from a label alphabet* Σ *or the null label* ε*, and a function* $c : V \rightarrow \mathbb{R}^d$*, assigning a coordinate in* $\mathbb{R}^d$ *to every vertex.*

The main difference between geometric and non-geometric graphs are the *spatial properties* of the graphs, which we define as follows:

Definition 2.5. (Spatial Property of Geometric Graph) *For a geometric graph, its spatial property means the coordinates of its vertices in addition to the lengths of the edges, which is computed for an edge using the Euclidean distance between the coordinates of its end vertices.*

The scope of this thesis considers simple undirected labeled geometric graphs, where the vertices have coordinates in some 2D space. For simplicity, refer to as geometric graphs. It is worth mentioning that we do not restrict a geometric graph to be planar, i.e., graphs that can be embedded in the plain such that edges intersect only at the vertices.

2.2 Graph Matching

Due to the flexibility of the graph data structure, computing the optimal solution to the graph similarity problem is a very difficult task. For example, there are $n!$ different permutations, i.e., representations, for a graph of n vertices. Such a lack of

a *canonical order* is the main problem that makes the task of computing the similarity between graphs a very hard one.

The graph similarity problem can be classified into two categories. The first one is the *graph comparison problem* and the second one is the *graph matching problem.*

Definition 2.6. (Graph Comparison Problem) *Given a space of graphs $\mathcal{G}$, the graph comparison problem searches for a function $d : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}$ such that $d(G, Q)$ estimates the distance between the two graphs G and Q.*

Several graph-based applications require finding why two graphs are similar and not only whether they are similar or not. Such a task is answered by identifying similar common (sub)graphs, which is formalized as the graph matching problem.

Definition 2.7. (Graph Matching Problem) *Given two graphs G and Q, the graph matching problem searches for correspondences between $V(G)$ and $V(Q)$ such that the mapped vertices have similar labels and similar connectivity.*

Given two graphs G and Q with their vertex sets V and U, respectively, a match between the two graphs can be represented as a function $f : V \rightarrow U$. $f(v_i) = u_k$ denotes that vertex $v_i \in V$ corresponds to vertex $u_k \in U$. Using the function f, the match between two graphs can be also represented as a matrix $M \in \{0, 1\}^{|V| \times |U|}$. An entry m_{ik} is an *assignment variable* indicating whether vertex v_i is being in correspondence with vertex u_k. m_{ik} is formally defined as:

$$m_{ik} := \begin{cases} 1, & \text{if } f(v_i) = u_k \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

under the conditions:

$$\forall k \in \{1, \ldots, |U|\}, \left(\sum_{i=1}^{|V|} m_{ik} \right) \leq 1, \text{ and} \tag{2.2}$$

$$\forall i \in \{1, \ldots, |V|\}, \left(\sum_{k=1}^{|U|} m_{ik} \right) \leq 1 \tag{2.3}$$

According to the previous two conditions, each vertex from G is matched to at most one vertex from Q and vise versa.

Graph matching finds mappings between the vertices and edges of one graph to the vertices and edges of another one as shown in Figure 2.2. In the case of labeled

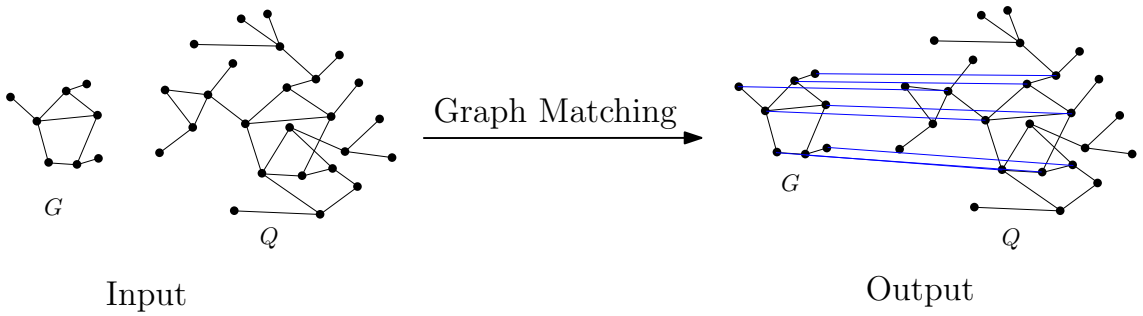


Figure 2.2: A graph matching algorithm finds the correspondences between the vertices of two graphs such that the mapped vertices are structurally similar. The match between G and Q is represented by the blue lines.

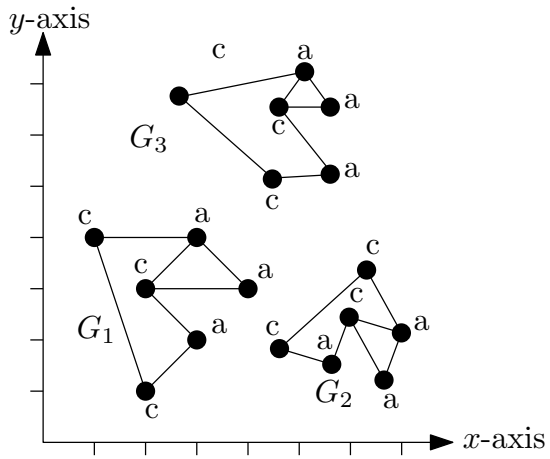


Figure 2.3: The spatial properties of both G_1 and G_2 are considered identical utilizing one geometric transformation. The spatial property of G_3 is similar to both G_1 and G_2 but not identical since there is no single geometric transformation that transforms their coordinates identical.

graphs, the correspondent vertices and edges must have similar labels. For unlabeled graphs, only the structure of the graphs is used for graph matching.

Definition 2.8. (Structural Similarity) *The structure of two graphs G and Q is similar based on a match M if edge $e_{ij} = (v_i, v_j) \in G$ is similar to edge $e_{kl} = (u_k, u_l) \in Q$ whenever $m_{jl} = 1$ and $m_{ik} = 1$.*

For unlabeled graphs, only the existence or absence of an edge defines the structural similarity. For labeled graphs, the labels of the two edges must be also similar.

The notion of *geometric graph similarity* considers the spatial properties of the graphs in addition to the similarity of the labels and structure. Normally, the Euclidean distance is used to measure the similarity between real-valued labels. However,

the Euclidean distance between the coordinates of different vertices does not capture the spatial similarity between geometric graphs. This is because two graphs are considered similar if there is a *geometric transformation*, which consists of a translation, rotation, and scaling, that transforms the coordinates of one graph identical to the coordinates of the other. Thus, two geometric graphs may be considered identical even though the Euclidean distances between the coordinates of their vertices are large.

To illustrate such a concept, we give an example of two geometric graphs G_1 and G_2 in Figure 2.3. A geometric transformation that consists of a rotation, translation, and scaling makes the coordinates of graph G_1 identical to the coordinates of graph G_2. As a result, the spatial properties of both graphs are considered identical. However, the Euclidean distance between the coordinates of G_1 and G_2, without considering any geometric transformation, gives the indication that their spatial properties are different.

In the following two sections we give general foundations and techniques related to the graph matching problem. In Section 2.3 we discuss concepts related to exact graph matching. After that, we detail the inexact graph matching problem in Section 2.4.

2.3 Exact Graph Matching

The exact graph matching problem is a strict notion of graph similarity, where two vertices are only matched if they are identical in terms of label and structure. In this section, we discuss three concepts related to the exact graph matching problem. In particular, we discuss *graph isomorphism*, *subgraph isomorphism*, and *maximum common subgraph matching*.

2.3.1 Graph Isomorphism

The isomorphism between two graphs is a one-to-one mapping between their vertices, therefor, two graphs that differ in the number of vertices are not isomorphic. Additionally, the structure of the two graphs must be identical.

Definition 2.9. (Graph Isomorphism) *Two graphs $G = (V, E, l_g)$ and $Q = (U, T, l_q)$ are called isomorphic or they posses an isomorphism if there is a bijective function $f : V \to U$ that satisfies the following:*

1. $\forall v_i \in V$, $l_g(v_i) = l_q(f(v_i))$

2. $\forall u_k \in U$, $l_q(u_k) = l_g(f^{-1}(u_k))$, *where* f^{-1} *is the inverse function of* f

3. $\forall e_{ij} = (v_i, v_j) \in E$, $\exists e_{kl} = (f(v_i), f(v_j)) \in T \wedge l_g(e_{ij}) = l_q(e_{kl})$

4. $\forall e_{kl} = (u_k, u_l) \in T$, $\exists e_{ij} = (f^{-1}(u_k), f^{-1}(u_l)) \in E \wedge l_q(e_{kl}) = l_g(e_{ij})$

Unfortunately, the complexity of the graph isomorphism problem is NP [98]. There is no known polynomial algorithm to solve it, neither is proven to be NP-complete. In the worst case, the computational complexity of any of the current solutions to the graph isomorphism problem is in general exponential with respect to graph size.

On of the most common techniques to check for graph isomorphism is tree search with backtracking [32]. Initially, a set of candidate mappings $C(v_i)$ is computed for each vertex $v_i \in G$. Vertex $u_k \in Q$ is added to the candidate set of $v_i \in G$ if they have identical labels and degrees. The search algorithm starts with an empty match, which is then expanded iteratively by adding new pairs of vertices from the two graphs. At an iteration, suppose the last pair of vertices that is added to the match be (v_i, u_k). The algorithm selects a new vertex v_j from G to be added to the match. It searches the candidate similar vertices $C(v_j)$ for a vertex $u_l \in Q$ such that mapping the two vertices is compatible with the structure of the two graphs and the vertices in the match. If no vertex from $C(v_j)$ passes the test, a backtrack is applied and the last pair of vertices that was added to the match, i.e., (v_i, u_k), is removed and a new mapping for vertex v_i is tested.

Since tree search with backtracking scales poorly with respect to graph size, several graph matching algorithms adopt different techniques to prune the search space and reduce the runtime. The graph matching algorithm of Ullmann [117] follows this technique. He propose a look-ahead technique to further prune the search space. It tests the compatibility between the unmatched vertices that are connected to the vertices in the match M. Such a pruning technique makes the best time complexity $O(n^3)$, where n is the size of a graph. Another search-based algorithm is the VF2 with best time complexity of $O(n^2)$ [33]. The search order of Ullmann's algorithm is random. However, VF2 requires that the new vertex to be added to the match is directly connected to one of the vertices in M. To prune the search space further, QuickSI has been proposed [111]. It utilizes a search order that prunes as much as possible from the search space and is guided by the frequency of the labels that are assigned to the graphs in a given database.

Another class of algorithms to solve graph isomorphism utilizes the concept of *canonical labeling*. A canonical form is used to represent all graphs that are considered

isomorphic. The challenge is that finding a canonical form for a graph is as difficult as the graph isomorphism problem itself. To prune the search space, such algorithms utilize the concept of *automorphisms*, which is an isomorphism between a graph and itself. The main prominent algorithms following this approach are Nauty [83], Bliss [64], and Traces [94]. A detailed comparison between the different algorithms is presented by McKay and Piperno [84].

2.3.2 Geometric Isomorphism

In the domain of geometric graphs, isomorphism tests for both structural isomorphism, which is discussed in the previous section, and geometric isomorphism.

Definition 2.10. (Geometric Isomorphism) *Given a bijection function f between the two geometric graph $G = (V, E, l_g, c_g)$ and $Q = (U, T, l_q, c_q)$, the two graphs are considered geometric isomorphic if there is a geometric transformation ϕ, which consists of a rotation, translation, and scaling, such that:*

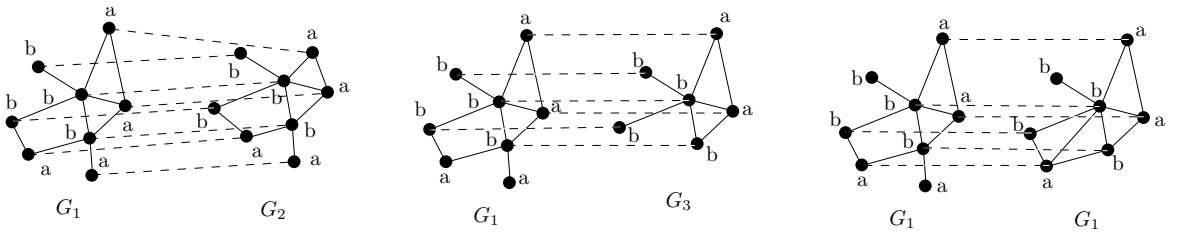
$$\forall v_i \in G : \phi(c_g(v_i)) = c_q(f(v_i)) \tag{2.4}$$

The notion of isomorphism is a very strict notion of similarity. To show how this restriction applies to geometric graphs, we refer to Figure 2.3. Even though graphs G_1 and G_3 are structural isomorphic, they are not geometric isomorphic. There is no *affine* transformation that transfers the coordinates of all vertices of G_1 to be identical to the coordinates of the vertices of G_3. However, there is a set of different affine transformations that transfers the coordinates of the vertices of G_1 identical to the coordinates of the vertices of G_3.

The aforementioned notion of geometric graph similarity is not practical for several scientific applications, such as chemoinformatics and bioinformatics. For such applications, the extracted geometric graphs have normally measurement errors with respect to the coordinates of the vertices. Kuramochi and Karypis [75] consider this problem and propose to relax the constraints in Equation 2.4 to consider a certain tolerance of error r as follows:

$$\forall v_i \in G :\| \phi(c_g(v_i)) - c_q(f(v_i)) \| \leq r \tag{2.5}$$

The above equation suggests that the Euclidean distance between the coordinates of two vertices, after applying the geometric transformation ϕ, must be less than or equal to a certain threshold r.



Graph Isomorphism | Subgraph Isomorphism | Maximum Common Subgraph

Figure 2.4: Different notions of exact graph matching.

The runtime complexity of solving the general graph isomorphism problem in NP [98]. However, solving the graph isomorphism problem between two geometric graphs runs in polynomial time [75]. This is feasible because the spatial property prunes the search space of the possible structural isomorphisms between the two graphs.

Approaching the geometric graph isomorphism consists of two steps: 1) finding all the geometric transformations that make the coordinates of two graphs identical, and 2) verifying the structural isomorphism for each computed transformation. Given the two geometric graphs $G = (V, E, l_g, c_g)$ and $Q = (U, T, l_q, c_q)$ in some 2D space, finding the match between their vertices based on just the spatial property can be considered as matching two point sets. A local coordinate frame is generated from any two vertices v_i and v_j in graph G. The x-axis is spanned by the vector $\vec{x}_{ij}$, which starts at point $c_g(v_i)$ and ends at point $c_g(v_j)$. The y-axis is defined by the vector $\vec{y}_{io}$, which starts at point $c_g(v_i)$ and is orthogonal to $\vec{x}_{ij}$. Thus, graph G has $|V|^2$ different local frames. To estimate a match between G and Q based on just the coordinates of the vertices, first, a local frame LF_{kl} is computed for Q utilizing the two vertices u_k and u_l. Then, for each local frame LF_{ij} from G, find the best match for each vertex from G utilizing the nearest neighbor Euclidean distance to the vertices of Q. The runtime complexity of this step is $O(|V||U|)$. As a result, finding the optimal match based on only the spatial property runs in $O(|V|^3|U|)$. After that, such a geometric isomorphism is tested for structural compatibility, which runs in $O(|E|)$ [7].

It is worth mentioning that polynomial solutions for the graph isomorphism problem exist for special graphs, such as graphs with unique labels [37], graphs of bounded degree [79], planner graphs [56], ordered graphs [61], and trees [5].

2.3.3 Subgraph Isomorphism

The difference between graph isomorphism and subgraph isomorphism is that the latter one considers graphs that differ in the number of vertices. However, the structure of the smaller graph must be identical to the structure of its correspondent subgraph from the larger one.

Definition 2.11. (Subgraph Isomorphism) *A graph G is subgraph isomorphic to another graph Q if there exist a subgraph $Q^{'} \subseteq Q$ such that G and $Q^{'}$ are isomorphic.*

In fact, the graph isomorphism is a special case of subgraph isomorphism when the two graphs have the same number of vertices. However, the computational complexity of solving the subgraph isomorphism problem is NP-complete [98]. This is because the matching algorithm searches for graph isomorphism between a smaller graph G and all subgraphs of size $|G|$ of another graph Q.

Most of the approaches to tackle the graph isomorphism problem, such as Ullmann [117] and VF2 [33], can be used as well to solve subgraph isomorphism. However, for subgraph isomorphism, they use a less than or equal criterion instead of only an equality. This relaxation is because the larger graph has more vertices and edges than the smaller one. As a result, two vertices $v_i \in G$ and $u_k \in Q$ are mapped to each others if $deg(v_i) \leq deg(u_k)$ and both vertices have the same label.

2.3.4 Maximum Common Subgraph

Unfortunately, subgraph isomorphism cannot estimate the similarity between two graphs in the case that the smaller graph is not identical to a subgraph of the bigger one. To solve this problem, a notion of graph similarity is proposed based on the concept of maximum common subgraph *mcs* [18].

Definition 2.12. (Maximum Common Subgraph (*mcs*)) *Given two graphs G and Q, their mcs is the maximum subgraph from G that is isomorphic to a subgraph from Q.*

The above definition implies several issues, in particular 1) the size of the maximum common subgraph between two graphs is less than or equal the size of the smaller of them, 2) there could be several maximum common subgraphs between a two given graphs, and 3) an exact match, i.e., (sub)graph isomorphism, is used between the *mcs* and both graphs. In the literature, the *mcs* problem is classified based on the

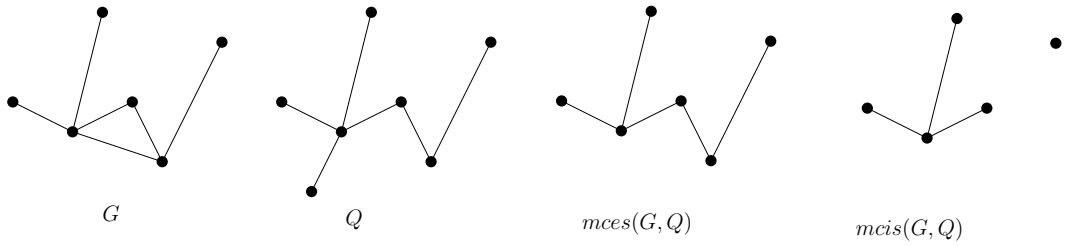


Figure 2.5: The maximum common induced subgraph *mcis* and the maximum common edge subgraph *mces* for the two graphs G and Q. Notice that a *mcs* could be disconnected.

definition of maximality, which is either maximality based on the number of vertices *(mcis)* or the number of edges *(mces)* as shown in Figure 2.5.

The computational complexity of finding the *mcs* between two graphs is NP-complete [48]. McGregor proposes a solution based on searching with backtracking [82]. However, most of the existing approaches to the *mcs* problem reduce it to the problem of finding a maximum clique in their *product graph* [12, 31, 43, 96], see Definition 2.3. Even though the problem of maximum clique detection is NP-complete [48, 67], a branch-and-bound technique is used to speed up and prune the search space [16, 72], which is considered faster than directly tackling the maximum common subgraph problem.

2.4 Inexact Graph Matching

The exact graph matching approaches map two vertices from two graphs only if their labels and structure are identical. However, such a strict matching criterion is impractical for several scientific applications. For instance, in image analysis, different scales of the same color are considered similar even though they are not identical. In chemoinformatic, two different atoms may have similar chemical reaction. In social network, two users may have similar interests even though they are not connected to the same friends. In these applications and many more, some error may be also introduced as a result of graph extraction, i.e., representing an object by the graph data structure. Thus, two similar objects can be modeled by two non-identical graphs.

The concept of *inexact graph matching problem* is introduced to estimate the similarity between graphs that differ in the number of vertices and structure [108]. It allows the mapping of any two vertices with a penalty reflecting the difference of their labels and structure [50].

Definition 2.13. (Inexact Graph Matching Problem) *Given two labeled undirected graphs* $G = (V, E, l_g)$ *and* $Q = (U, T, l_q)$*. Let the function* $f(v_i, u_k)$ *define the distance between* $l_g(v_i)$ *and* $l_q(u_k)$*. Also, let the function* $f(e_{ij}, e_{kl})$ *define the distance between* $l_g(e_{ij})$ *and* $l_q(e_{kl})$ *such that* $e_{ij} = (v_i, v_j) \in E$ *and* $e_{kl} = (u_k, u_l) \in T$*. Suppose a match between the two graphs is represented by a matrix* $M \in \{0, 1\}^{|V|\times|U|}$ *such that* $m_{ik} = 1$ *if* $v_i \in V$ *is matched to* $u_k \in U$ *and* 0 *otherwise. The optimal solution to the inexact graph matching problem is the matrix* M^* *such that:*

$$M^* = \arg\min_M \sum_i \sum_k f(v_i, u_k) m_{ik} + \sum_i \sum_j \sum_k \sum_l f(e_{ij}, e_{kl}) m_{ik} m_{jl}$$

subjected to the constraints:

$$\forall k \in \{1, \ldots, |U|\}, \sum_{i=1}^{|V|} m_{ik} \leq 1, \text{ and}$$

$$\forall i \in \{1, \ldots, |V|\}, \sum_{k=1}^{|U|} m_{ik} \leq 1$$

The first summation of the above objective function, i.e., the function needed to be minimized, captures the compatibility in the labeling information between the vertices of two graphs. The second summation captures the structural similar between the vertices. The above two constraints guarantee a one-to-one mapping between the vertices of two graphs. In addition to this, they allow a vertex not to be matched to any other vertex from the other graph.

Computing the optimal solution to the inexact graph matching problem is NP-hard [20, 134]. Although the optimal solution can be computed [17], it is impractical for scientific applications as it requires exponential runtime [20, 50].

In the following sections we discuss three approaches that give approximate solutions to the inexact graph matching problem. In Section 2.4.1, we discuss the graph edit distance approach, which is a generalization of the concept of string edit distance. Then, we detail the spectral graph matching technique in Section 2.4.2, which utilizes the spectra of the adjacency or the Laplacian matrices. Finally, we discuss the continuous optimization approach in Section 2.4.3, which iteratively improves the match between two graphs at consecutive steps.

2.4.1 Graph Edit Distance

The concept of graph edit distance was introduced by Sanfeliu and Fu to measure the similarity between graphs that differ in the number of vertices, graph structure, and labeling information [108]. Its main idea is to measure the number of modifications needed to make one graph identical to another. For this, *edit operations* are provided to quantify the penalty or the cost of changing a vertex or an edge to become identical to its counterpart vertex or edge, respectively.

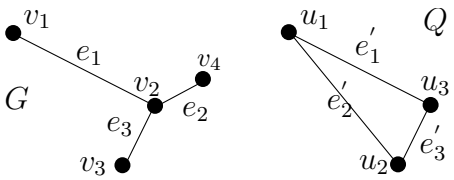
Three operations are used to edit the vertices and another three operations are used to edit the edges. Given the two vertices $v \in G$ and $u \in Q$, the vertex operations are: 1) a *substitution operation* $(v \rightarrow u)$ to change the label of vertex v to be identical to the label of u, 2) a *deletion operation* $(v \rightarrow \varepsilon)$ to delete or to remove vertex v from G, and 3) an *insertion operation* $(\varepsilon \rightarrow u)$ to insert or to add vertex u to graph Q, such that ε denotes the null non-existent vertex. Similarly, another three operations are defined for editing the edges of different graphs. The cost of an edit operation is indicated by the function $c(.)$. For instance the cost of vertex substitution is indicated by $c(v \rightarrow u)$. Normally, the costs of vertex and edge edit operations are provided by the user and customized to match the notion of similarity and labeling information for each application domain [103].

Definition 2.14. (Edit Path) *An edit path between two graphs is a sequence of edit operations that transfers the two graphs to be identical.*

To construct an edit path between two graphs, edge operations are triggered when their adjacent vertices are edited [103]. Given the vertices $v_i, v_j \in G$ and $u_k, u_l \in Q$, and the two edges $e_{ij} = (v_i, v_j)$ and $e_{kl} = (u_k, u_l)$, the edge operations are triggered according to the following rules:

1. Edge substitution. $(v_i \rightarrow u_k) \wedge (v_j \rightarrow u_l) \wedge (e_{ij} \in E(G)) \wedge (e_{kl} \in E(Q)) \rightarrow (e_{ij} \rightarrow e_{kl})$.
2. Edge deletion.
   (a) $(v_i \rightarrow u_k) \wedge (v_j \rightarrow u_l) \wedge (e_{ij} \in E(G)) \wedge (e_{kl} \notin E(Q)) \rightarrow (e_{ij} \rightarrow \varepsilon)$.
   (b) $(v_i \rightarrow \varepsilon) \wedge \{e_{ij} \in E(G)\} \rightarrow (e_{ij} \rightarrow \varepsilon)$, the same rule applies to graph Q.
3. Edge insertion. $(v_i \rightarrow u_k) \wedge (v_j \rightarrow u_l) \wedge \{e_{ij} \notin E(G)\} \wedge \{e_{kl} \in E(Q)\} \rightarrow (\varepsilon \rightarrow e_{ij})$

These rules ensure that the edit path between two graphs is structurally compatible. However, there is an exponential number of different edit paths by which a graph



$$P_1 = ((v_3 \to \varepsilon), (e_3 \to \varepsilon), (v_1 \to u_1), (v_2 \to u_2), (e_1 \to e_2'), (v_4 \to u_3), (e_2 \to e_3')(\varepsilon \to e_1'))$$
$$P_2 = ((v_4 \to \varepsilon), (e_2 \to \varepsilon), (v_1 \to u_1), (v_2 \to u_3), (e_1 \to e_1'), (v_3 \to u_2), (e_3 \to e_3'), (\varepsilon \to e_2'))$$
$$P_3 = ((v_1 \to \varepsilon), (v_2 \to \varepsilon), (v_3 \to \varepsilon), (v_4 \to \varepsilon), (e_1 \to \varepsilon), (e_2 \to \varepsilon), (e_3 \to \varepsilon),$$
$$(\varepsilon \to u_1), (\varepsilon \to u_2), (\varepsilon \to u_3), (\varepsilon \to e_1'), (\varepsilon \to e_2'), (\varepsilon \to e_3'))$$

Figure 2.6: Different edit paths between the two graphs G and Q.

can be made identical to another. For example, Figure 2.6 shows three different edit paths between the two unlabeled graphs G and Q. Assuming that we assign a cost of 0 for each of the vertex and edge substitutions and a cost of 1 for the remaining edit operations, then, the edit paths P_1 and P_2 have the same cost of 3. Whereas, the cost of edit path P_3 is 13, which is created by first deleting all the vertices and edges of graph G and then inserting all the vertices and edges of Q.

Since different edit paths have different costs, the graph edit distance is defined based on the one with the least cost.

Definition 2.15. (Graph Edit Distance) *Given the two graphs G and Q, the edit distance between them $ged(G, Q)$ is defined as:*

$$ged(G, Q) := \min_{P_i \in \mathcal{P}(G,Q)} cost(P_i)$$

where $\mathcal{P}(G, Q)$ is the set of all edit paths between G and Q and $cost(P_i)$ is the sum of the costs of all edit operations that are included in the edit path P_i.

Based on this definition, the match between two graphs is defined by the edit path with the least cost, and the distance between the two graphs is the cost itself. A graph similarity *metric* can be defined based on the edit distance concept. A graph distance function $d(G, Q)$ is considered metric if it satisfies the following conditions:

1. $d(G, Q) \geq 0$: *non-negativiy*
2. $d(G, Q) = 0$ if and only if G = Q: *identity of indiscernibles*
3. $d(G, Q) = d(Q, G)$: *symmetry*

4. $d(G, Q) \leq d(G, C) + d(C, Q)$: *triangle inequality*

Conditions 1 and 2 ensure that the distance is always positive or zero between an object and itself. Condition 3 means that the order of the variables does not affect the result of the function. The last condition ensures the least distance between two graphs. Based on Sanfeliu and Fu [108], the graph edit distance is a metric function when the following conditions are satisfied:

1. The cost of each edit operation is computed using a metric function.

2. For any two vertices v_i and v_j having identical labels, $c(v_i \rightarrow \varepsilon) = c(\varepsilon \rightarrow v_j)$.

3. For any two edges e_{ij} and e_{kl} having identical labels, $c(e_{ij} \rightarrow \varepsilon) = c(\varepsilon \rightarrow e_{kl})$.

Conditions 2 and 3 ensure that the deletion cost equals the insertion cost for edges or vertices having identical labels.

The graph edit distance is very flexible to handle any type of labeling information for the vertices and edges. However, the complexity of computing the exact solution to the graph edit distance is NP-hard [134]. Bunke and Allermann [17] use the well-known A^* algorithm [52] to find the exact graph edit distance. This approach follows a tree search paradigm such that intermediate nodes in the search tree represent partial edit paths between the two graphs, and leaves represent full edit paths.

Although the A^* algorithm finds the optimal solution, it has an exponential computational complexity. As a result, several techniques are used to give approximate solutions. The A^* algorithm is also used to approximate the graph edit distance [88]. Instead of keeping all possible partial paths from a certain node in the search tree, only the k least cost are kept. Following this, a large area of the search space can be pruned leading to a fast searching algorithm. The graph edit distance is formalized as a *binary linear programming* (BLP) problem by Justice and Hero [65]. Since the complexity of solving BLP problems is NP-complete [48], the authors approximate the solution by providing upper and lower bounds to the exact solution. Given that the graph size is n, the lower bound to the BLP problem is computed in $O(n^7)$ using the *interior point method* [89]. On the other side, the upper bound is computed in $O(n^3)$ by utilizing the Hungarian algorithm [90], as will be detailed later.

Approximate solutions to the graph edit distance are typically computed by neglecting the overall structure of the graphs. Only the structure in the *neighborhood* of a vertex is used to match that vertex to another one. For this, a *feature vector* is extracted from the labels of the neighborhood of each vertex. Then, approximate

solutions are computed by solving the *assignment problem* between the features of two graphs, which is formalized as the *bipartite graph matching problem*. In this approach, three questions must be answered, 1) how to define the neighborhood of a vertex?, 2) how to compute the distance between two features?, and 3) how to solve the assignment problem?

The structure and labeling information of the neighborhood of a vertex is used to represent its feature. The neighborhood for a vertex can defined in several ways. For instance, the vertex and its incident edges [139], in addition to the previous, the neighboring vertices are included [63, 69, 101, 104, 134], a spanning tree with a certain depth [123], and simple paths up to a certain length [131].

Once the neighborhood is defined, the labels assigned to the vertices and edge compose the feature of that vertex. Different distance functions are used in the literature to compute the distance between the features of two vertices, such as the Manhattan distance [63], the Euclidean distance [134], the Heterogeneous Euclidean Overlap Metric [62], which is similar to the Euclidean distance but handles features composed of multiple data types.

After defining the features and their distance function, the graph edit distance is approximated by first building a *distance matrix* between the features of two graphs, then, solving the *assignment problem* [69, 101, 104, 134]. The latter problem represents the task of selecting a minimum cost mapping between a group of workers and a group of jobs. The assignment problem is also referred as the weighted complete bipartite graph matching problem. The solution to the assignment problem is then the match with the least cost in such a bipartite graph.

Definition 2.16. (Bipartite Graph) *A weighted complete bipartite graph is defined as* $G_{BP} = (\{V_1 \cup V_2\}, E, w)$*, where* $\{V_1 \cup V_2\}$ *is a finite set of vertices such that* $V_1 \cap V_2 = \emptyset$*.* $E = V_1 \times V_2$ *is a finite set of undirected edges, and the function* $w : E \rightarrow \mathbb{R}$ *assigns a real-valued weight to each edge.*

For a graph to be bipartite, its vertex set must be divided into two disjoint vertex classes V_1 and V_2, such that there is no edge connecting two vertices from the same vertex class. Also, for a bipartite graph to be complete, there has to be an edge from every vertex in V_1 to some vertex in V_2, and vise versa. We refer to the weighted complete bipartite graph as the bipartite graph.

Definition 2.17. (The Bipartite Graph Matching Problem) *For a bipartite graph* $G_{BP} = (\{V_1 \cup V_2\}, E, w)$*, a match is a disjoint-set of edges* $E' \subset E$*, such that no two edges in* E' *share the same vertex. For a match to be minimum, it*

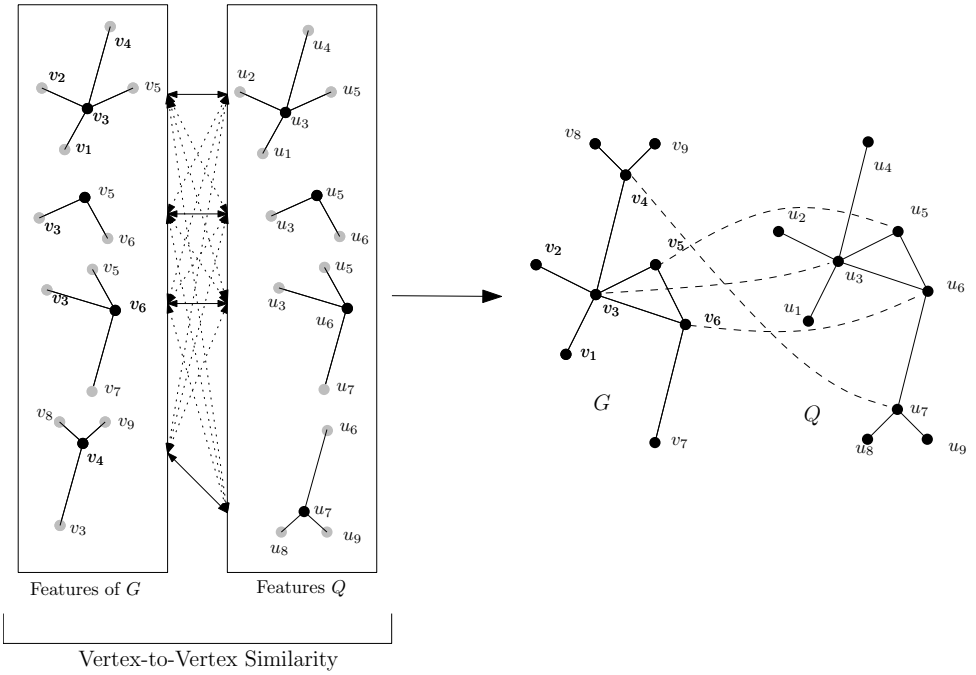


Figure 2.7: The solution to the assignment problem is considered as a suboptimal solution to the graph matching problem. Vertex $v_4 \in G$ must be mapped to $u_4 \in Q$. However, it is mapped instead to $u_7 \in Q$. For simplicity pendant vertices are not shown.

has to minimize the overall weight assigned to its edges, $\sum_{i=1}^{n} w(e_i)$, $e_i \in E'$, *and* $n = \min\{|V_1|, |V_2|\}$.

To solve the bipartite matching problem, the Hungarian algorithm is used [90], which was initially proposed by Kuhn [74] based on the work of the Hungarian mathematicians Dánes Kőnig and Jenő Egerváry. Such an algorithm has been reviewed and improved by Munkres [85]. After that the algorithm is referred as *Kuhn-Munkres' algorithm* or shortly *Munkres' assignment*. The fastest implementation of Munkres' algorithm runs in $O(n^3)$ where n is the size of the bipartite graph, which was proposed by Edmonds and Karp [44]. In this thesis, we use the Hungarian algorithm and Munkres' algorithm interchangeably to refer to the solution of the assignment problem.

It is worth mentioning that Munkres' algorithm computes the optimal solution to the assignment problem. However, such a solution is considered as an approximate solution to the graph edit distance. A match between two graphs based on such an approach may be structurally incompatible. This means that two direct neighboring vertices in one graph may be mapped to non-direct neighboring vertices in the other

graph, as shown in Figure 2.7. To be more specific, in the figure, vertex $v_4 \in G$ is mapped to vertex $u_7 \in Q$ since their neighborhoods are similar. However, when the overall graph structure is considered, vertex v_4 should be mapped to $u_4 \in Q$.

The concept of edit distance for geometric graphs is formalized by Cheong *et al.* [26]. Even though the geometric graph isomorphism problem is solved in polynomial time, they prove that the graph edit distance for geometric graphs is NP-hard. As a result, they propose an approximate solution that runs in cubic time complexity with respect to the graphs size. First, a feature vector is extracted for each vertex from a graph, which represents the shortest path distances between that vertex a set of *landmark* vertices. To select the set of landmarks, they propose to use four extreme vertices in the boundaries of the graph, i.e., peripheral vertices. Based on the features, a vertex-to-vertex distance matrix is created utilizing the Manhattan distance. Then, an approximate solution to the geometric graph edit distance is estimated by solving the assignment problem. Instead of using the Hungarian algorithm, the authors propose to use the *Earth Mover's Distance (EMD)* algorithm [106]. It is used to find the minimum cost of moving piles of earth so that to fill a group of holes taking into consideration that a pile can be split into several holes.

2.4.2 Spectral Graph Matching

The graph edit distance provides a flexible approach that can handle different types of labeling information that is assigned to the vertices and edges. However, it performs poorly in the case of unlabeled graphs. In other words, in the case of matching two graphs based on only their structure, the graph edit distance looses its power. This is because the features extracted from the neighborhoods of the vertices have very poor selectivity power. In this section, we discuss the second family of algorithms to solve the inexact graph matching problem, which is build based on ideas from *spectral graph theory* [30]. In the following, we give some preliminary definitions, then we discuss how graph spectra can be used for graph matching.

The structure of a graph can be represented in different ways using different connectivity matrices, which includes mainly the adjacency matrix, the Laplacian matrix, and the normalized Laplacian matrix.

Definition 2.18. (Adjacency Matrix) *For an undirected unlabeled graph $G = (V, E)$, its adjacency matrix $A \in \{0,1\}^{|G|\times|G|}$ is a square matrix of size $|G| \times |G|$, such that an entry a_{ij} indicates if vertex v_i is a direct neighbor of v_j. Formally, it is defined as:*

$$a_{ij} := \begin{cases} 1, & \text{if } e = (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases} \tag{2.6}$$

The adjacency matrix is an approach to represent the structure of a graph. For a weighted graph, an entry of its adjacency matrix a_{ij} represents the weight of edge $e = (v_i, v_j)$.

Definition 2.19. (Degree Matrix) *For an undirected unlabeled graph $G = (V, E)$, its degree matrix D is a square matrix of size $|G| \times |G|$, such as an entry d_{ij} is defined as:*

$$d_{ij} := \begin{cases} deg(v_i) & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

The degree matrix has the degrees of the vertices on the diagonal and zero otherwise.

Definition 2.20. (Laplacian Matrix) *For an undirected unlabeled graph $G = (V, E)$, its Laplacian matrix L is a square matrix of size $|G| \times |G|$, such that:*

$$l_{ij} := \begin{cases} deg(v_i) & \text{if } i = j \\ -1, & \text{if } e = (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases}$$

According to this definition, the Laplacian matrix stores the degree of the vertices on the diagonal, zero when there is no edge between the two vertices that are indexed by i and j, and a value of -1 in the case that the two vertices are connected by an edge. The Laplacian matrix can be easily computed using the adjacency matrix and the degree matrix of a graph as $L = D - A$.

Definition 2.21. (Normalized Laplacian Matrix) *For an undirected unlabeled graph $G = (V, E)$, its normalized Laplacian matrix $\mathcal{L}$ is a square matrix of size $|G| \times |G|$, such that:*

$$l_{ij} := \begin{cases} 1 & \text{if } i = j \\ \frac{-1}{\sqrt{deg(v_i)deg(v_j)}}, & \text{if } e = \{v_i, v_j\} \in E \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

One can compute the normalized Laplacian matrix from the Laplacian matrix or from the adjacency matrix according to the following relationship:

$$\begin{aligned}\mathcal{L} &= D^{-\frac{1}{2}} L D^{-\frac{1}{2}} \\ &= I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}\end{aligned} \tag{2.8}$$

where I is the identity matrix and $D^{-\frac{1}{2}}$ is a diagonal matrix such that an entry on the diagonal is defined as $-\frac{1}{\sqrt{deg(v_i)}}$.

In addition to unlabeled graphs, the previous matrices can also represent weighted graphs such that the entries store the weights that are assigned to the edges of a graph. For example, an entry in the Laplacian matrix L of a weighted graph $G = (V, E, w)$ with a weighting function $w : E \to \mathbb{R}$ is defined as:

$$l_{ij} := \begin{cases} \sum\limits_{v_k \in N(v_i)} w(e = (v_i, v_k)) & \text{if } i = j \\ -w(e), & \text{if } e = (v_i, v_j) \in E \\ 0, & \text{otherwise} \end{cases} \tag{2.9}$$

The spectral graph matching algorithms utilize the concept that graph structure is highly related to the Eigenvalues/vectors of its matrix representations. A vector x is considered as an Eigenvector of a square matrix A if it satisfied the following equations

$$Ax = \lambda x \tag{2.10}$$

where λ is the Eigenvalue corresponding to x. According to the previous equation, an Eigenvector does not change its direction when multiplied by A, its length is only changed to become λx. The Eigenvalues of a matrix A are the roots of its *characteristic polynomial* $p(\lambda)$, which is defined as:

$$p(\lambda) = det(A - \lambda I) \tag{2.11}$$

where $det(.)$ is the determinate of a matrix. Based on this, the Eigenvalues are the solution of the equation $p(\lambda) = 0$. The Eigenvector x_i that corresponds to the Eigenvalue λ_i is computed by solving $(A - \lambda_i I)x_i = \mathbf{0}$. In the following we give some definitions and properties for the Eigenvalue/vectors of a matrix.

Definition 2.22. (Spectrum) *The spectrum of the graph is the Eigenvalues for its adjacency or (normalized)Laplacian matrices.*

Definition 2.23. (Algebraic Multiplicity) *For an Eigenvalue λ_i, its algebraic multiplicity is the multiplicity of λ_i as a root of the characteristic polynomial.*

Definition 2.24. (Geometric Multiplicity) *For an Eigenvalue λ_i, its geometric multiplicity is the maximal number of linearly independent Eigenvectors corresponding to λ_i.*

Definition 2.25. (Principal Eigenvector) *It is the Eigenvector that correspondence to the maximum Eigenvalue of a matrix.*

Definition 2.26. (Eigendecomposition) *A symmetric matrix can be decomposed as:*

$$A = U\Lambda U^T$$

such that Λ is a diagonal matrix of the Eigenvalues, U is the matrix of Eigenvectors, and U^T is the transpose matrix of U.

Proposition 2.1. *If the Eigenvalues for a symmetric matrix A are strictly ordered, i.e., $0 \leq \lambda_1 < \lambda_2 < \cdots < \lambda_n$, then matrix A has n unique Eigenvectors up to a sign.*

The aforementioned proposition means that each Eigenvalue has a geometric multiplicity of one. Notice that, if x is an Eigenvector for a matrix, then $-x$ is also considered an Eigenvector of that matrix, which is denoted by a unique Eigenvector up to a sign in the previous definition.

Definition 2.27. (Rayleigh Quotient) *The Rayleigh quotient for the symmetric matrix A is $R(x)$ such that:*

$$R(x) = \frac{x^T A x}{x^T x}, \text{ where } x \neq \mathbf{0}$$

Theorem 2.1. *Supposed that λ_n and x_n are the maximum Eigenvalue and its Eigenvector (principal Eigenvector) for the symmetric matrix A, respectively, then:*

$$x_n = \arg\max(R(x))$$
$$\lambda_n = R(x_n)$$

In other words, the principal Eigenvector x_n of a matrix A is the vector that maximizes the Rayleigh quotient of A. Such a theorem plays a key role in tackling the graph matching problem, as will be shown later.

Definition 2.28. (Cospectrality of Graphs) *Two graphs are called cospectral if they are not isomorphic and they have the same spectrum.*

If two graphs are isomorphic then they should have the same Eigenvalues. However, cospectrality means that two graphs could be not isomorphic but they share the same Eigenvalues [126]. As a result, the focus is on the Eigenvectors of the connectivity matrices to tackle the graph matching problem.

In the following, we discuss three approaches to utilize the spectra of the connectivity matrices for graph matching. The first approach uses the spectra of both graphs, the second one uses the spectra of an affinity matrix, which is a kronecker product of their adjacency matrices, and the third approach uses the spectra of a vertex-to-vertex similarity matrix between two graphs.

Most of the following approaches have been applied to geometric graphs, although this is not explicitly mentioned. Several authors use geometric graphs in their evaluation. They utilize the weighted adjacency or Laplacian matrices, which store the lengths of the edges.

• Spectra of Both Graphs

The first to use the spectra of the connectivity matrices for graph matching is Umeyama [118]. The main idea of his approach is to embed the vertices of a graph in its Eigenspace, which is the space spanned by the Eigenvectors of the adjacency matrix of a graph. Then, each vertex can been seen as a point in a higher dimensional space such that each dimension corresponds to an Eigenvector. After that, he proposes to use the Hungarian algorithm to approximate the graph matching problem. Given the two graphs G and $G^{'}$ with their adjacency matrices A and $A^{'}$, where $|G| = |G^{'}| = n$, the solution to the graph matching problem between the two graphs is the permutation matrix P^{*}, such as:

$$\min_{P \in \mathcal{P}} \parallel PAP^{T} - A^{'} \parallel^{2} \tag{2.12}$$

such that $\mathcal{P}$ is the space of all permutation matrices of size $n \times n$. The above equation minimizes the Frobenius norm of the difference between PAP^{T} and $A^{'}$. Note that a permutation matrix is a double stochastic matrix representing a one-to-one mapping between the vertices of two graphs, also it is an orthogonal matrix.

The approximate solution to the graph matching problem that is proposed by Umeyama depends on the assumptions that 1) both A and $A^{'}$ have distinct Eigenvalues that can be ordered, and 2) the domain of the permutation matrices is extended to the domain of orthogonal matrices $\mathcal{Q}$. As a result, minimizing $\parallel QAQ^{T} - A^{'} \parallel^{2}$,

$Q \in \mathcal{Q}$, is achieved by Q^*, such that $Q^* = U^{'} S U^T$, $U^{'}$ and U are the matrices of the Eigenvectors of $A^{'}$ and A, respectively, S is a diagonal matrix such that $s_{ii} \in \{-1, 1\}$. Notice that the matrix Q^* is not a permutation matrix since an entry $q_{ij} \in [0, 1]$. To solve the sign ambiguity in the eigendecomposition of both matrices, which means to consider an Eigenvector λ_i or its negative $-\lambda_i$, Umeyama's heuristic is to use the absolute values of the entries in both U and $U^{'}$. As a result, an entry in Q^* can be seen as the cosine similarity between two vertices based on their coordinates in U and $U^{'}$. To get a zero-one discrete values from the Q^*, i.e., to recover a permutation matrix from it, Umeyama proposes to use the Hungarian algorithm to select the best match from Q^*.

The above approach has two limitations. First, it handles only graphs with the same number of vertices. Second, the Eigenvalues/vectors are not unique, which makes their order not reliable for graph matching. Notice that eigendecomposition creates two different Eigenspaces for a two given graphs. So, before computing the similarity between the vertices of two graphs, the dimensions of the Eigenspaces must be first mapped. In other words, it is required to determine which dimension of the first Eigenspace corresponds to which from the other Eigenspace. For that, Umeyama uses a total ordering of the dimensions of an Eigenspace by sorting them based on the magnitude of the Eigenvalues. However, in practice, Eigenvalues may have algebraic multiplicity and geometric multiplicity leading to an incorrect mapping of the dimensions of two Eigenspaces.

In addition to the adjacency matrix, the spectra of the Laplacian are used for graph matching [71]. To match two graphs G and Q with sizes m and n, respectively, first the Eigenspaces of both graphs are truncated by keeping only k Eigenvectors from both graphs that correspond to the largest k Eigenvalues where $k < \min\{m, n\}$. To align the Eigenspaces of the two graphs, the authors use the histograms of the eigenfunctions, i.e., a histogram of the values of each Eigenvector. The main idea of such an approach is that the histograms are invariant to vertex permutation.

Once the similarity between the Eigenvectors of two graphs is computed, the Hungarian algorithm is used to estimate the best match between the dimensions of the two Eigenspaces. After that, the similarity of two vertices is computed as the cosine similarity of their values across different Eigenvectors. To estimate the match between the two graphs, they proposed to use the well-known expectation maximization technique (EM) [36], which combines both the similarity in the Eigenspace and graph structure.

Xiao *et al.* use the normalized Laplacian to embed the vertices of a graph into its Eigenspace [130]. They build their solution based on the concept of the *heat kernel* that is computed by exponentiating the spectrum of the normalized Laplacian matrix. Let U and Λ be the matrix of Eigenvectors and the diagonal matrix of the Eigenvalues for the matrix A, respectively, then the heat kernel h_t is defined as:

$$h_t = U \exp[-t\Lambda]U^T \tag{2.13}$$

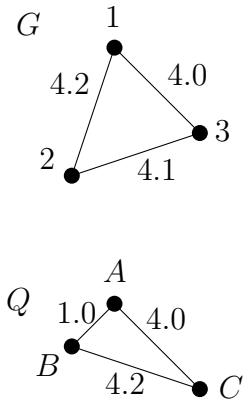
The heat-kernel describes the flow of information across the edges of a graph over time t. The main idea behind it is that the differences between the structure of two graphs mainly affect their smaller Eigenvalues. This means that the Eigenvectors that correspond to the smaller Eigenvalues are unstable. To reduce their effect on graph matching, the Eigenvalues are exponentiated as seen in the previous equation. Also, Eigenvectors corresponding to the smaller Eigenvalues are truncated to cope with the differences in graph size. To create a vector-based representation for the vertices, they use the Young-Householder decomposition of the heat kernel, i.e., $h_t = Y^T Y$. As a result, the coordinate matrix of the vertices is Y such that:

$$Y = \exp[-\frac{1}{2}\Lambda t]U^T \tag{2.14}$$

To find the distance between two vertices based on their heat kernel embedding, they propose using the squared Euclidean distance. This creates a distance matrix between the vertices of two graphs, where the best match between the vertices is selected using Scott and Longuet-Higgins algorithm [110]. Such an algorithm is considered as a greedy approach to solve the assignment problem. It matches two vertices if their similarity is the maximum among the other pairs of vertices.

• Spectra of Affinity Matrix

We discussed in Section 2.3 that the product graph is utilized in approaching the maximum common subgraph problem. The main idea was to find the maximum clique in the product graph, which represents the maximum common subgraph. Following the same methodology, the spectra of the *affinity matrix* of two graphs are used to estimate the match between them. The difference between the affinity matrix and the product graph is that the similarity between the weights of two edges is utilized by the affinity matrix, however, the product graph requires that the two edges have identical labels.



	1,A	1,B	1,C	2,A	2,B	2,C	3,A	3,B	3,C
1,A					0.24	0.95		0.25	1.0
1,B				0.24		1.0	0.25		1.0
1,C				0.95	1.0		1.0	0.95	
2,A		0.24	0.95					0.24	0.97
2,B	0.24		1.0				0.24		0.24
2,C	0.95	1.0					0.97	0.24	
3,A		0.25	1.0		0.24	0.97			
3,B	0.25		0.95	0.24		0.24			
3,C	1.0	1.0		0.97	0.24				

Figure 2.8: The affinity matrix between the two graphs G and Q. Empty entries represent a similarity of value zero and are removed for convenience.

Given the two graph $G = (V, E, w_g)$ and $Q = (U, T, w_q)$, such that w_g and w_q are the weighting functions assigning a real-valued label to the edges in E and T, respectively. Also suppose that $m = |V|$ and $n = |U|$. Then, the affinity matrix of G and Q is a square matrix $K \in \mathbb{R}^{mn \times mn}$, such that an entry k_{ab}, $a = (v_i, u_k)$ and $b = (v_j, u_l)$, represents the similarity of the two edges $e_{ij} = (v_i, v_j) \in E$ and $e_{kl} = (u_k, u_l) \in T$. k_{ab} is defined as :

$$k_{ab} = \begin{cases} s(e_{ij} = (v_i, v_j), e_{kl} = (u_k, u_l)), & \text{if } e_{ij} \in E \wedge e_{kl} \in T \\ 0, & \text{otherwise} \end{cases} \tag{2.15}$$

where $s(.)$ is an edge similarity function. Figure 2.8 shows the affinity matrix between two graphs such that the function $s(.)$ is a Jaccard function based on the weights of two edges.

Once the affinity matrix is constructed, matrix decomposition is used to estimate the cluster of entries in the affinity matrix that represents an approximate solution to the inexact graph matching problem. Formally, the match between the two graphs is estimated by a cluster of assignments $C = \{(v_i, u_k) | v_i \in E, u_k \in T\}$ in their affinity matrix that has the maximum inter-cluster similarity score:

$$C^* = \arg\max_C \sum_{a,b \in C} k_{ab} \tag{2.16}$$

This optimization problem can be written by utilizing a vector of assignment indicators $x \in \{0, 1\}^{mn}$, such that an entry indicates where $v_i \in E$ is mapped to

vertex $u_k \in T$. Using such a vector, the optimal solution x^* is computed as:

$$x^* = \arg\max(x^T K x) \tag{2.17}$$

under the constraints:

$$1.\ \forall i \in \{1, \ldots, |G|\}, \sum_{k=1}^{|Q|} x_{ik} \leq 1$$

$$2.\ \forall k \in \{1, \ldots, |Q|\}, \sum_{i}^{|G|} x_{ik} \leq 1$$

Conditions 1 and 2 guarantee a one-to-one mapping between the vertices of two graphs. The above optimization problem is considered as a quadratic assigned problem, which is known to be NP-complete [48]. To give an approximate solution, Leordeanu and Hebert [78] propose to relax the previous two constraints and utilize a solution vector from the continuous domain $x \in [0, 1]^{mn}$. Since K is positive and symmetric and $||x^*|| = 1$, then based on Raleigh's ratio theorem (Theorem 2.1), x^* that maximizes the inter-cluster score $x^T K x$ is the principal Eigenvector of K. Notice that $x^* \in [0, 1]^{mn}$ does not represent an assignment vector. To binarize x^*, they follow a greedy algorithm by taking the assignment that has the maximum value in the principal Eigenvector, and add it to the match. Then all the candidate matches in K that contradict the vertices in the match are removed. This is because the higher the value in the principal Eigenvector the stronger the association with the cluster.

Another approach to select the cluster from the affinity matrix is to adopt a *random walk ranking* algorithm [27]. The higher the rank the more reliable the match between two vertices. Notice that the constraints in Equation 2.17 are adopted as a post-processing step during the binarization step, which leads to a weak local minimum. Such constraints are considered by a *reweighed random walk* algorithm proposed by the same authors in [27]. The key idea is that a random walk is *personalized* by the constraints following the idea of *teleporting* of the well-known PageRank algorithm [76].

Unfortunately, the previous approaches require an explicit computation of the affinity matrix, which scales poorly as graph size increases. Suppose that the size of a graph is n, then the memory complexity of storing the affinity matrix of two graphs is $O(n^4)$. To solve this problem, Zhou and la Torre [140] utilize the fact that the affinity matrix is sparse and propose to factorize it into smaller and denser matrices. This includes the incident matrices, which represent the relationships between the vertices and edges of the graphs, the vertex-to-vertex similarity matrix, and the edge-to-edge similarity matrix. After factorization, the affinity matrix is

represented by four smaller and denser matrices. In addition to the sparsity of the affinity matrix, Kang *et al.* [66] use the redundancy in the affinity matrix for matrix factorization. They consider two entries that have a difference within a certain threshold as redundant, which is a consequence of having several pairs of vertices that are connected by edges that have nearly the same weight. They use a binning method to assign all edges within a certain difference to the same value. Then, a new approximated matrix is represented by the linear combination of the Kronecker product of several smaller compressed matrices. Based on such smaller vertices, the principal Eigenvector is computed without the need to materialize the affinity matrix.

- **Spectra of Similarity Matrix**

As we discussed, the Hungarian algorithm is used to find the match between two sets of vertices based on their similarity matrix. Similarly, the spectra of the vertex-to-vertex similarity matrix can be used to approximate the match between two graphs [110, 112]. Such an approach was initially proposed to match two sets of features and later used for graph matching [130]. Given two graphs G and Q such that $m = |G|$ and $n = |Q|$ and $m \neq n$, a similarity matrix S is built based on the features extracted from the neighborhoods of the vertices. Then, *singular value decomposition* (SVD) is used to factorize S as follows:

$$S = U\Sigma V^T \tag{2.18}$$

where U and V are orthogonal matrices of sizes m and n, respectively and Σ is a diagonal matrix of the singular values of the similarity S. The rows of U are considered as the coordinates of the vertices of G in the Eigenspace. The columns of V^T are considered the coordinates of the vertices of Q. After that, a new matrix P is computed by replacing Σ by a diagonal matrix D such that all elements at the diagonal are assigned the value 1.

$$P = UDV^T \tag{2.19}$$

An entry p_{ij} represents the similarity between $v_i \in G$ and $u_j \in Q$. To select the best match from P, they follow an *extremum principle*, which means that v_i is matched to v_j if p_{ij} is the maximum value in both row i and column j.

2.4.3 Continuous Optimization Approaches

In this section, we discuss another approach for tackling the inexact graph matching problem. It gives an approximate solution by first relaxing the graph matching problem, which is a discrete optimization problem, to another continuous non-linear optimization one. Several algorithms can be then used to approximate the latter problem. Many of them follow an iterative continuous improvement approach. During each iteration, the probability of mapping any two vertices is updated by considering the structure of the graph and the mapping probabilities that are computed at the previous iteration. Notice that the solution of the graph matching problem is expressed as a permutation matrix, i.e., the entries are either 1 or 0. However, the solution of the relaxed continuous optimization problem is a matrix such that each entry expresses the quality of matching any two vertices, i.e., the entries belong to the interval $[0, 1]$. To convert the mapping probabilities to a permutation matrix, a binarization technique is used to convert back the solution to the discrete domain.

Following this, *relaxation labeling* is an approach to approximate the graph matching problem. Each vertex from a graph is assigned a label indicating its corresponding vertex from another graph. For this, each vertex has a vector of probabilities indicating the strength of mapping that vertex to every vertex of the other graph. At the beginning, such probability is measured mainly based on a vertex-to-vertex label similarity. Then, the probabilities are updated iteratively taking the information of the neighboring vertices into account [45]. A maximum a posteriori estimation (MAP) approach for relaxation labeling is proposed by Hancock and Kittler [51]. Given two graphs $G = (V, E, l_g)$ and $Q = (U, T, l_q)$, suppose a match between the two graphs is represented by a function $f : V \rightarrow \{U \cup \varepsilon\}$, then the maximum a posteriori estimate (MAP) $P(f|G, Q)$ is written as follows:

$$\arg\max_f P(f|G, Q) = \arg\max_f P(G, Q|f) \times P(f) \tag{2.20}$$

where $P(G, Q|f)$ the conditional density function of the similarity between the labels of the mapped vertices given the matching function f, $P(f)$ is the prior probability of the match f. Wilson and Hancock propose gradient ascent to optimize the above objective function [127]. The crucial part of their model is the definition of the prior $P(f)$. They propose to compute the prior based on the average similarity between the neighborhoods of the vertices of one graph to the neighborhoods of the vertices of another one. They define the neighborhood of a vertex as that vertex and its direct neighboring vertices and called it a *super-clique* [86].

Luo and Hancock propose a maximum likelihood estimation (MLE) formalism for graph matching [80]. They assume the vertices of graph G as observed data and the corresponding vertices from another Q as the hidden data. A mixture model is then defined based on the vertices of graph Q. This means that any vertex $v_i \in V$ may be generated from a vertex $u_j \in U$. As a result, M can be seen as a mixture model parametrized by the set of assignment variables m_{ij}. Following these assumptions, the solution to the graph matching problem is the mixture model M^* that maximizes $P(G|M)$, which is the incomplete-data likelihood of the observed graph. This is formalized as:

$$M^* = \arg\max_{M \in \mathcal{M}} P(G|M) \tag{2.21}$$

where $\mathcal{M}$ is the space of all possible matches between the two graphs. In general, there are $2^{|G||Q|}$ different matches and thus mixture models. To maximize the objective function, the authors propose the expectation maximization technique (EM).

A well-known algorithm that follows the continuous optimization scheme is the graduated assignment graph matching of Gold and Rangarajan [50]. Given an initial match between two graphs, the authors use Taylor series expansion to approximate the optimal solution to the graph matching problem. They prove that minimizing the solution of the Taylor series is equivalent to solving the assignment problem. Their iterative approach starts by an initial match, then, the derivative of the Taylor series with respect to such a match is computed, which represents the sum of the similarities between the neighboring vertices of one vertex to the neighboring vertices of another one. In other words, such a derivation can be seen as a vertex-to-vertex similarity matrix. The match computed at an iteration is then used to vote for the similarity of the vertices of two graphs. Instead of solving the optimal solution to the assignment problem, they propose to compute a *soft assignment*. It represents the probability of matching any two vertices of the two given graphs. This can be achieved by, first, exponentiation the entries of the vertex-to-vertex similarity matrix by utilizing a control parameter β, then, performing a two way normalization to get a double stochastic matrix. Through consecutive iterations, the algorithm increases the control parameter β such that when it converges the vertex-to-vertex similarity matrix, which is a double stochastic matrix, becomes mostly a permutation matrix. To produce a permutation matrix, two vertices from the two graphs are matched if their similarity is the maximum among other possible vertex similarities.

2.5 Summary and Discussion

In this chapter we have discussed the graph matching problem. Since the complexity of computing the optimal solution to the inexact graph matching problem is NP-hard, several approaches are presented in the literature to give approximate solution. Even though each approach follows a different heuristic, in the following, we summarize the general framework that nearly all such approaches follow.

1. **Vertex similarity.** The basic task for any graph matching algorithm is to estimate the similarity of two vertices from two different graphs. This is accomplished, in most of the related work, by utilizing the labels of the vertices. In addition to this, the labeling information of the direct neighboring vertices is utilized.

2. **Graph matching using the assignment problem.** To alleviate the complexity of the inexact graph matching problem, the overall graph structure is neglected and only the concept of vertex-to-vertex similarity is utilized. A distance matrix between the vertices of two graphs is computed. Then, approaches to the assignment problem utilize such a distance matrix to approximate the match between the two graphs.

3. **Iterative graph matching.** Utilizing only the concept of vertex-to-vertex similarity generates a match that is structurally incompatible. Two direct neighboring vertices from a graph may be matched to non-direct neighboring vertices of another one. To overcome this problem, several approaches follow an iterative scheme to improve the match. An initial match is computed utilizing solutions to the assignment problem. Then, iteratively, the vertices of the match are used to update the vertex-to-vertex similarity by considering more structural information. In general, given two vertices v and u, if the direct neighboring vertices of v are matched to the direct neighboring vertices of u, then the similarity of v and u increases.

Chapter 3

Vertex Similarity

The previous chapter discussed several approaches and algorithms to tackle the graph matching problem. Even though each approach has its own techniques, most of the approaches utilize the concept of *vertex similarity* as a metric to estimate the match between two graphs. This makes such a concept the basis of any graph matching algorithm.

In addition to graph matching, the concept of vertex similarity has been used by a variety of graph-based algorithms such as link prediction [22], web search [13], and frequent subgraph discovery [91]. A naive solution to estimate the similarity between two vertices is based on the similarity of their labels, which is denoted as *unary* notion of similarity. For example, two atoms in a chemical compound are similar if they have the same type, or two users in a social network have similar interests if they have the same age. To implement such a notion of similarity, one can use the *Minkowski distance* or a *Dirac* function that assigns a distance of 0 if the two vertices have the same label and 1 otherwise. Any unary notion of similarity, normally, does not consider the structure of the graph. However, the similarity between two graphs is highly related to their structure. Therefore, a more accurate *structure-based* similarity approach is normally used. It is built on the concept that two vertices are similar if they are connected to similar vertices [77].

We distinguish two classes of applications that utilize the structure-based similarity approach: applications that use the similarity between two vertices in one big graph and applications utilizing the similarity between two vertices that belong to two different graphs. The first class of applications analyzes one big graph like a social network, a biological network, or a co-author network. The purpose of vertex similarity for such applications is to predict or recommend friendships, discover common behaviors, or find communities. For such a class of applications, two vertices

are considered similar if they are connected to the same vertices, which can be implemented as a *Jaccard* or a *cosine* similarity functions. However, in many cases, two vertices maybe considered similar even if they do not share common vertices. Several algorithms implement this notion of similarity, including HITS [70], SimRank [59], and P-Rank [136].

The second class of applications utilize vertex similarity for the purpose of graph matching and frequent subgraph discovery. In Chapter 2, we discussed different techniques to estimate the similarity of two graphs based on the similarity of their vertices, which is estimated by using *local features* that are extracted from the neighborhood of each vertex. The neighborhood of a vertex can be defined in different ways such as 1) the vertex and its incident edges [139], 2) in addition to the previous, the neighboring vertices are also included, 3) a spanning tree with a certain depth [123], and 4) simple paths with a certain length [115, 131]. Once the neighborhood is defined, the labels assigned to the vertices and edges are used as the features of that vertex. In addition to local-based features, *global-based* features are also used based on spectral graph theory. The projection of each vertex into the Eigenspace of the adjacency or the Laplacian matrices are considered as a global feature of that vertex. The reader may refer to Section 2.4.2 for more details.

In this chapter, we detail the problem of vertex similarity for geometric graphs. In Section 3.1, we survey related work. Section 3.2 formalizes a local similarity concept for vertices of geometric graphs. We prove that for general geometric graphs, the complexity of computing the similarity between two vertices is NP-hard. In Section 3.3, we concentrate on vertex similarity for geometric graphs in 2D space and propose an approach that runs in cubic time with respect to the vertex degree. Experimental results for different data sets are presented in Section 3.4. Finally, Section 3.5 summarizes the chapter.

3.1 Related Work

Several graph matching algorithms use the Euclidean distance to estimate the similarity between real-valued labels that are assigned to the vertices and edges [101, 134]. For geometric graphs, the coordinates of the vertices cannot be simply treated as real-valued attributes since they are measured with respect to the particular reference axis frame for each graph. This makes the Euclidean distance incapable of estimating the spatial distance between vertices of two graphs underlying, e.g., a geometric transformation. In addition to this, pure structural graph matching

approaches, such as the spectral approach, cannot be applied to geometric graphs because they do not consider the spatial property of a graph, as shown in Figure 1.1. In addition to the Euclidean distance and graph spectra, several other approaches have been proposed to solve the vertex similarity problem for non-geometric graphs, as discussed in Chapter 2. However, for geometric graphs, little can be found in the literature. We believe that this is a consequence of the complexity of the problem in the case of geometric graphs, which will be discussed in later sections. In the following, we discuss current solutions to estimate the similarity between vertices of geometric graphs. We divide them into two groups, the global-based approaches and the local-based ones.

• **Global Features.** The global-based feature approaches extract a feature for each vertex using the overall graph structure. In the following, we discuss two approaches: the spectral and the landmark distance.

In Section 2.4.2, we discussed how graph spectra are used to match two unlabeled graphs. The main idea is to extract a feature for each vertex based on the values of the Eigenvectors. Such features are then used by the Hungarian algorithm for graph matching. To use the same concept for geometric graphs, the spectra of the *weighted* adjacency or the weighted Laplacian matrices are used [118]. The weight of an entry represents the length of an edge, which is computed using the Euclidean distance between the coordinates of its incident vertices. Then, eigendecomposition is used to generate a spectral feature for each vertex, which is represented by the values of the Eigenvectors with respect to that vertex. Since graphs with different number of vertices create a different number of Eigenvectors, the spectral features for the vertices are truncated by keeping the values with respect to the most dominant Eigenvectors [141], i.e., the Eigenvectors that correspond to the largest Eigenvalues. Based on this, the distance between two vertices equals the Euclidean distance between their spectral features. A major drawback for the spectral approach is that it cannot handle labeling information. Also, such an approach is sensitive to differences in the number of vertices, the structure of the graph, and the lengths of the edges.

Another global-based vertex similarity approach is based on the *landmark distance* concept [26]. First, a set of vertices from each graph is selected as landmarks. Then, every vertex from the graph is represented by a feature vector containing the distances to the landmarks. The distance is measured as the length of the shortest path between the vertex and a landmark. Then, the distance between two vertices is computed using the Manhattan distance between their landmark-based features.

The basis of such an approach is the selection of landmarks for each graph. Cheong *et al.* [26] propose to use four landmarks as the extreme vertices in the boundaries of the graph, i.e., peripheral vertices. However, such an approach is incapable of matching graphs that differ in the number of vertices.

• **Local Features.** The main drawback of the global-based approaches is their sensitivity to changes in the graph structure and the number of vertices. To overcome such a problem, local-based features are used based on the neighborhood of each vertex, which will be discussed in the following.

One of the earliest approaches to estimate the similarity of different vertices is the *histogram-based* approach [46, 57, 69, 116]. A histogram is created from the spatial properties of the neighborhood of each vertex, which is defined by the vertex and its direct neighbors. The histogram stores the pair-wise relationships between the edges that are incident to that vertex, which consists of the ratio of the lengths of the edges in addition to the angle between them. As a result, the local feature is a 2D histogram of edge lengths and angle values. These two properties preserve the spatial property of a graph, i.e., the edge length indicates the spatial distance between a vertex and a direct neighbor and the angle between two edges is used as an estimate of the distance between their incident vertices. As a result, the distance between two vertices is estimated by the distance between their geometric histograms, which is computed by the χ^2 or the Bhattacharyya distances [21]. Unfortunately, histogram approaches face problems in binning and normalization, especially when dealing with real-valued attributes such as the length of an edge or the angle between two edges.

Notice that the above approaches extract features that are invariant to geometric transformation. Another approach to solve vertex similarity is to use *geometric hashing* based on the coordinates of the vertices. The basis of this approach is to create several local frames for the neighborhood of each vertex, which is defined again by that vertex and its direct neighbors. Then, the coordinates of the vertices in the neighborhood of a vertex are measured with respect to each local frame. To estimate the similarity between two vertices given their local frames, the Euclidean distance is used based on the coordinates of the vertices. Since there are several local frames for the neighborhood of a vertex, all of them must be tested to guarantee an optimal distance. To overcome such a complexity, hashing is used to speed up the search for the local frame that best estimates the distance between two vertices. The geometric hashing approach is efficient in the case of matching vertices that have a homogeneous transformation, i.e., affine transformation. But, in the case of inexact matching, such

an approach fails to estimate the similarity of the vertices. This is because the hashing function is not tolerant to differences in the local frames of two vertices.

3.2 Local-based Vertex Similarity

As we have discussed, finding the similarity between vertices either utilizes global properties of the graphs or local properties of the neighborhoods of the vertices. In our framework, we follow the local-based approach, which has been proved to give good results for general non-geometric graphs [63, 69, 101, 104, 134]. This can be justified since, in general, two non-identical graphs that model two similar objects have many differences in their overall structure and spatial properties. However, in many cases, local properties of the neighborhoods of the vertices are highly similar. For example, given images of a moving object over time, the background of the object may have several differences over time. However, the spatial property and structure of the object remain the same across different snapshots.

The local-based vertex similarity approach is based on the concept that two vertices are similar when their neighbors are similar, i.e., the more similar neighbors the two vertices have, the more similar they are. Different algorithms define the meaning of *locality* in different ways. Examples are paths that start from a certain vertex and have a certain length [115, 131], trees rooted at each vertex [123], and a subgraph that has all vertices reachable with a certain number of hubs from that vertex [63, 69, 101, 104, 134]. Based on the research of non-geometric graphs, subgraph features have the highest selectivity power. However, finding the optimal similarity between two subgraphs is as complex as the graph matching problem itself. To overcome this bottleneck we use minimal subgraphs for our framework, each consisting of a vertex and its direct neighbors. We call such a subgraph *vertex signature*. Notice that the vertex signature concept has been used under different names such as *clique* [114], *local structure* [101], *starts* [134], and *subgraph* [95].

Definition 3.1. (Vertex Signature) *Given a vertex v_i in a graph $G = (V, E)$, the vertex signature $S(v_i)$ is a subgraph $G' = (V', E')$ of G such that $V' = \{v_i \cup \{v_j | (v_i, v_j) \in E\}\}$. For each vertex $v_j \in V'$, $v_j \neq v_i$, there exists an edge $(v_i, v_j) \in E'$. v_i is called the* ***root vertex*** *of $S(v_i)$.*

Figure 3.1 shows the vertex signatures for different vertices. For graph G with n vertices, a multi-set of n vertex signatures represents the local features for all vertices.

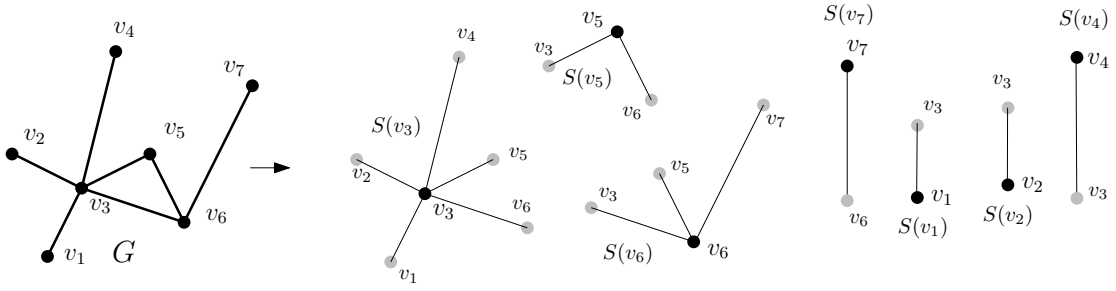


Figure 3.1: Several vertex signatures are defined for the graph G. The root vertex is shown in black for each vertex signature.

For simplicity, the figure shows an unlabeled graph (G). However, the concept naively scales to include labeling information for the vertices and edges.

After defining the meaning of locality, the similarity between two vertices is estimated by computing the similarity of their vertex signatures. A function that quantifies the similarity between two vertex signatures must satisfy geometric transformations, i.e., two vertex signatures are considered spatially identical if there is a geometric transformation that makes the coordinates of one vertex signature identical to the coordinates of the other [75]. For two vertices, v and u, an *exact vertex similarity* between their vertex signatures is an instance of the geometric graph isomorphism problem. Given that $n = deg(v) = deg(u)$, the exact vertex similarity can be computed in $O(n \log n)$ for geometric graphs in 2D space or $O(n^{d-2} \log n)$ for geometric graphs in d-dimensional space [7].

However, exact vertex similarity obviously fails to measure the similarity between vertices under noise and outliers, that is, when there are differences in the number of vertices, labeling information, and spatial properties of two vertex signatures. For example, in scientific applications, two similar objects are often represented by two non-identical graphs. In pattern recognition applications, acquisition methods often introduce noise in the number of vertices and their locations. Also, the structure and connectivity of vertices often vary between graphs representing similar objects. As a result, two vertex signatures representing similar vertices have differences in the number of neighbors, labeling information, the distance between the root vertex and its neighbors, and the distance between the neighbors themselves. This leads to the concept of *inexact vertex similarity*, which will be detailed in the following section.

In Figure 3.2, we show a couple of examples for the concepts addressed above. If no spatial properties are considered, vertex signature $S(v_1)$ is considered identical to $S(v_3)$. However, spatial properties make $S(v_1)$ more similar to $S(v_2)$ than $S(v_3)$.

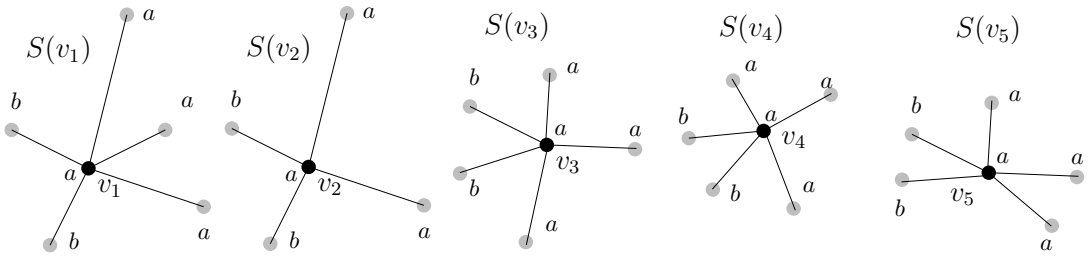


Figure 3.2: Spatial properties make vertex v_1 more similar to v_2 than to v_3. An affine transformation makes $S(v_3)$ identical to $S(v_4)$. Only inexact vertex similarity can estimate the distance between $S(v_3)$ and $S(v_5)$.

A rigid transformation that consists of a rotation and a scaling makes $S(v_3)$ identical to $S(v_4)$. However, there is no homogeneous transformation that makes $S(v_3)$ identical to $S(v_5)$. For the latter case, only an inexact match can be used to make the two vertex signatures identical.

3.2.1 Vertex Edit Distance

To compute the inexact similarity between two vertices based on their vertex signatures, we adopt the *edit distance* concept that is been used in matching strings and graphs [108, 120]. It is defined as the minimum amount of distortion that is needed to make a string or a graph identical to another. We call the edit distance of two vertex signatures the *vertex edit distance (VED)*. For two vertex signatures $S(v)$ and $S(u)$, the key idea of the VED is to delete some vertices and edges from $S(v)$, re-label some other vertices and edges, change the coordinates of some vertices, and insert some vertices and edges into $S(u)$ such that the two vertex signatures become identical. For this, we adopt three *edit operations*: substitution (re-label), insertion, and deletion. A sequence of edit operations that transfer one vertex signature identical to another is called an *edit path*. Obviously, there are many possible edit paths from one vertex signature to another. As a result, the VED is defined as the distance with the minimum cost of all of them:

Definition 3.2. (Vertex Edit Distance) *Let $\phi(S(v), S(u))$ be the set of all geometric transformations between the coordinates of $S(v)$ and $S(u)$, $\Upsilon_{\phi_i}(S(v), S(u))$ be the set of all edit paths between the two vertex signatures $S(v)$ and $S(v)$ after applying the geometric transformation ϕ_i, then the vertex edit distance is defined as:*

$$d(v,u) = \min_{\phi_i \in \phi(S(v),S(u)), p_j \in \Upsilon_{\phi_i}(S(v),S(u))} cost(p_j) \tag{3.1}$$

where cost(p_j) is the total cost of all edit operations that consist the path p_j.

The cost of an edit path depends on the cost of its edit operations, which we define as the following. The cost of a substitution between two vertices is defined by the Euclidean distance between their coordinates, the distance between their labels, and the substitution costs of their edges. The substitution cost between two edges is defined as the distance between their labels in addition to the distance between their lengths. The cost of vertex insertion or deletion equals to a constant α.

The problem of vertex edit distance between two vertex signatures $S(v)$ and $S(u)$ consists of two optimization problems. The first problem is a linear assignment problem to find a correspondence between the neighbors of v and the neighbors of u. The second problem is a least-square continuous problem to transform the coordinates of the neighboring vertices. Each problem is solved in polynomial time complexity if treated separately, but it is very difficult to find a solution when the two problems are combined [29]. For example, if we get the optimal mapping between the vertices of $S(v)$ and $S(u)$, one can easily find the distance between them as follows. First, a local coordinate frame is created for $S(v)$ and another one for $S(u)$. To create a local frame, for example, in 2D space, two vertices v and any direct neighboring vertex v_i are selected from $S(v)$ to create the basis of its local frame. Their correspondent vertices in $S(u)$, u and u_j, are selected to define the basis of its local frame. Then, the coordinates of the vertices are measured with respect to such local frames. Finally, the Euclidean distance is used to find the distance between the correspondent vertices.

Lemma 3.1. *The problem of vertex edit distance for geometric graphs in the $\mathbb{R}^d$ space is NP-hard for $d \geq 2$.*

In the following, and without loss of generality, we give a proof for the above lemma in the case of unlabeled geometric graphs.

Proof. For two unlabeled geometric graphs, we reduce the problem of inexact point set matching to the problem of vertex edit distance. Let $P = \{p_1, p_2, \ldots, p_n\}$ and $Q = \{q_1, q_2, \ldots, q_m\}$ be two point sets in $\mathbb{R}^d$, $d \geq 2$. The two point sets are reduced to two vertex signatures in polynomial time as follows. All points from the point set P become vertices directly connected to a dummy root vertex v_p. The coordinate of v_p is computed as the center of the point set P. In the same way, the points from Q create a vertex signature with a dummy root vertex u_q. The optimal match between P and Q is the optimal mapping of the neighbors of v_p and the neighbors u_q. Such an

optimal match represents the optimal edit path between the two vertex signatures. Non-matched points (vertices) represent the insertion and deletion operations. A substitution operation is indicated by a correspondence from one point to another.

The problem of inexact point set matching in the $\mathbb{R}^d$ space is proved to be NP-hard [6] where $d \geq 2$. As result, the problem of computing the optimal solution of vertex edit distance for geometric graphs in $\mathbb{R}^d$ space is also NP-hard.

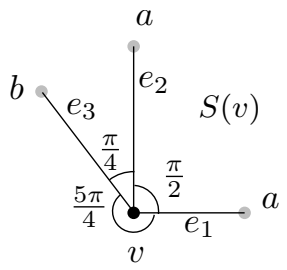
□

3.3 Vertex Similarity for 2D Geometric Graphs

In the previous section, we proved that computing the vertex edit distance is a very hard problem. The discussion was mainly about the computation of the spatial distance based on the coordinates of the vertices. We now concentrate on computing the vertex edit distance for geometric graphs in 2D space, which is a special case where we propose an algorithm that runs in polynomial time. The main idea of our solution is to utilize the property that the edges in a vertex signature have a natural cyclic order, which is a consequence of the embedding of the direct neighboring vertices in 2D space. Such a total order is then used to extract a spatial feature from each vertex signature that is invariant to geometric transformations. Based on the spatial features of two vertices, the edit distance between their vertex signatures is computed by solutions to the string edit distance problem [120]. In the following, we first discuss the optimal solution to the VED problem for vertices in 2D space. Then, in Section 3.3.2, we propose a suboptimal solution that can be computed in cubic time with respect to the vertex degree.

3.3.1 Optimal Solution

Inspired by the geometric histogram approach, we propose to compute the distance between two vertex signatures based on spatial features that are invariant to geometric transformations. This means that the coordinates of the vertices are not used to estimate their spatial distance. Such spatial features consist of the lengths of the edges in addition to the angles between them. The intuition behind this is that the lengths of the edges define the distances between a vertex and its direct neighbors. On the other hand, the angle between two adjacent edges is used as an estimate of the distance between their incident vertices.



$$\mathcal{F}_v = \{F_1, F_2, F_3\}$$
$$F_1 = [(|e_1|, 0, a), (|e_2|, \tfrac{\pi}{2}, a), (|e_3|, \tfrac{3\pi}{4}, b)]$$
$$F_2 = [(|e_2|, 0, a), (|e_3|, \tfrac{\pi}{4}, b), (|e_1|, \tfrac{3\pi}{2}, a)]$$
$$F_3 = [(|e_3|, 0, b), (|e_1|, \tfrac{5\pi}{4}, a), (|e_2|, \tfrac{7\pi}{4}, a)]$$

Figure 3.3: The spatial feature for the vertex signature $S(v)$, where a and b are the labels of the neighboring vertices. For simplicity, edges are unlabeled.

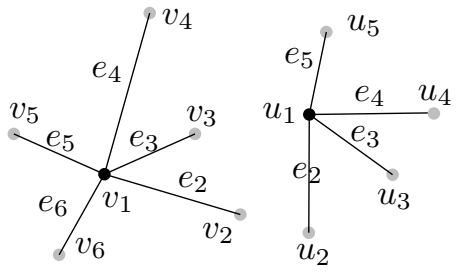
Definition 3.3. (Spatial Feature) *Given a geometric graph $G = (V, E, l, c)$, the spatial feature $\mathcal{F}_v$ for the vertex signature $S(v)$ is a* ***set of strings*** *$\mathcal{F}_v = \{F_1, F_2, \ldots, F_n\}$, $n = \deg(v)$, such that a string $F_i = [f_{i,1}, f_{i,2}, \ldots, f_{i,n}]$ represents the pair-wise geometric relationships between a* ***reference edge*** *e_i and every other edge $e_i \in S(v)$, where a token $f_{i,j} \in F_i$ is defined as:*

$$f_{i,j} := (|e_j|, \angle e_i e_j, l(e_j), l(v_j))$$

where $|e_j|$ denotes the length of the edge e_j, $\angle e_i e_j$ denotes the counter-clockwise order angle between the edges e_i and e_j, $l(e_j)$ is the label of the edge e_j, and $l(v_j)$ is the label of the neighboring vertex that is incident to the edge e_j.

For geometric graphs in 2D space, the total ordering property is used to represent a vertex signature as a set of strings. A vertex signature $S(v)$ that has n edges has a set of n different strings $\mathcal{F}_v = \{F_1, F_2, \ldots, F_n\}$. Each string F_i represents the pair-wise geometric relationships between an edge e_i and every other edge $e_j \in S(v)$. In this case, we call e_i the reference edge. As a result, a token $f_{i,j}$ of a string F_i represents the pair-wise geometric relationship between the two edges e_i and e_j. The ordering of the tokens in a sting follows the cyclic order of the edges at that vertex signature. For the string F_i with the reference edge e_i, the first token represents the relationship between e_i and itself, the second token represents the pair-wise relationship between e_i and the next edge in a counter clock-wise order, and so on. In Figure 3.3, we show the spatial feature for vertex signature $S(v)$, which consists of three different strings. The angle in the first token of any string equals to zero since it represents the pair-wise geometric relationship between an edge and itself.

Following this feature extraction method, one can see that each string represents a *polar coordinate system* for that vertex signature. The root vertex of a vertex signature is considered as the *pole*, the reference edge of a string represents the *polar*



The conditional vertex distance between v_1 and u_1, given v_3 and u_5, is defined as the string edit distance between the two strings

$F_3 = [f_{3,3}, f_{3,4}, f_{3,5}, f_{3,6}, f_{3,2}]$ and
$F_5 = [f_{5,5}, f_{5,2}, f_{5,3}, f_{5,4}]$

Figure 3.4: The conditional vertex distance between vertex v_1 and u_1 .

axis, a token in the string represents the polar coordinate of a neighboring vertex such that the counter clock-wise angle represents the angle with respect to the polar axis, and the edge length represents the distance between that point, i.e., the neighboring vertex, and the pole, i.e., the root vertex.

After representing each vertex signature as a spatial feature, the solution to the vertex edit distance between two vertex signatures $S(v)$ and $S(u)$ is defined by the minimum distance between the strings of $\mathcal{F}_v$ and the strings of $\mathcal{F}_u$. To formalize this, we first define the concept of *conditional vertex distance*.

Definition 3.4. (Conditional Vertex Distance) *Given two vertices v_i and u_j with their spatial features $\mathcal{F}_{v_i}$ and $\mathcal{F}_{u_j}$, respectively. Suppose that $v_k \in S(v_i)$, $v_k \neq v_i$, and $u_l \in S(u_j)$, $u_l \neq u_j$. The conditional vertex distance between v_i and u_j, given v_k and u_l, denoted as $d(v_i, u_j|v_k, u_l)$, is defined as the string edit distance between the two strings F_k and F_l such that the edge (v_i, v_k) is the reference edge of F_k and the edge (u_j, u_l) is the reference edge of F_l.*

Figure 3.4 shows an example of the conditional vertex distance between two vertex signatures. The conditional part of the distance specifies the reference edge for the strings to be compared. In other words, it defines the polar axis for each of the polar systems of the two vertex signatures to be compared. To solve the conditional vertex distance, one can use solutions to the string edit distance. For this, three edit operations are defined between the tokens of different strings. The substitution cost is defined as the polar distance between two tokens in addition to the distance in the labeling information of the edges and the neighboring vertices. The insertion and deletion costs are computed proportional to the distance between a neighboring vertex and the vertex defining the polar axis. In addition to this, a constant α defines the cost of inserting or deleting the labels of the vertices and edges.

Using the conditional vertex distance between the strings from one vertex signature to the strings of another one, the vertex edit distance is formalized as follows:

Definition 3.5. (Vertex distance) *The distance between two vertices v_i and u_j is defined as:*

$$d(v_i, u_j) := \min_{\substack{v_k \in S(v_i) \\ u_l \in S(u_j)}} d(v_i, u_j | v_k, u_l) \tag{3.2}$$

We call the previous solution to the VED problem naive since every string of a spatial feature is compared to all strings of another one. In general, the solution to the VED problem between two vertex signatures $S(v)$ and $S(u)$, $m = deg(v)$ and $n = deg(u)$, consists of $m \times n$ string comparisons. Each comparison uses the string edit distance algorithm with a runtime complexity of $O(mn)$ [120]. As a result, the runtime complexity of computing the distance between the two vertex signatures based on this approach is $O(m^2 n^2)$.

3.3.2 Approximate Solution

To enhance the running time of the naive approach, we propose an approximate solution to the VED problem that can be computed in $O(mn \log n)$, such that n and m are the number of edges for two vertex signatures. Notice that the high complexity of the naive approach is a consequence of extracting n different strings from a vertex signature. On the other hand, the approximate solution can be computed in less than cubic complexity by representing all the strings extracted from a vertex signature as one cyclic string. We require that edges in a vertex signature are sorted in counter-clockwise order around the root vertex, and that an angle is measured relatively to the previous edge in the sequence and not with respect to the reference edge. As a result, the invariant feature that is extracted from each vertex signature is a cyclic string that is defined as:

Definition 3.6. (Spatial Feature) *Given a geometric graph $G = (V, E, l, c)$, the spatial feature F_v for the vertex signature $S(v)$ is a* ***cyclic string*** *$F_v = [f_1, f_2, \ldots, f_n]$, $n = deg(v)$, such that each token f_i is defined as:*

$$f_i := (|e_i|, \angle e_i e_{i-1}, l(e_i), l(v_i))$$

where $|e_i|$ denotes the length of the edge e_i, $\angle e_i e_{i-1}$ denotes the angle between the edges e_i and e_{i-1}, i.e., counter-clockwise order, $l(e_i)$ is the label of edge e_i, and $l(v_i)$ is the label of the neighboring vertex incident to edge e_i.

The spatial feature is created by selecting an edge from the vertex signature and going over the rest of the edges in a counter-clockwise order. For a vertex signature

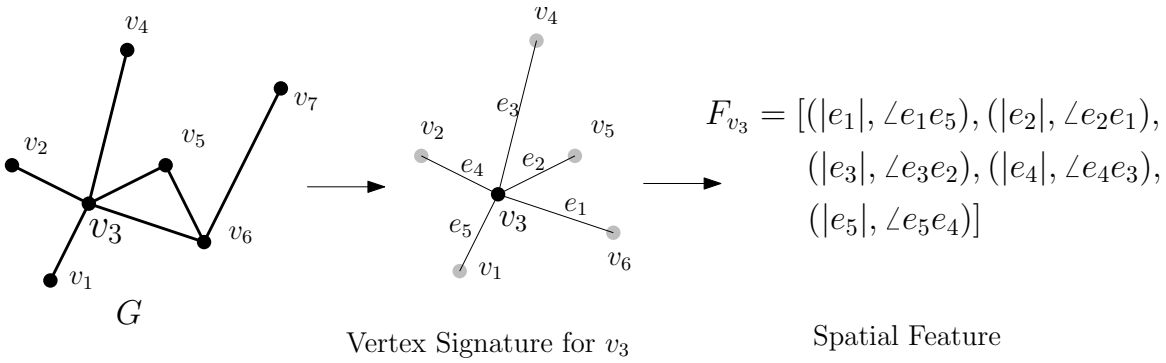


Figure 3.5: Feature extraction for vertex v_3. Initially, the vertex signature of v_3 is determined, then, a cyclic string of the spatial properties and labeling information of the neighborhood of v_3 defines its feature.

with n edges, there will be n different ways to represent its spatial feature. However, all of them are considered equivalent with a cyclic shift from one to another. Such a property enables the approximate solution to be computed faster than the optimal one.

The previous feature extraction method is illustrated in Figure 3.5. To extract the feature for vertex v_3, initially, the vertex signature is selected as the subgraph of vertex v_3 and its direct neighbors. After that, a spatial feature is extracted from that vertex signature. Notice that we use an unlabeled graph in the figure just for simplicity.

Once the spatial features are represented as cyclic strings, the VED between two vertex signatures is computed by the *cyclic string edit distance* (CS) [81], which is a natural extension of the string edit distance. A naive approach to solve it runs in $O(nm^2)$, where n and m are the numbers of edges of the two vertex signatures. This is done by applying the algorithm by Wagner and Fisher [120] to the first spatial feature and all cyclic shifts of the second one. Maes in [81] proposes a faster solution to the cyclic string edit distance that runs in time $O(nm \log m)$. Thus, the cyclic string edit distance solves the VED problem with a time complexity of $O(nm \log m)$.

To utilize the CS approach, we define three edge edit operations: substitution, insertion, and deletion. We propose edit operations that combine spatial attributes and labeling information. In the following we discuss two sets of edit operations. The first one computes the edit operations based on the absolute values of the edge length and the angel value. The second one uses a polar distance based on the lengths of two edges and the angle between them.

• Edit operations using the Manhattan distance

Given two vertex signatures $S(v)$ and $S(u)$ such that $n = |S(v)|$ and $m = |S(u)|$, let edge $e_i \in S(v)$, edge $e_j \in S(u)$, $f_i = (|e_i|, \angle e_i e_{i-1}, l(e_i), l(v_i))$, $f_j = (|e_j|, \angle e_j e_{j-1}, l(e_j), l(u_j))$, then, the substitution cost $c(f_i \rightarrow f_j)$ is defined as:

$$c(f_i \rightarrow f_j) := d_L(f_i, f_j) + d_S(f_i, f_j) \tag{3.3}$$

In the case of labeled graphs, the function $d_L(f_i, f_j)$ computes the distance between the label of edge e_i and the label of e_j, in addition to the distance between the labels of the vertices that are incident to them, i.e., v_i and u_j. The function $d_S(f_i, f_j)$ calculates the spatial distance based on the angle and the edge length. For an edge e, let θ_e and l_e denote the angle and edge length, as defined earlier in Definition 3.6, then, the function d_S is formally defined as follows:

$$d_S(f_i, f_j) \quad := \quad \frac{|\theta_{e_i} - \theta_{e_j}|}{2\pi} + \left| \frac{l_{e_i}}{\sum_{k=1}^{n} l_{e_k}} - \frac{l_{e_j}}{\sum_{k=1}^{m} l_{e_k}} \right| \tag{3.4}$$

The angles and the lengths of the edges at a vertex signature are normalized, as can be seen by the denominators used in Equation 3.4. An angle is normalized by 2π since the sum of angles at a local signature sums up to this value. Also, an edge length is normalized by the sum of the lengths of the edges at a local signature. For example, for a local signature $S(v)$, the edge length normalization factor is $\frac{l_{e_i}}{\sum_{k=1}^{n} l_{e_k}}$, where n is the number of edges connected to v.

In the following, we define the insertion and deletion operations. Let λ represent the null (non-existent) edge, then the insertion $c(\lambda \rightarrow f_i)$ and deletion $c(f_i \rightarrow \lambda)$ with respect to f_i are defined as follows:

$$c(\lambda \rightarrow f_i) = c(f_i \rightarrow \lambda) := c(f_i) + \left(\frac{\theta_{e_i}}{2\pi} + \frac{l_{e_i}}{\sum_{k=1}^{n} l_{e_k}} \right) \tag{3.5}$$

The cost of edge insertion or deletion is computed based on the angle value, edge length, and labeling information. For labeled graphs, the function $c(.)$ defines the cost of inserting or deleting the label assigned to that edge in addition to the label assigned to its incident vertex.

For unlabeled graphs, the cost of an edit operation lies in the range [0,2]. This is because each of the angle value and edge length is normalized to the range [0,1]. For labeled graphs, the range increases depending on the range of the function d_L for the

substitution operation and $c(f_i)$ for the insertion and deletion.

• Edit operations using polar coordinate

The second set of edit operations shares many similarities with the previously defined edit operations. However, the spatial distance between two vertex signatures is computed based on the polar distance between the neighboring vertices of two vertex signatures. Given two vertex signature $S(v)$ and $S(u)$, let edge $e_i \in S(v)$ and edge $e_j \in S(u)$. The substitution cost $c(f_i \rightarrow f_j)$ is defined as:

$$c(f_i \rightarrow f_j) := d_L(f_i, f_j) + d_S(f_i, f_j) \tag{3.6}$$

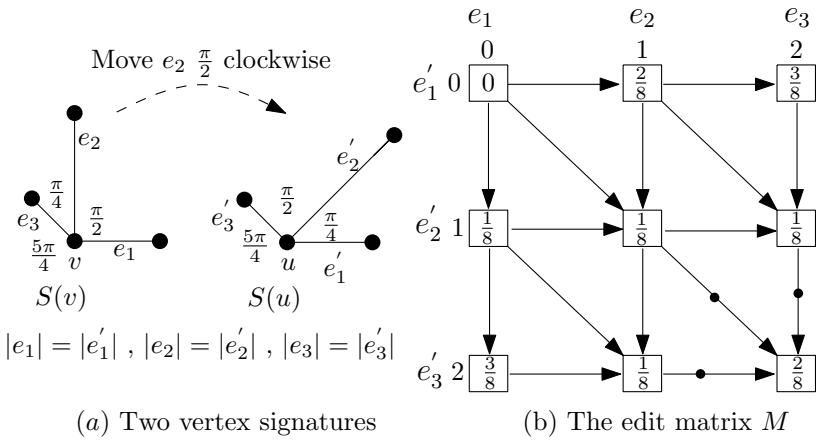
In the case of labeled graphs, the function $d_L(f_i, f_j)$ computes the distance between the label of edge e_i and the label of e_j. It also computes the distance between the label of the neighboring vertex connected to e_i to the label of the one connected to e_j. The function $d_S(f_i, f_j)$ calculates the spatial distance based on the angles and the lengths of the edges. For an edge e, let θ_e denote the angle between e and the previous edge in a counter-clockwise order, and let l_e denote the edge length. The function d_S is defined as:

$$d_S(f_i, f_j) := \sqrt{l_{e_i}^2 + l_{e_j}^2 - 2\ l_{e_i} l_{e_j} cos(|\theta_{e_i} - \theta_{e_j}|)} \tag{3.7}$$

The substitution cost is defined as the distance needed for the neighboring vertex of edge e_i to align with the neighboring vertex of e_j. To compute such a distance we utilize the lengths of the two edges and the angle between. This can be seen as the polar distance between them such as the polar axis for each vertex is the edge that precedes it in the counter-clockwise order. Analogously, we define the insertion and deletion operations. Let λ represent the null (non-existent) edge. Then, the insertion $c(\lambda \rightarrow f_i)$ and deletion $c(f_i \rightarrow \lambda)$ with respect to f_i are defined as:

$$c(\lambda \rightarrow f_i) = c(f_i \rightarrow \lambda) := c(f_i) + l_{e_i} \tag{3.8}$$

The cost of edge insertion or deletion is computed based on the edge length (l_{e_i}). For labeled graphs, the function $c(f_i)$ defines the cost of inserting or deleting the label assigned to the edge e_i in addition to the label assigned to the neighboring vertex connected to e_i.



(a) Two vertex signatures (b) The edit matrix M

Figure 3.6: The edit matrix M used in one of the iterations to compute the cyclic string edit distance between $S(v)$ and $S(u)$.

After having defined the three edge edit operations, the cyclic string edit distance is solved by using the dynamic programming paradigm as proposed by Wagner and Fischer [120]. An edit matrix is used to store the edit distance between incrementally increasing prefixes of two strings. In other words, the edit matrix stores the edit distance between all prefixes of the first string to all prefixes of the second one. At each cell in the edit matrix, the algorithm selects the edit operation that results in the minimum edit distance between the two prefixes.

Figure 3.6(b) shows the edit matrix M for computing the distance between the two vertex signatures given in Figure 3.6(a). Using the first set of edit operations, the edit path between the two vertex signatures is $(f_1 \rightarrow f_1'), (f_2 \rightarrow f_2'), (f_3 \rightarrow f_3')$. When calculating the cost at entry $m_{2,2} \in M$, the string edit distance takes the minimum of:

- $M[1,1] + c(f_3 \rightarrow f_3')$ (substitution),
- $M[2,1] + c(\lambda \rightarrow f_3)$ (insertion), and
- $M[1,2] + c(f_3' \rightarrow \lambda)$ (deletion)

which are highlighted by the solid circles on the arrows of Figure 3.6(b). Since $|e_3| = |e_3'|$, the substitution cost, using the Manhattan distance edit operations, is $c(f_3 \rightarrow f_3') = \frac{\frac{\pi}{2} - \frac{\pi}{4}}{2\pi} = \frac{1}{8}$, and the cost of the edit path is $\frac{1}{8} + \frac{1}{8} = \frac{2}{8}$, which is shown by entry $m_{2,2}$ of Figure 3.6(b).

Lemma 3.2. *The cyclic string edit distance finds a suboptimal solution to the VED problem.*

Proof. Let the string $S^{'} = (e_1, \ldots, e_{k-1})$ be the prefix of $S = (e_1, \ldots, e_n)$. Let the string $S^{''} = (e_k, \ldots, e_n)$ be the suffix of string S. The string edit distance considers the tokens of a cyclic string totally independent. In other words, any string edit operation applied to the prefix $S^{'}$ does not change the values of string $S^{''}$. This assumption is not sufficient and thus cannot be adopted when computing the cyclic string edit distance between two spatial features. The edit operation applied to edge $e_{k-1} \in S^{'}$ may change the angle of edge $e_i \in S^{''}$. For example, in Figure 3.6(a), if one assumes $F_1 = [f_1, f_2, f_3]$ is the spatial feature of $S(v)$ starting from edge e_1, $F_1^{'} = [f_1^{'}, f_2^{'}, f_3^{'}]$ is the spatial feature for $S(u)$ starting from edge $e_1^{'}$, then the edit path is $c(f_1 \rightarrow f_1^{'}) = 0$, $c(f_2 \rightarrow f_2^{'}) = \frac{1}{8}$ and $c(f_3 \rightarrow f_3^{'}) = \frac{1}{8}$. Whereas $(f_2 \rightarrow e_2^{'})$ means moving edge $e_2 \in S(v)$ by $\frac{\pi}{4}$ clockwise, this increases the angle $\angle e_2 e_3$ by $\frac{\pi}{4}$. As a result, the optimal cost $c(f_3 \rightarrow f_3^{'})$ is 0 and not $\frac{1}{8}$. This makes the cyclic string edit distance as proposed in [81] a suboptimal solution to the vertex edit distance in the context of geometric graphs. □

Note that the dependency between edges is only related to the angle value. On the other hand, two edges are independent with respect to the edge length.

Lemma 3.3. *The suboptimal solution to the VED problem, as discussed above, is a metric.*

Proof. Since the edit operations that are proposed in Equations 3.4, 3.5, 3.7 , and 3.8 are metrics, the cyclic string edit distance is also a metric [81]. Thus, the solution to the VED problem is a metric distance for vertex signatures in 2D space. □

In the previous two sections, we discussed approaches to tackle the vertex similarity problem for geometric graphs in 2D space. Initially, we discussed an approach to compute the optimal solution to the VED problem, i.e., the similarity of two vertices based on their vertex signatures. Such a solution utilizes spatial features that are invariant to geometric transformations. To enhance the runtime complexity, we proposed another approach that computes a suboptimal solution in cubic time with respect to the vertex degree. For this, a spatial feature from each vertex signature is extracted, which is represented as a cyclic string. Based on this, the cyclic string edit distance is used to compute the distance between two vertex signatures. To utilize the string edit distance concept, we proposed two sets of edit operations. The first one uses the absolute values of the edge length and the angle value. The second one uses the polar distance between two neighboring vertices from two vertex signatures.

3.4 Experimental Evaluation

In this section, our proposed approach to the vertex similarity problem and its usage for graph matching is empirically evaluated. Wince there are several approaches to compute the similarity between different vertices, our evaluation tries to answer the following questions:

1. Which approach is better, using a global-based or a local-based approach for vertex similarity?

2. Can we use only labeling information and neglect the spatial property of a graph to evaluate the similarity of different vertices?

3. Which is more accurate for vertex similarity: invariant spatial features or the coordinates of the vertices?

4. What is the relationship between the accuracy of a vertex similarity approach and its scalability as the graph size, the database size, and the vertex degree increase.

For our evaluations, we use geometric graphs that are extracted from different application domains such as chemoinformatic, computer vision, road networks, and character recognition. We use 6 different datasets: 1) the antiviral screen database of active compounds (AIDS) [42, 99], which represents a group of chemical compounds, 2) Chinese characters [1], 3) the GREC image dataset [39, 99], 4) the COIL-100 image dataset [87], 5) the CMU house and hotel image datasets [2], and 6) three road networks consisting of the California, city of Oldenburg, and North America road networks [4]. As shown in Table 3.1, besides coming from different application domains, our datasets vary in many aspects such as the size of the dataset, the number of classes (in case geometric graphs have been assigned to classes), as well as the number of vertices and edges.

The focus of this section is to empirically evaluate different vertex similarity approaches. We compare our algorithms **(CSv1)**, which uses the first set of edit operations, and **(CSv2)**, which uses the second set of edit operations, with four other approaches:

1. Graph spectra using the weighted adjacency matrix of the lengths of the edges **(SP)** [118]. To compare vertices from two graphs that differ in their sizes, the Eigenvectors of the larger graph is truncated by keeping the n dominant

Table 3.1: Statistics for the datasets used in our experiments. $|D|$ is the size of the dataset. $|\Omega|$ is the number of classes. ϕ is the average function. $|V|$ denotes the number of vertices in a graph. $|E|$ denotes the number of edges in a graph.

Dataset	$\|D\|$	$\|\Omega\|$	$\phi\|V\|$	$\phi\|E\|$
AIDS	2000	2	15	16
GREC	1000	22	12	12
Chinese	9384	1564	20	18
COIL-100	3900	100	21	53
CMU house	111	NA	30	76
CMU hotel	101	NA	30	92
Road network	3	NA	4125	5475

Eigenvectors, such that n is the size of the smaller graph [141]. We compare with the graph spectral approach to evaluate the performance of global-based vertex similarity approaches in general.

2. Geometric histogram of the pair-wise relations between the edges in the neighborhood of a vertex **(GH)** [58, 116], where the pair-wise relation between two edges is defined as the ratio of the lengths of the edges and the angle between them. For binning information, we used 10 bins for the edge length ratio and 18 bins for the angle value. Also, we normalize the histogram by its number of entries. Then, the distance between two vertices is computed by the χ^2 distance of their geometric histograms.

3. Graph edit distance based on only labeling information **(GED)** [134]. The distance between two vertices is computed by the *star edit distance* of the labeling information of two star structures, i.e., vertex signatures. For two vertices v and u, such a distance function is computed as the difference of their vertex degrees, in addition to the differences of the labeling information of their vertex signatures, $S(v)$ and $S(u)$. The **GED** approach is used to evaluate the contribution of the labeling information to the similarity of different vertices, in the case of labeled geometric graphs.

4. A unary vertex distance function based on only the coordinates of the vertices **(CO)**, thus neither global nor local structural information is used. We test such an approach to evaluate the effect of using the coordinates of the vertices on their similarities.

For all the approaches and for labeled geometric graphs, the distance between numeric labels, which are assigned to the vertices and the edges, is computed by the Manhattan distance. The distance between non-numeric labels is 0 if they are identical and 1 otherwise.

To compare the different approaches, we embed them in a unified graph matching algorithm. It consists of two steps. First, a vertex-to-vertex distance matrix is created using any of the previous vertex similarity approaches. Second, the Hungarian algorithm [85] is used to select the best match between the two graphs. Since all the approaches use the Hungarian algorithm for graph matching, the differences in the matching results are affected by only the approach that is used to estimate the similarity between two vertices.

To measure the performance of a graph matching algorithm, which indicates the performance of a vertex similarity approach, we use different criteria. The first one is the effect of vertex similarity on a graph similarity metric. This is evaluated by embedding the graph matching algorithm in a classification task. The higher the classification accuracy the better the vertex similarity approach is, which is formalized as:

$$\text{classification accuracy} = \frac{\text{the number of graphs correctly classified}}{\text{the total number of graphs}} \tag{3.9}$$

To create a graph distance metric, we follow a graph edit distance approach. This means that the distance between two graphs consists of the cost of the match between them, i.e., the substitution cost, in addition to the cost of inserting the unmatched vertices. The second criterion is selectivity power, which means that a vertex similarity approach reflects the similarity notion of an application domain. This is measured by the quality of the match computed by the graph matching algorithm. The third criterion is scalability with respect to the vertex degree, the number of vertices, and the size of the dataset. We measure this by the runtime required to match different graphs.

3.4.1 Graph Similarity and Classification

In this section, we evaluate the relation between different vertex similarity approaches and graph similarity in general. To measure this, we test the different approaches in a graph classification task. The higher the classification accuracy, the better the vertex similarity approach. In our experiments, we used the first nearest neighbor classifier

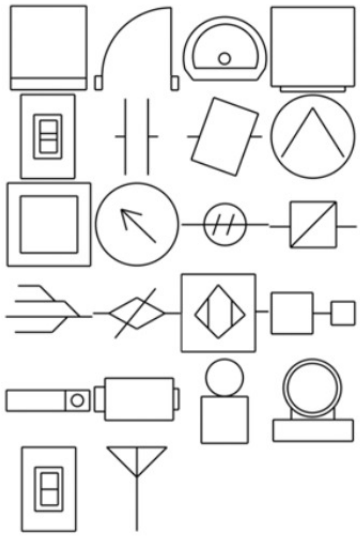
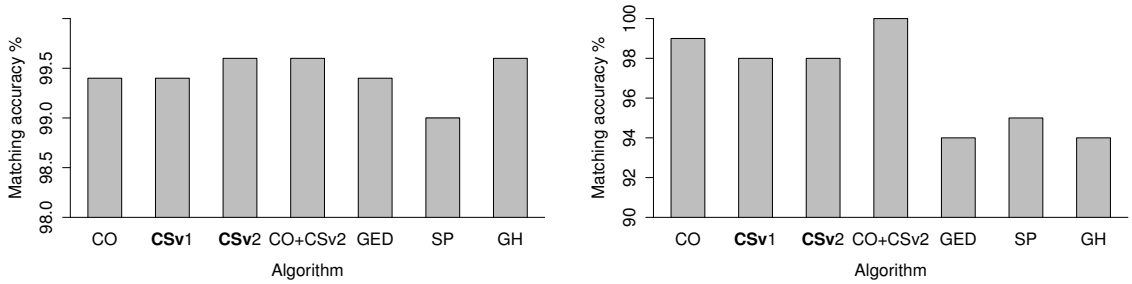


Figure 3.7: Sample images of the GREC dataset [39, 99, 103].

(1-NN) based on the similarities of the graphs. Four datasets are used: two of them contain labeled geometric graphs, AIDS and GREC, and the other two, Chinese and COIL-100, contain unlabeled geometric graphs.

The AIDS dataset has chemical compounds that belong to one of two classes: active or inactive against HIV. A graph is extracted from each compound. From a total of 2000 graphs, we use a training set of 1500 graphs: 300 active and 1200 inactive. For testing, we use a set of 500 graphs: 100 from the active class and 400 from the inactive class. On the other hand, the GREC dataset has images of architectural and electronic drawings. From the GREC dataset, 22 (images) classes are selected, which are shown in Figure 3.7. A training set of 814 and a testing set of 286 images are used. For each class, 37 images for training and 13 for testing are chosen.

In Figure 3.8(a), the classification accuracy for the AIDS dataset is shown. From the figure, we see that all approaches have nearly the same classification accuracy ranging from 99% to 99.6%. The first observation we report is that using only the labeling information (**GED**) is sufficient to obtain a high classification accuracy. Also, using the spatial properties alone, i.e., **CSv1**, **CSv2**, and **GH**, gives the same result. This is justified since for chemical compounds, an atom with a specific type is connected to the neighboring atoms with the same number of covalent bonds, i.e., edges. Also, it is always the case that the angle between two covalent bonds is identical



(a) AIDS dataset (b) GREC dataset

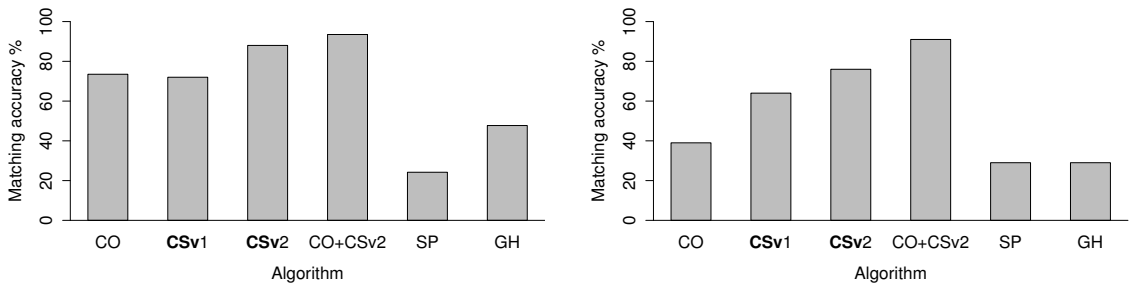
Figure 3.8: Classification accuracy for labeled geometric graphs.

for the same atom. In other words, the spatial property of the graph is encoded by the labeling information.

Figure 3.8(b) shows the classification accuracy for each approach for the GREC dataset. The first observation is that using only the coordinates of the vertices (**CO**) is highly reliable for this dataset. This is because all images are measured with respect to nearly the same coordinate frame. On the other hand, using only the labeling information (**GED**) gives less classification accuracy than using the spatial property of the vertices. However, the difference between the classification accuracy of **CO** and **GED** is statistically insignificant. The best classification accuracy is for the **CO+CSv2** approach, which has an accuracy of 100%.

From these two labeled geometric graph datasets, we conclude that using only labeling information is sufficient to estimate the similarity between two vertices. This is because the labeling information, most of the time, encodes the spatial property of the graph.

Let us move our attention to unlabeled geometric graphs, where we exclude the label-based vertex similarity approach (**GED**). For this experiment, we use the COIL-100 dataset, which consists of images of 100 different objects taken at different degrees. From 3900 graphs, we select 2900 for training, 29 graphs for each object. For testing, we select 1000 graphs, 10 graphs for each object. We also use the Chinese dataset, which contains a total of 9384 characters that belongs to 6 different fonts, i.e., 1564 characters from each font. A test dataset of 1564 graphs is extracted from the Dotum Korean font. The remaining five fonts build a training dataset of 7820 graphs. Ideally, for a query character, its most similar character from the train dataset must have the same Unicode.



(a) COIL-100 dataset (b) Chinese dataset

Figure 3.9: Classification accuracy for unlabeled geometric graphs.

Figure 3.9(a) shows the classification accuracies for the COIL-100 dataset for different approaches. The least classification accuracy is for the **SP** approach. This is because spectral approaches are sensitive to changes in the number of vertices in addition to their spatial properties. We conclude that a local-based vertex similarity approach is better than a global-based one. The best classification accuracy is for the **CO+CSv2** approach, followed by the **CSv2** approach. Notice that the use of invariant spatial features by our approaches (**CSv1** and **CSv2**) gives better results than using the coordinates of the vertices (**CO**). However, combining both of them gives the best result.

The classification accuracy for the last dataset, i.e., the Chinese dataset, is presented in Figure 3.9(b). Also, for this dataset, the best result is for the **CO+CSv2** approach. On the other hand, the **CSv1** and **CSv2** approaches are much better than **CO** alone. The least classification accuracy is for **SP** and **GH**.

From these two datasets we conclude that using invariant spatial features is better than using only the coordinates of the vertices. However, and for many applications, still, the coordinates of the vertices can be used to give good graph matching results. Also, using the second set of edit operations, i.e., **CSv2**, gives better results than the first set. i.e., **CSv1**. This is justified since **CSv1** gives the same weight for the differences in the angle value and the edge length.

3.4.2 Graph Matching

In this section, we test the selectivity power of a vertex similarity approach. This means whether an approach reflects the similarity notion of an application domain or not. To evaluate this, we measure the quality of the match computed by the graph matching algorithm. Higher matching quality indicates higher selectivity power for

the vertex similarity approach. We use the matching accuracy to evaluate the quality of a match. For two graphs G and Q, the matching accuracy is estimated by the agreement between the match M that is computed by a graph matching algorithm, and the ground truth match $\hat{M}$. Given that M and $\hat{M} \in \{0,1\}^{|G|\times|Q|}$, the matching accuracy is formalized as:

$$\text{matching accuracy} := \frac{\sum\limits_{i\in|G|,j\in|Q|}(\hat{m}_{ij} \times m_{ij})}{\sum\limits_{i\in|G|,j\in|Q|}\hat{m}_{ij}} \tag{3.10}$$

under the conditions:

$$1)\ \forall j \in |Q|, \left(\sum_{i\in|G|} m_{ij}\right) \leq 1,\ \text{and}\ \forall i \in |G|, \left(\sum_{j\in|Q|} m_{ij}\right) \leq 1$$

$$2)\ \forall j \in |Q|, \left(\sum_{i\in|G|} \hat{m}_{ij}\right) \leq 1,\ \text{and}\ \forall i \in |G|, \left(\sum_{j\in|Q|} \hat{m}_{ij}\right) \leq 1$$

Notice that the last two conditions mean that each vertex from G is matched to only one vertex from Q and vise versa. Also, a vertex from G or Q is given the chance not to be matched to any other vertex from the other graph.

For this test, we use the CMU hotel and house datasets [2]. It consists of images for a toy house and hotel, subjected to rotation in 3D. As a result, each object creates a sequence of images taken at different angles of view. Given two images of the same object taken from two different angles of view, as shown in Figure 3.10, the task is to use graph matching to map features from the first image to their similar features from the second image. For example, if one take an image of a house and change his position and take another image, the task is to use graph matching to map the window from the first image to the same window in the second image. Notice that, the higher the difference in the angle of view the harder the task is. For our experiment, we match all images spaced at 10, 20, 30, 40, 50, 60, 70, 80, and 90 in the rotation sequence, and compute the average matching accuracy.

From Figures 3.11(a) and 3.11(b), one can see that for all the approaches, the matching accuracy decreases when the distance in the rotation sequence between the images increases. This is a consequence of the increase in the structural difference between the geometric graphs. The lowest matching accuracy is for the **SP** approach, which is sensitive to the changes in the structure of the graphs. The best matching accuracy is for **CSv2** and **CO**. However, **CO+CSv2** is not always better than using

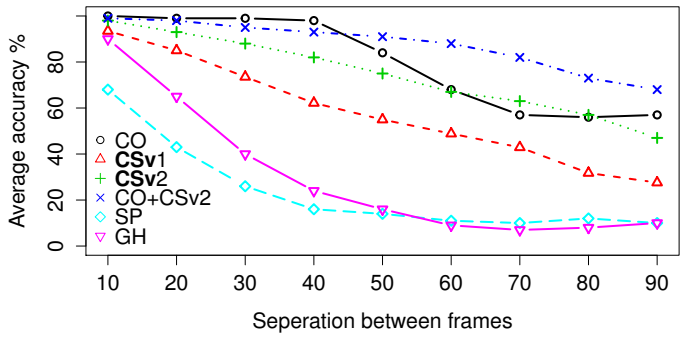
Figure 3.10: Two images taken at different angles of view [2].

either of the single approaches alone. Also, the matching accuracy of **CSv2** is better than **CSv1**. This means that using the polar distance gives better results than using just the absolute values of the length of the edge and the angle value.

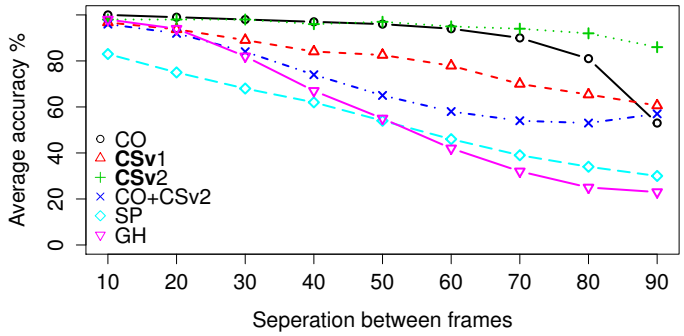
3.4.3 Subgraph Matching

In all the previous experiments, two graphs that are nearly the same size are compared to discover whether they are similar or not. However, for many applications it is critical to answer subgraph queries. Given a small graph, the problem is to find a subgraph from a larger graph that is highly similar to the smaller one. For this, we use the California road network. From this road network, we extracted manually 5 subgraphs from different locations of the road network. For each subgraph, a local coordinate frame is created and all the vertices are measured with respect to it. This demonstrates the scenario when different graphs are measured with respect to different coordinate frames. We matched all subgraphs against the California road network and averaged the matching accuracy.

Figure 3.12 shows that the best matching accuracy is for the approaches **CSv1** and **CSv2**. In the third place comes the histogram approach (**GH**). On the other hand, all other approaches have nearly zero matching accuracy. From this test, we report the following two observations. First, using the coordinates of the vertices is totally unreliable for graphs measured with different coordinate frames. As a result, combining **CO** and **CSv2** gives very low matching accuracy. Second, spectral graph matching approaches have very poor matching accuracy in the case of matching two graphs that have a large difference in the number of vertices.



(a) Hotel dataset



(b) House dataset

Figure 3.11: Matching quality for the CMU hotel/house datasets.

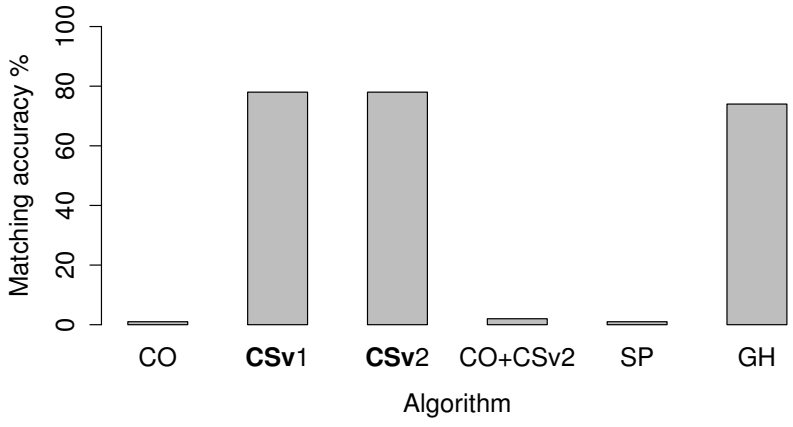


Figure 3.12: Average matching accuracy for subgraph matching using the California road network.

Table 3.2: Worst case runtime complexity for computing the distance between two vertices using different vertex similarity approaches. n is the average graph size, d is the average vertex degree, and l and a are the number of bins for a geometric histogram.

Approach	Complexity
CO	$O(1)$
CSv1	$O(d^3)$
CSv2	$O(d^3)$
CO+CSv2	$O(d^3)$
GEM	$O(d)$
SP	$O(n^3)$
GH	$O(la)$

3.4.4 Scalability Study

In this section, we evaluate the scalability of different vertex similarity approaches with respect to the vertex degree, the number of vertices, and the size of the dataset. We measure this by the runtime required to match different graphs. Table 3.1 compare the scalability of the different approaches. It shows the worse case runtime complexity for computing the similarity between two vertices. Let n denote the average size of the graph and d the average degree of a vertex, our approaches **CSv1** and **CSv2** run in cubic time with respect to the vertex degree, which is related to the complexity of solving the cyclic string edit distance. The spectral approach **SP** runs in cubic time with respect to the size of the graph. This complexity is the result of computing the Eigenvectors for each graph. All the other approaches can be considered to have a constant runtime complexity. For example, the worst case runtime complexity for the **GH** is related to the number of entries in the geometric histogram and equals to $O(la)$, where l and a are the dimensions of the histogram. **CO** and **GED** have the least runtime complexity.

In Figure 3.13(a), we show the scalability with respect to the vertex degree. For this test, we use a dataset that consists of randomly generated geometric graphs [3]. We generate 7 graphs, each with 100 vertices. To study the effect of the vertex degree, i.e., the density of the graph, we increase the vertex degree from 2 to 15 as shown in Figure 3.13(a). Our approaches **CSv1** and **CSv2** have poor scalability with respect to the vertex degree. This is because computing the similarity of two vertices runs in cubic time complexity with respect to the vertex degree. Also, there is a remarkable difference in the scalability between **CSv1** and **CSv2**. This comes from the use of

the cosine function by **CSv2**, which requires more CPU time than just using the absolute values of the edge length and the angle value, as used by **CSv1**. All other approaches scales well with respect to the vertex degree.

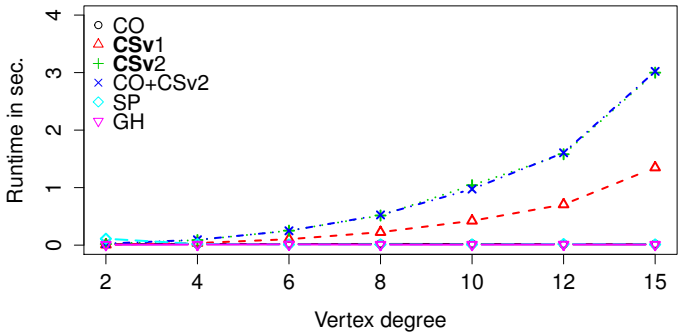
To study the scalability with respect to the graph size, we report the runtime of matching three road networks: California with 1365 vertices, city of Oldenburg with 3494 vertices, and North America with 7517 vertices. Each graph was matched against itself and the runtime was reported. Figure 3.13(b) shows that the worst scalability is for the spectral approach **SP**. The result of this experiment matches our analytical study in Table 3.1. On the other hand, the **GH** approach has the best scalability with respect to graph size. The **CSv1** scales much better than the **CSv2**, which is also related to the use of the cosine function by the latter approach.

Finally, we report the relation between the size of the dataset and the runtime for different approaches in Figure 3.13(c). In general, all related approaches scale better than our approaches **CSv1** and **CSv2**. However, our approaches give the best results in terms of classification accuracy and matching quality. The best runtime result was for the **GH** approach. The runtime for **GED** is reported for only GREC and AIDS since only these two datasets contain labeled graphs. Even though the Chinese dataset has more graphs than COIL-100, the runtime for **CSv1** and **CSv2** is higher for the latter one. This is because the average vertex degree for the graphs in COIL-100 is greater than the graphs in Chinese. Since our approaches have cubic time complexity with respect to the vertex degree, their runtime for the COIL-100 is greater than the Chinese. From the Chinese dataset, we also see that our approach **CSv1** scales similarity to the related approaches. We conclude that for sparse graphs, **CSv1** scales well with respect to the dataset size.

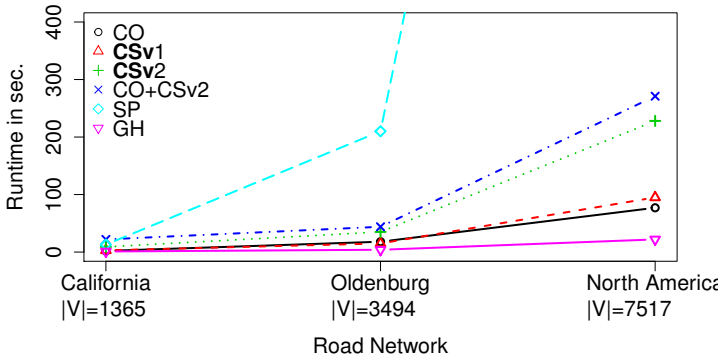
Our scalability study shows that our approach **CSv2** does not scale well with respect to the vertex degree. On the other hand, our approach **CSv1** scales better than **CSv2**. This is because of the use of the cosine function in computing the edit operations by **CSv2**. However, our approaches gives much higher classification accuracy and matching quality than the related approaches.

3.5 Summary and Discussion

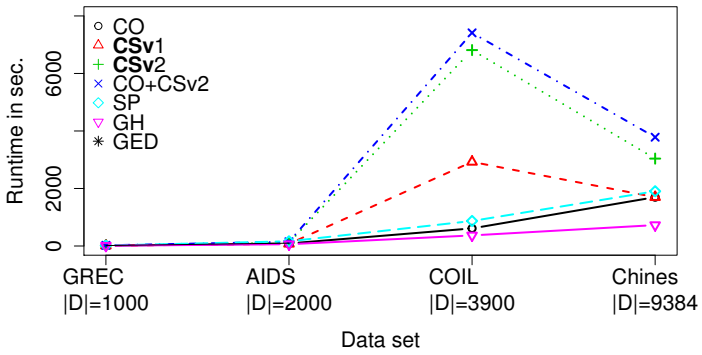
In this chapter, we discussed the problem of vertex similarity for geometric graphs. Our discussion focused on local-based vertex similarity approaches, which use the properties of the neighborhoods of different vertices to estimate their similarities. One of the main results of our study is the proof that the problem of vertex similarity



(a) Vertex degree



(b) Graph size



(c) Dataset size

Figure 3.13: Scalability with respect to different parameters.

for geometric graphs is an NP-hard problem. On the other side, we proposed an algorithm to approximate the similarity between vertices for geometric graphs in 2D space. Our solution utilizes the property that the direct neighbors of a vertex have a total order, which is a consequence of the embedding of the neighboring vertices in 2D space. To find the similarity between two vertices, first, a spatial feature is extracted from the neighborhood of each vertex, which is a cyclic string of edges. After that, the cyclic string edit distance is used to estimate the similarity of different vertices. For this, we proposed two sets of edit operations. The first set uses the absolute values of the edge length and the angle value. The second set uses the polar distance between the neighboring vertices. Experimentally, we tested our algorithm with several data sets coming from a variety of application domains such as chemoinformatic, computer vision, road networks, and character recognition. In the following, we summarize our experimental results for the vertex similarity problem of geometric graphs in 2D space.

1. For labeled geometric graphs, using only the labeling information to estimate the similarity of different vertices gives good results. In other words, there is no need to include the spatial property of a graph for vertex similarity.

2. For several applications, such as chemoinformatic and some applications in image processing, using the coordinates of the vertices are reliable to estimate the similarity of different vertices. This is because in such applications, the coordinates of the vertices of different graphs are measured with respect to nearly the same coordinate frame.

3. In general, local-based vertex similarity approaches give better results than global-based approaches. This is because for scientific applications, two similar objects are model by two graphs that have many differences on the global graph level, but have many similarities with respect to the neighborhoods of the vertices. For example, given images of a car taken at different timestamps, the spatial properties of the car remain the same across different images, however, the background of the image has many changes since the car is moving from one place to another.

4. Nearly for all the datasets that we used in our experiments, using spatial features from the neighborhoods of the vertices gives better results than using only the coordinates of the vertices.

5. Our solution to the vertex similarity problem had the best results in terms of the classification accuracy and the matching quality. However, it requires more runtime than the related approaches especially for dense graphs.

Chapter 4

Geometric Graph Matching: A Probabilistic Approach

Motivated by a variety of graph-based applications, in the previous chapter, we discussed the vertex similarity problem, which is the basis of any graph matching algorithm. For geometric graphs in 2D space, we propose to compute the similarity between different vertices based on the similarity of their neighborhoods. We also propose a naive approach to tackle the graph matching problem, where a vertex-to-vertex similarity matrix is initially created, and then, the Hungarian algorithm is used to select the best match between the two graphs.

Unfortunately, matching two graphs based on only the neighborhoods of the vertices neglects the overall graph structure. As a result, the computed match is structurally incompatible. To give a feeling of this problem, we show an example in Figure 4.1. The left part of the figure shows the vertex signatures for the two graphs G and Q. The right part shows the match computed by the Hungarian algorithm using only the similarity of the neighborhoods of the vertices, i.e., our algorithm **CSv2**, discussed in the previous chapter. When only the neighborhoods of the vertices are considered, vertex $v_4 \in G$ is more similar to $u_7 \in Q$ than $u_4 \in Q$. However, when the overall graph structure is considered, vertex $v_4 \in G$ is matched to $u_4 \in Q$ even though their vertex signatures are not highly similar.

To solve this problem, in this chapter, we propose a probabilistic graph matching algorithm that combines both the similarity of the vertices based on their vertex signatures and the overall graph structure. Our algorithm is inspired by the approach of Sanromá *et al.* [109]. They propose a graph matching algorithm based on the assumption that the vertices of a graph G are sampled from a mixture model of the vertices of another graph Q. As a result, the graph matching problem is formalized

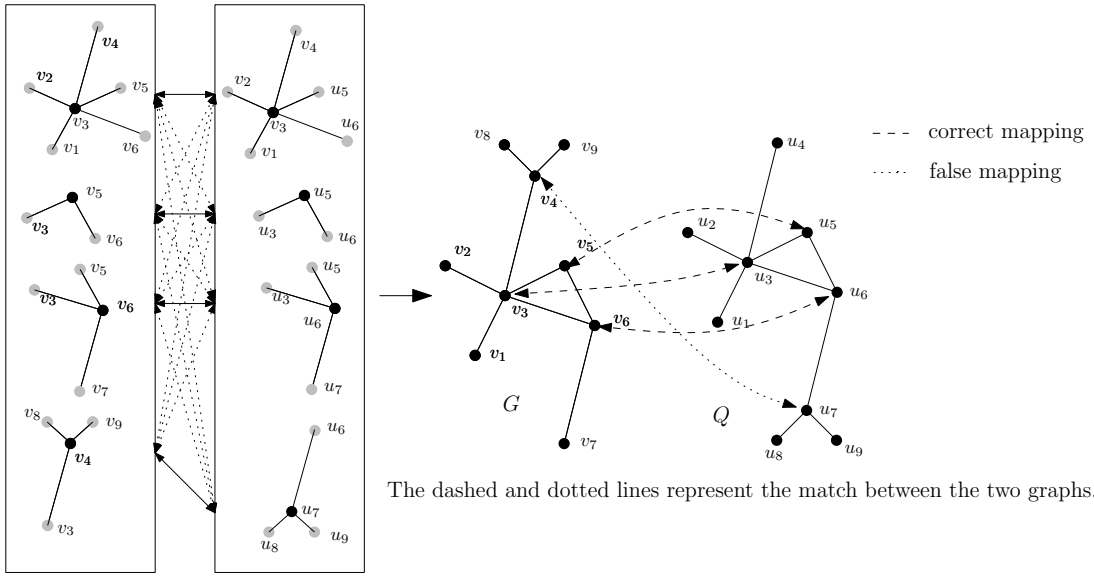


Figure 4.1: Considering only the neighborhoods of the vertices makes $v_4 \in G$ highly similar to $u_7 \in Q$. For simplicity, not all vertex signatures are shown.

as a *parameter estimation problem*. To be more specific, the graph matching problem is formalized as a maximum likelihood estimation (MLE) problem. Then, the expectation maximization (EM) technique is used to select the parameters of the mixture model that maximize the likelihood function.

In this chapter, we extend such a model and propose a novel density function for geometric graphs in 2D space that considers the spatial properties in addition to the structure of the graphs. Our density function consists of two components: 1) a *vertex similarity* measure based on the vertex edit distance concept, which is discussed in Chapter 3, and 2) a *shortest path similarity* function that estimates the similarity between two vertices based on their connectivity to other non-neighboring vertices. There are three main advantages of our model over the original model of Sanromá *et al.* [109]:

1. The ability to estimate subgraph and common subgraph matchings. Especially for the latter case, our algorithm employs several pruning techniques to exclude the vertices of the non-common subgraph from the match computed between different graphs.

2. The utilization of the overall graph structure in estimating the similarity of different vertices. We adopt this using a shortest path similarity function between

two vertices based on their connectivity to other non-neighboring vertices. However, Sanromá *et al.* [109] utilize only the overall spatial properties of a graph without considering its overall structure.

3. The scalability in terms of graph size. This includes both runtime and memory consumption. A major problem of the original model is that the density function is not manageable and goes to infinity as graph size increases, which we will detail later in this chapter. Our proposed algorithm is made scalable with respect to graph size by adopting the following techniques:

   - Computing the density function based on a sub-space of the mixture model, which is what we call the *k-active subgraph*. This solves the problem of the non-manageable behavior of the density function of the original model.
   - The ability to estimate the spatial similarity of two graphs without the need to compute the geometric transformation that maximizes their similarity. Our approach accomplishes this by utilizing the spatial features that are extracted for the vertices of each graph. On the other side, the original probabilistic graph matching algorithm computes a geometric transformation for each pair of vertices from the two graphs and at each iteration of the algorithm.

In the following, Section 4.1 introduces concepts related to the maximum likelihood estimation (MLE) problem and the expectation maximization technique (EM). In Section 4.2, we detail the formalism of the graph matching problem as MLE and propose a novel graph density function. In Section 4.3, we discuss how to use the EM technique to compute the match between two graphs. Our experimental evaluations are presented in Section 4.4. Finally, we summarize the chapter in Section 4.5.

4.1 Preliminaries

In many scientific domains, such as bioinformatic or social media analysis, people are interested in discovering the laws behind certain behaviors of the system under study. Unfortunately, such laws and principles are not directly observed. Instead, a mathematical model is used to describe the internal structure of the system that leads to a specific behavior. Such a mathematical model is mainly represented by parametric families of probabilistic distributions (PD) such as the Gaussian or the Poisson distributions. Since a family of PDs represents a wide variety of systems with

different behaviors, *parameter estimation* is used to select the parameters of the PD that fit some sample data taken from the system.

In the following sections we discuss some general concepts that are going to be used through the rest of this chapter. First, in Section 4.1.1, we discuss maximum likelihood estimation as a solution to the parameter estimation problem. Second, in Section 4.1.2, we introduce the expectation maximization technique, which is used to maximize a likelihood function of a mixture model that has latent variables or hidden data.

4.1.1 Maximum Likelihood Estimation

Maximum likelihood estimation (MLE) is one of the famous methods that is used for parameter estimation. The main idea is to consider the parameters of a PD as fixed hidden values. Then, they are estimated by maximizing the likelihood such that the PD generates some given sample data. Suppose we have a data set of samples $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$, where each sample x_i is a vector in m dimensional space, i.e., $x_i = (x_{i1}, x_{i2}, \ldots, x_{im})$. Here, we assume that the samples are independent and identically distributed (i.i.d). Suppose the parameters of the model are represented by the vector $\theta = (\theta_1, \theta_2, \ldots, \theta_c)$, then the *likelihood function* that the sample data was generated from a PD with parameters θ is formalized as:

$$P(\mathcal{X}|\theta) = \prod_{x_i \in \mathcal{X}} P(x_i|\theta) \tag{4.1}$$

Notice that the likelihood of the data is factorized as the product of the likelihoods of the single data points, which is a consequence of the assumption that our sample data is i.i.d. Normally, the logarithm of the likelihood function is maximized to estimate the parameters θ^* that make the model best fit the sample data $\mathcal{X}$. This is because the logarithm function is a monotonically increasing function. As a result, the parameters that maximize the logarithm of the likelihood function are the same parameters that maximize the likelihood itself. On the other hand, the benefit we gain is that the logarithm function is always easier to deal with from an analytical point of view. As a result, the maximum likelihood estimation problem is formalized as:

$$\theta^* = \arg\max_{\theta \in \Theta} \ln P(\mathcal{X}|\theta)$$

$$= \arg\max_{\theta \in \Theta} \sum_{x_i \in \mathcal{X}} \ln P(x_i|\theta) \tag{4.2}$$

such that Θ is the space of all parameters and ln(.) is the natural logarithm function. In case that $P(\mathcal{X}|\theta)$ is well-behaved, θ^* can be computed using standard deferential calculus, i.e., θ^* is determined when the gradient of the likelihood function with respect to θ equals to zero.

Until now we have been talking about a system that is modeled by just one probability distribution. However, such an assumption, usually, fails to model a complex system that is composed of sub-systems where each has its own laws and principles. A more powerful way to model such a complex system is to use a mixture of probabilistic distributions such that each PD models a sub-system. The major problem that faces such an assumption is that we cannot observe which sub-system leads or controls an observation. In this case, the mixture model is said to have *latent variables* or *hidden data*. Let $\mathcal{X}$ denote the observed data, and let Z be the latent variables, i.e., the random variables that define which model generates a specific sample data point, then we call $\{\mathcal{X}, Z\}$ the *complete data*, and $\mathcal{X}$ the *incomplete data*. In addition to this, the likelihood function of the complete data is denoted as $P(\mathcal{X}, Z|\theta)$, and the likelihood function for the incomplete data is denoted as $P(\mathcal{X}|\theta)$. The latter probability is considered as a marginal distribution with respect to the latent variable. The likelihood of the incomplete data is formalized in terms of the complete data as follows:

$$\begin{aligned} P(\mathcal{X}|\theta) &= \prod_{x_i \in \mathcal{X}} P(x_i, Z|\theta) \\ &= \prod_{x_i \in \mathcal{X}} \sum_{z_j \in Z} P(x_i, z_j|\theta) \end{aligned} \tag{4.3}$$

Notice that the product comes from the assumption that the data is i.i.d. and the summation is related to the marginal distribution. By using the logarithm function, the likelihood of the incomplete data is written as:

$$\ln(P(\mathcal{X}|\theta)) = \sum_{x_i \in \mathcal{X}} \ln \left(\sum_{z_j \in Z} P(x_i, z_j|\theta) \right) \tag{4.4}$$

The key observation of Equation 4.4 is that the summation over the latent variable is inside the logarithm function. As a result, it is very hard to analytically find

the parameters that maximize the logarithm likelihood of the incomplete data. To solve this problem, normally, a suboptimal solution to the maximization problem is computed using the *expectation maximization* technique, which will be outlined in the following section.

4.1.2 Expectation Maximization

Expectation maximization (EM) is a technique proposed by Dempster *et al.* [36] to maximize a likelihood function in the case of hidden data. They prove that maximizing the log-likelihood function is equivalent to maximizing a weighted sum of the complete log-likelihoods where the weights are the a posteriori of the hidden data. Such a weighted sum can be iteratively maximized using the EM technique such that each iteration has two steps. At iteration $t+1$, the expectation step (E-step) computes the a posteriori probability of the hidden data $P(z_j|x_i,\theta^{(t)})$ using the observed data and the previous estimation of the parameters $\theta^{(t)}$. In the maximization step (M-step), the parameters of the model $P(x_i,z_j|\theta^{(t+1)})$ are estimated using the observed data and the a posteriori probabilities of the hidden data. As a consequence, the objective function $\Lambda(\theta^{(t+1)}|\theta^{(t)})$, which is used by the EM technique, can be formulated as:

$$\Lambda(\theta^{(t+1)}|\theta^{(t)}) = \sum_{x_i\in\mathcal{X}}\sum_{z_j\in Z} \underbrace{P(z_j|x_i,\theta^{(t)})}_{\text{E-step}} \underbrace{\ln P(x_i,z_j|\theta^{(t+1)})}_{\text{M-step}} \tag{4.5}$$

Notice that the EM technique does not directly optimize the log likelihood function $\ln(P(\mathcal{X}|\theta))$, instead, it optimizes the objective function $\Lambda(\theta^{(t+1)}|\theta^{(t)})$. As a result, the solution computed by EM is not guaranteed to be a maximum likelihood estimator. However, the parameters that improve the objective function $\Lambda(\theta^{(t+1)}|\theta^{(t)})$ implicitly improve the log likelihood function $\ln(P(\mathcal{X}|\theta))$.

In many cases, the M-step of the EM technique cannot be maximized in closed form. As a result, the generalized expectation maximization (GEM) is proposed such that to select $\theta^{(t+1)}$ that improves the object function but not to maximizes it [36], which is formally written as follows:

$$\Lambda(\theta^{(t+1)}|\theta^{(t)}) \quad \geq \quad \Lambda(\theta^{(t)}|\theta^{(t)}). \tag{4.6}$$

The convergence of both EM and GEM is guaranteed. This because their objective functions are monotonically increasing [129]. However, the convergence speed of GEM is faster than EM.

4.2 Graph Mixture Model

Given two graphs G and Q with their vertex sets V and U, respectively, a match between the two graphs can be represented as a function $f : V \to U$. $f(v_i) = u_j$ denotes that vertex $v_i \in V$ corresponds to vertex $u_j \in U$. Using the function f, the match between the two graphs can be further represented as a matrix $M \in \{0,1\}^{|V|\times|U|}$. An entry m_{ij} is an *assignment variable* indicating whether vertex $v_i \in G$ corresponds to vertex $u_j \in Q$, which is defined as:

$$m_{ij} = \begin{cases} 1, & \text{if } f(v_i) = u_j \\ 0, & \text{otherwise} \end{cases} \tag{4.7}$$

The graph matching problem can be formalized as a maximum likelihood estimation problem. We assume that the vertices of the two graphs are generated by a process such that first, a vertex from Q is selected, and then, its corresponding vertex from G is generated. Following this assumption, the vertices of graph G are considered the observed data and the corresponding vertices from Q are the hidden data. Notice that the matching function f is unknown, although we are able to observe graph Q. A mixture model is then defined based on the vertices of graph Q. This means that any vertex $v_i \in V$ may be generated from a vertex $u_j \in U$. As a result, M can be seen as a mixture model parameterized by the set of assignment variables m_{ij}. It is worth to notice that in case $m_{ij} = 1$, the probability that v_i is generated from vertex u_j is 1. Following these assumptions, the solution to the graph matching problem is the mixture model M^* that maximizes $P(G|M)$, which is the incomplete-data likelihood of the observed graph. This is formalized as:

$$M^* = \arg\max_{M\in\mathcal{M}} P(G|M) \tag{4.8}$$

where $\mathcal{M}$ is the space of all possible matches between the two graphs. In general, there are $2^{|G||Q|}$ different matches and thus mixture models. In the case where we restrict the match to be one-to-one, the space of possible matches is reduced to $\frac{|G|!}{(|G|-|Q|)!}$ s.t. $|G| \geq |Q|$. Under this assumption, any vertex of G has the chance to be matched to any vertex of Q.

To define the likelihood function we assume the vertices of graph G to be i.i.d., given the match M. As a result, the likelihood function is factorized over the observed data (the vertices of G) and summed over the hidden data (the corresponding vertices

from Q), as follows:

$$P(G|M) = \prod_{v_i \in V} \sum_{u_j \in U} P(v_i, u_j|M) \tag{4.9}$$

where $P(v_i, u_j|M)$ is the density function of the complete-data likelihood. It represents the probability that vertex v_i corresponds to vertex u_j given the matching matrix M.

4.2.1 MLE for Graph Matching

In the following, we factorize the density $P(v_i, u_j|M)$ to the single assignment variables in a similar way as proposed in [80, 109]. Using Bayes' rule we have:

$$P(v_i, u_j|M) = \frac{P(v_i, u_j, M)}{P(M)} = \frac{P(M|v_i, u_j)\ P(v_i, u_j)}{P(M)} \tag{4.10}$$

Since M is treated as a mixture of the models, we assume the models to be independent of each other. This means that the probability of matching any two vertices v_i and u_j is not affected by the connectivity between the vertices of M. As a result, $P(v_i, u_j|M)$ is factorized over the parameters of the mixture model, i.e., the assignment variables m_{ij}.

$$\begin{aligned} P(v_i, u_j|M) &= \frac{\{\prod_{v_k \in V} \prod_{u_l \in U} P(m_{kl}|v_i, u_j)\}\ P(v_i, u_j)}{P(M)} \\ &= \frac{\left\{\prod_{v_k \in V} \prod_{u_l \in U} \frac{P(v_i, u_j|m_{kl})\ P(m_{kl})}{P(v_i, u_j)}\right\} P(v_i, u_j)}{\prod_{v_k \in V} \prod_{u_l \in U} P(m_{kl})} \\ &= \left\{\prod_{v_k \in V} \prod_{u_l \in U} \frac{P(v_i, u_j|m_{kl})}{P(v_i, u_j)}\right\} P(v_i, u_j) \\ &= \left[\frac{1}{P(v_i, u_j)}\right]^{(|V||U|)-1} \prod_{v_k \in V} \prod_{u_l \in U} P(v_i, u_j|m_{kl}) \end{aligned} \tag{4.11}$$

If vertex v_i is assumed to depend on vertex u_j only in the existence of the match M, then the unconditional likelihood that the two vertices are in correspondence is rewritten as $P(v_i, u_j) = P(v_i) \cdot P(u_j)$. This assumption means that the vertices of M are used to estimate the possibility of matching any two vertices from the two graphs. This is accomplished by utilizing the connectivity between v_i and $v_k \in M$ from one side, and between u_j and $u_l \in M$ on the other side. However, in the absence of M, the possibility of matching any two vertices does not depend on the structure of the

graphs. Furthermore, we assume a uniform distribution for the probabilities $P(V)$ and $P(U)$, $P(v_i) = \frac{1}{|V|}$ and $P(u_j) = \frac{1}{|U|}$, which means that all the vertices are equally important for the matching of any two graphs. As a result, $P(v_i, u_j|M)$ is simplified:

$$P(v_i, u_j|M) = K \prod_{v_k \in V} \prod_{u_l \in U} P(v_i, u_j|m_{kl}) \tag{4.12}$$
$$\text{where } K = [|V||U|]^{(|V||U|)-1}$$

Here, the density function $P(v_i, u_j|m_{kl})$ represents the conditional likelihood of $v_i \in V$ being in correspondence with $u_j \in U$, given the status of the assignment variable $m_{kl} \in \{0, 1\}$. In the previous equation, K is constant with respect to the mixture model. As a consequence, the maximization of the likelihood function is not affected by discarding K. By discarding K and using an exponential form, the complete-data likelihood can be rewritten as:

$$\begin{aligned} P(v_i, u_j|M) &= \exp\left[\ln\left(\prod_{v_k \in V} \prod_{u_l \in U} P(v_i, u_j|m_{kl}\right)\right] \\ &= \exp\left[\sum_{v_k \in V} \sum_{u_l \in U} \ln\left(P(v_i, u_j|m_{kl})\right)\right] \end{aligned} \tag{4.13}$$

We use an exponential form in the previous equation to simplify our formalism of the problem as will be explained later. By using Equation 4.13, the incomplete likelihood function in Equation 4.9 is rewritten as:

$$P(G|M) = \prod_{v_i \in V} \sum_{u_j \in U} \exp\left[\sum_{v_k \in V} \sum_{u_l \in U} \ln\left(P(v_i, u_j|m_{kl})\right)\right] \tag{4.14}$$

Based on the above equation, the conditional likelihood function $P(v_i, u_j|m_{kl})$ is the basic block for the probabilistic graph matching algorithm. Notice that till now we have not addressed the logarithm likelihood function, and the logarithm in the previous equation is the result of using the exponential form when defining $P(v_i, u_j|M)$.

4.2.2 A Graph Density Function

In this section, we propose our novel probability density function for the conditional likelihood $P(v_i, u_j|m_{kl})$. It is defined based on both structural and spatial properties in addition to the labels of the vertices and edges. The main idea is to compute

the conditional likelihood based on the compatibility between the vertices that are already selected in the match M and both the structure and geometry of the graphs. In other words, every two vertices $v_k \in V$ and $u_l \in U$ vote to support or reject the mapping of $v_i \in V$ to $u_j \in U$. We build our voting scheme with the following properties:

1. Given that $v_k \in N(v_i)$ and $u_l \in N(u_j)$ such that $N(.)$ defines the set of direct neighboring vertices for a given vertex, the density function is computed using a radial basis function (RBF) kernel of the conditional vertex distance (see Definition 3.4 in Section 3.3.1). This enables matching labeled graphs under geometric transformation, i.e., translation and rotation.

2. We use the shortest path similarity as a measure of how non-neighboring vertices contribute to the conditional likelihood function. By this, we combine a geometric feature, i.e., the lengths of the edges, with the structure of the graph, i.e., the shortest path. The benefit of using shortest paths is to utilize the overall structure of the graph for graph matching.

3. We propose a variation of our model to support scalability with respect to graph size. The main idea is to compute the density function based on a subspace of the mixture model, which is what we call the *k-active subgraph*.

Given the vertices $\{v_i, v_k\} \in V$ and $\{u_j, u_l\} \in U$, the density function $P(v_i, u_j|m_{kl})$ depends on two properties: 1) whether v_k and u_l are matched to each other, i.e., $m_{kl} = 1$, and 2) whether v_k and u_l are direct neighbors to v_i and u_j, respectively. Suppose $G_{ik} = 1$ denotes that v_i and v_k are direct neighbors, and $Q_{jl} = 1$ denotes that u_j is a direct neighbor to u_l. Our density function is then defined as:

$$P(v_i, u_j|m_{kl}) = \begin{cases} P_{ij,kl}\ F_{ik,jl} & \text{, if } m_{kl} = 1\ \wedge\ (G_{ik} = 1\ \wedge\ Q_{jl} = 1) \\ P_g\ F_{ik,jl} & \text{, if } m_{kl} = 1\ \wedge\ (G_{ik} = 0\ \vee\ Q_{jl} = 0) \\ P_g\ P_e & \text{, if } m_{kl} = 0 \end{cases} \tag{4.15}$$

where P_g is a constant representing the geometrical similarity, P_e is a constant representing the structural similarity, $P_{ij,kl}$ is the conditional vertex similarity, and $F_{ik,jl}$ is the shortest path similarity.

The intuition behind such a density function is that the geometrical similarity between two vertices is set to a minimum value of P_g. Then, the geometrical similarity is updated given the information of the neighborhoods of two vertices and the status of the match between the two graphs. If the neighbors of a vertex v_i are match to

the neighbors of another vertex u_j, then the neighborhoods of the two vertices are used to vote for the geometrical similarity between v_i and u_j. Other than that, we assume the similarity of v_1 and v_j minimum with a value of P_g. The same logic applies to the structural similarity P_e and the information of the shortest paths connecting different vertices. We build our density function $P(v_i, u_j|m_{kl})$ to satisfy the following observations and assumptions:

- The probability of matching any two vertices depends on their structural similarity, i.e., P_e and $F_{ik,jl}$, and their spatial similarity, i.e., P_g and $P_{ij,kl}$.
- Non-correspondent vertices support the matching of any two other vertices with a low constant probability. In other words, we only trust the vertices in the match M, i.e., $m_{kl} = 1$, in estimating the probability of matching the vertices of two graphs. This is indicated by the last term of our density function.
- Given two vertices v_k and u_l such that they are in correspondence, i.e., $m_{kl} = 1$, the probability of matching any other two vertices v_i and u_j is estimated in two different ways:
  1. Given that v_k is a direct neighbor to v_i and u_l is a direct neighbor to u_j, the conditional likelihood function is computed by utilizing both the conditional neighbor distance and the distance between the lengths of the edges $e_1 = (v_i, v_k)$ and $e_2 = (u_j, u_l)$. The latter distance is simply the distance between the shortest path between v_i and v_k and the shortest path between u_j and u_l.
  2. Given that $v_k \notin N(v_i)$ and $u_l \notin N(u_j)$, only the shortest path compatibility is used to estimate the probability of matching v_i with u_j. In addition to this, the spatial similarity between the two vertices is assumed minimum of a value of P_g.

Figure 4.2 illustrates our proposed density function between the two vertices v_1 and u_1 with the three possible cases in Equation 4.15. In the following, we detail how $P_{ij,kl}$ and $F_{ik,jl}$ are computed.

- **The conditional vertex similarity** $P_{ij,kl}$ between the two vertices $v_i \in V$ and $u_j \in U$, given that their direct neighbors $v_k \in N(v_i)$ and $u_l \in N(u_j)$ are matched to each other, is defined as:

$$P_{ij,kl} = \exp\left\{-\frac{d(v_i, u_j|v_k, u_l)^2}{2\sigma^2}\right\} \tag{4.16}$$

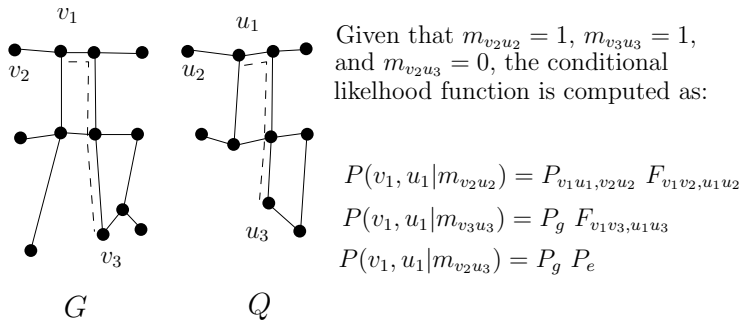


Figure 4.2: Computing the density value between v_1 and u_1. The dashed line is the shortest path.

where $d(v_i, u_j|v_k, u_l)$ is the conditional vertex distance based on Definition 3.4. The above equation can be considered as an RBF kernel. To compute the bandwidth σ, we first create a vector $\vec{x} \in \mathbb{R}^{|G|}$. An element x_i represents the minimum distance between vertex $v_i \in V$ and all the vertices of U. It is defined as $x_i = \min_{u_j \in U} d(v_i, u_j)$, where $d(v_i, u_j)$ is the vertex edit distance based on the second set of edit operations in Chapter 3. Then, σ is defined as:

$$\sigma = \mu(\vec{x}) + 3 * stdev(\vec{x}) \tag{4.17}$$

where μ and $stdev$ are the mean and standard deviation of the vector $\vec{x}$, respectively.

- **The shortest path similarity** $F_{ik,jl}$ between the shortest path $v_i, \ldots, v_k$ in graph G and the shortest path $u_j, \ldots, u_l$ in Q is defined as:

$$F_{ik,jl} = 1 - \left[\frac{|d_{v_iv_k} - d_{u_ju_l}|}{\max\{d_{v_iv_k}, d_{u_ju_l}\}} \right] \tag{4.18}$$

where $d_{v_iv_k}$ is the geodesic distance between v_i and v_k defined by the shortest path between them, in a similar way $d_{u_ju_l}$ is defined. To give an example on how $F_{ik,jl}$ is computed, we take the two graphs in Figure 4.2. Let us assume the length of the shortest path $d_{v_1v_3} \in G$ is 1.4, and assume the length of the shortest path $d_{u_1u_3} \in Q$ is 1.2, then $F_{ik,jl} = 1 - \left[\frac{|1.4-1.2|}{\max\{1.4,1.2\}} \right] = 1 - \frac{0.2}{1.4} = 0.857$. To speed up the computation of the density function, $F_{ik,jl}$ considers only the structure and the geometry of the graphs. However, the shortest path similarity can be defined to include labeling information as proposed by Tang *et al.* [115]. They use the optimal subsequence bijection algorithm (OSB) of Bai

and Latecki [11] to compute the similarity between two shortest paths. For this, first, a similarity matrix between the vertices of the two paths is created. The similarity of two vertices is computed based on the similarity of the labeling information of the neighborhoods of the two vertices. Then, the OSB algorithm is used to find a subsequence of the first path that is highly similar to another subsequence of the second path. Such a common subsequence is then used to estimate the similarity between the two shortest paths.

To simplify our analysis, Equation 4.15 is written by considering the conditional statements as exponential indicators. As a result, the density function is written as:

$$P(v_i, u_j|m_{kl}) = \left([P_{ij,kl}\ F_{ik,jl}]^{G_{ik}Q_{jl}m_{kl}}\right)\left([P_g\ F_{ik,jl}]^{(1-G_{ik}Q_{jl})m_{kl}}\right)\left([P_g\ P_e]^{1-m_{kl}}\right) \quad (4.19)$$

Substituting Equation 4.19 in the complete-data likelihood function (Equation 4.13) gives:

$$\begin{aligned} P(v_i, u_j|M) = \exp\Bigg[& \sum_{v_k \in V} \sum_{u_l \in U} m_{kl} G_{ik} Q_{jl}\ \ln(P_{ij,kl}\ F_{ik,jl}) \\ & + m_{kl}(1 - G_{ik}Q_{jl})\ \ln(P_g\ F_{ik,jl}) \\ & + (1 - m_{kl})\ \ln(P_g\ P_e)\Bigg] \end{aligned} \quad (4.20)$$

The previous equation is further simplified by distributing the logarithm functions and the multiplications as follows:

$$\begin{aligned} P(v_i, u_j|M) = \exp\Bigg[& \sum_{v_k \in V} \sum_{u_l \in U} m_{kl} G_{ik} Q_{jl}\ \ln(P_{ij,kl}) + \underline{m_{kl} G_{ik} Q_{jl}\ \ln(F_{ik,jl})} \\ & + \underline{m_{kl}\ \ln(P_g)} + m_{kl}\ \ln(F_{ik,jl}) - m_{kl} G_{ik} Q_{jl}\ \ln(P_g) - \underline{m_{kl} G_{ik} Q_{jl}\ \ln(F_{ik,jl})} \\ & + \ln(P_g) + \ln(P_e) - \underline{m_{kl}\ \ln(P_g)} - m_{kl}\ \ln(P_e)\Bigg] \end{aligned} \quad (4.21)$$

Notice that there are four terms that cancel each other, which are underlined in the above equation. Canceling these terms gives:

$$
\begin{aligned}
P(v_i, u_j|M) = \exp\Bigg[\sum_{v_k \in V} \sum_{u_l \in U} & m_{kl} G_{ik} Q_{jl} \ \ln(P_{ij,kl}) - m_{kl} G_{ik} Q_{jl} \ \ln(P_g) \\
& + m_{kl} \ \ln(F_{ik,jl}) - m_{kl} \ \ln(P_e) \\
& + \ln(P_g) + \ln(P_e) \Bigg]
\end{aligned} \tag{4.22}
$$

Merging the logarithm terms together gives:

$$
P(v_i, u_j|M) = \exp\left[\sum_{v_k \in V} \sum_{u_l \in U} m_{kl} G_{ik} Q_{jl} \ \ln\left(\frac{P_{ij,kl}}{P_g}\right) m_{kl} \ \ln\left(\frac{F_{ik,jl}}{P_e}\right) \ + \ \ln(P_g \ P_e) \right] \tag{4.23}
$$

From the above equation, it is noticed that the similarity constant P_g affects the way by which the conditional vertex similarity $P_{ij,kl}$ contributes to the density function. Moreover, there is a similar relation between the constant P_e and the shortest path similarity $F_{ik,jl}$. For example, the shortest path similarity contributes negatively to the density function when $F_{ik,jl} < P_e$. This also applies to the conditional vertex similarity $P_{ij,kl}$. Thus, choosing small values for P_e and P_g leads to a density function that is more tolerant to structural and spatial differences. We will show later that good values for P_e and P_g are 0.5 and 0.1, respectively.

The question is that why $P_e > P_g$, or why do we consider a higher threshold for the shorted path similarity? The answer to this question is divided into two points. First, the overall structure of two graphs, mostly, has many differences. This introduces many differences between paths that connect pairs of similar vertices. Second, even though the starting vertex of a path is matched to the starting vertex of another, several vertices that make the first path are not matched to the vertices that make the second one. These issues make the shortest path similarity less reliable than the conditional vertex similarity and requires a higher value for P_g.

Since our likelihood function has a hidden part, i.e., the vertices of graph Q, maximizing Equation 4.8 in closed form is intractable. In the following section, we draw our attention to the expectation maximization technique, which is used to estimate the parameters of a mixture model in the presence of hidden data.

4.3 Graph Matching Using the EM Technique

Expectation maximization (EM) is a technique proposed by Dempster *et al.* [36] to maximize a likelihood function in the case of hidden data. Since the graph matching

problem is formalized as maximum likelihood estimation, several authors use the EM technique and its variations to estimate the match between two graphs [40, 80, 109]. Following this, and after introducing our graph likelihood function, we use the EM technique for the matching of geometric graphs. Notice that using the EM technique is a standard approach. However, different graph matching algorithms use different density functions to estimate the similarity between different vertices, which we proposed in the previous section.

To simplify the computation of the MLE, normally, the log-likelihood of the observed data $\mathcal{L}(M)$ is maximized where $\mathcal{L}(M) = \ln P(G|M)$. Using $\mathcal{L}(M)$ in equations 4.8 and 4.13 gives:

$$\begin{aligned} M^* &= \arg\max_{M\in\mathcal{M}} \{\ln(P(G|M))\} \\ &= \arg\max_{M\in\mathcal{M}} \left\{ \sum_{v_i\in V} \ln \left\{ \sum_{u_j\in U} P(v_i, u_j|M) \right\} \right\} \end{aligned} \tag{4.24}$$

Dempster *et al.* [36] prove that maximizing the log-likelihood function is equivalent to maximizing a weighted sum of the complete log-likelihoods where the weights are the a posteriori of the hidden data. Such weighted sum can be iteratively maximized using the EM technique where each iteration has two steps. In the expectation step (E-step) and at an iteration $t+1$, the a posteriori probabilities of the hidden data $P(u_j|v_i, M^{(t)})$ are computed using the observed data and the previous estimation of the parameters $M^{(t)}$. In the maximization step (M-step), the parameters of the model are estimated using the observed data and the a posteriori probabilities of the hidden data $P(v_i, u_j|M^{(t+1)})$. As a consequence, the objective function $\Lambda(M^{(t+1)}|M^{(t)})$, which is maximized by the EM technique, can be formulated as:

$$\Lambda(M^{(t+1)}|M^{(t)}) = \sum_{v_i\in V} \sum_{u_j\in U} \underbrace{P(u_j|v_i, M^{(t)})}_{\text{E-step}} \underbrace{\ln P(v_i, u_j|M^{(t+1)})}_{\text{M-step}} \tag{4.25}$$

In the following two sections, we detail how the a posteriori of the hidden data is computed at the E-step, and the maximization of the objective function in the M-step.

4.3.1 Expectation

In the E-step, the a posteriori $P(u_j|v_i, M^{(t)})$ is computed using the observed data and an estimation of the parameters of the mixture model. For simplification, we will

use $R_{ij}^{(t)} = P(u_j|v_i, M^{(t)})$, such that $R^{(t)}$ is a matrix that stores all the a posteriori probabilities at iteration $t+1$. Using Bayes' rule one obtains:

$$R_{ij}^{(t)} = P(u_j|v_i, M^{(t)}) = \frac{P(v_i, u_j|M^{(t)})P(M^{(t)})}{\sum_{u_y \in U} P(v_i, u_y|M^{(t)})P(M^{(t)})} \tag{4.26}$$

Substituting Equation 4.23 in the previous equation gives:

$$R_{ij}^{(t)} = \frac{\exp\left[\sum_{v_k \in V}\sum_{u_l \in U} m_{kl}^{(t)} \left\{G_{ik}Q_{jl} \ \ln\left(\frac{P_{ij,kl}}{P_g}\right) \ + \ \ln\left(\frac{F_{ik,jl}}{P_e}\right)\right\}\right]}{\sum_{v_y \in U} \exp\left[\sum_{v_k \in V}\sum_{u_l \in U} m_{kl}^{(t)} \left\{G_{ik}Q_{yl} \ \ln\left(\frac{P_{iy,kl}}{P_g}\right) \ + \ \ln\left(\frac{F_{ik,yl}}{P_e}\right)\right\}\right]} \tag{4.27}$$

Notice that when computing $R_{ij}^{(t)}$, both $\ln(P_g \ P_e)$ and $P(M^{(t)})$ from Equation 4.23 are canceled out.

4.3.2 Maximization

In the maximization step of the EM technique, the mixture model $M^{(t+1)}$ that maximizes the objective function $\Lambda(M^{(t+1)}|M^{(t)})$ is selected. Such a mixture model is then used in the next iteration by the E-step. The maximization step is formalized as follows:

$$\begin{aligned} M^{(t+1)} &= \arg\max_{\hat{M} \in \mathcal{M}} \Lambda(\hat{M}|M^{(t)}) \\ &= \arg\max_{\hat{M} \in \mathcal{M}} \left\{ \sum_{v_i \in V}\sum_{u_j \in U} R_{ij}^{(t)} \ \ln P(v_i, u_j|\hat{M}) \right\} \\ &= \arg\max_{\hat{M} \in \mathcal{M}} \left\{ \sum_{v_i \in V}\sum_{u_j \in U} R_{ij}^{(t)} \sum_{v_k \in V}\sum_{u_l \in U} \left[\hat{m}_{kl} G_{ik} Q_{jl} \ \ln\left(\frac{P_{ij,kl}}{P_g}\right) \right.\right. \\ &\left.\left. + \ \hat{m}_{kl} \ln\left(\frac{F_{ik,jl}}{P_e}\right) \ + \ \ln(P_g \ P_e) \right] \right\} \end{aligned} \tag{4.28}$$

Since $\ln(P_g \ P_e)$ is constant with respect to the parameters of the mixture model, i.e., the assignment variables $\hat{m}_{kl}$, it can be discarded. Re-arranging the summation, the previous equation is rewritten as:

$$M^{(t+1)} = \arg\max_{\hat{M} \in \mathcal{M}} \left\{ \sum_{v_k \in V}\sum_{u_l \in U} \hat{m}_{kl} S_{kl} \right\} \tag{4.29}$$

$$\text{where } S_{kl} = \sum_{v_i \in V} \sum_{u_j \in U} R_{ij}^{(t)} \left[G_{ik} Q_{jl} \ \ln\left(\frac{P_{ij,kl}}{P_g}\right) \ + \ \ln\left(\frac{F_{ik,jl}}{P_e}\right)\right]$$

In the previous equation, the matrix S represents the vertex similarity and the structural compatibility score between vertices from graph G and Q. The higher the score between vertices $v_k \in V$ and $u_l \in U$, the more likely they are in correspondence. It is worth mentioning that the value of a similarity score between two vertices belongs to the $\mathbb{R}$ space.

Several algorithms can be used to maximize Equation 4.29. Luo and Hancock [80] use the extremum principal based on singular value decomposition of the score matrix S [110]. Sanromá *et al.* [109] use the Softassign algorithm. In this paper, we propose to use Hungarian algorithm [85]. This algorithm is well-known in solving the assignment problem and runs in $O(m^3)$ where $m = \max\{|G|, |Q|\}$. We use the Hungarian algorithm since it scales better than the other approaches with respect to the graph size.

4.3.3 Generalized EM

In the previous sections, we introduced a probabilistic model for graph matching. However, such a model has some problems. First, it has a scalability problem in computing the a posteriori $P(u_j|v_i, M^{(t)})$, see Equation 4.27. This is because the sum in the denominator goes to infinity as graph size increases, or to be more specific, when the number of assignment variables $m_{ij} = 1$ increases. Such a problem actually exists in the original model of Sanromá *et al.* [109] and prevents it from having good scalability with respect to graph size. Second, the model is sensitive to outlier vertices. An outlier vertex is a vertex that does not have a corresponding vertex in the other graph. As a result, the previous model solves (sub)graph matching but fails in finding common subgraphs.

In this section, we focus on an extension to our model that overcomes these two problems. To solve the scalability problem, we propose to compute the a posteriori $P(u_j|v_i, M^{(t)})$ and the scoring matrix S based on a subspace of the mixture model [40]. For this, we introduce the concept of *k-active subgraph*.

Definition 4.1. ***(k-active subgraph)*** *Given a graph $G = (V, E)$, the k-active subgraph for a vertex $v_i \in V$ is the subgraph $G_{v_i} = (V_{v_i}, E_{v_i})$ such that V_{v_i} is the set of the k-nearest neighbors to v_i. The distance between v_i and $v_k \in V_{v_i}$ is defined as the geodesic distance between them.*

Notice that the k-active subgraph for a vertex v is not defined based on the vertices that are reachable within k hubs from v. Instead, it is defined based on the k nearest neighbors so that all the k-active subgraphs for all vertices have the same cardinality. Our experiments will show later that a good value for k is 15. In the case of graphs with a number of vertices fewer than 15, the k-active-subgraph is simply the entire graph. Why does a value higher than 15 give worse results? Actually, this is related to the lengths of the shortest paths that are used to compute the shortest path similarity $F_{ik,jl}$. With values higher than 15, the density function utilizes more and more information about the overall graph structure. However, we discussed that longer paths are highly unreliable in estimating the density value. Given the two k-active subgraphs V_{v_i} and U_{u_j} for two vertices $v_i \in G$ and $u_j \in Q$, respectively, we rewrite the definition of both the a posteriori $P(u_j|v_i, M^{(t)})$ and the scoring matrix S as follows:

- **E-step**

$$R_{ij}^{\prime(t)} = \frac{\exp\left[\sum_{v_k \in V_{v_i}} \sum_{u_l \in U_{u_j}} m_{kl}^{(t)} \left\{ G_{ik} Q_{jl} \ln\left(\frac{P_{ij,kl}}{P_g}\right) + \ln\left(\frac{F_{ik,jl}}{P_e}\right)\right\}\right]}{\sum_{v_y \in U} \exp\left[\sum_{v_k \in V_{v_i}} \sum_{u_l \in U_{u_y}} m_{kl}^{(t)} \left\{ G_{ik} Q_{yl} \ln\left(\frac{P_{iy,kl}}{P_g}\right) + \ln\left(\frac{F_{ik,yl}}{P_e}\right)\right\}\right]} \tag{4.30}$$

- **M-step**

$$M^{\prime(t+1)} = \arg\max_{\hat{M} \in \mathcal{M}} \left\{ \sum_{v_k \in V} \sum_{u_l \in U} \hat{m}_{kl} S_{kl}^{\prime} \right\} \tag{4.31}$$

$$\text{where } S_{kl}^{\prime} = \sum_{v_i \in V_{v_k}} \sum_{u_j \in U_{u_l}} R_{ij}^{(t)} \left[G_{ik} Q_{jl} \ln\left(\frac{P_{ij,kl}}{P_g}\right) + \ln\left(\frac{F_{ik,jl}}{P_e}\right)\right]$$

It is true that the parameters of the mixture model $M^{(t+1)}$ in Equation 4.29 are not identical to $M^{\prime(t+1)}$ in Equation 4.31. However, a higher score at $S_{kl}^{\prime}$ still indicates a better match [40].

Next, we focus on updating the model so that it can find common subgraphs. The main idea is that the vertices from the common subgraph between G and Q have higher values in the score matrix $S^{\prime}$. We update the algorithm used to maximize Equation 4.31 so that the vertices from the non-common subgraph are excluded before applying Munkres' algorithm. We summarize this by the following two steps:

1. **Candidate selection.** From the score matrix $S^{\prime}$, only the top-$\hat{k}$ similar pairs of vertices are selected, which are called the *candidate set*. All other entries from

S' are assigned a value of zero. By this, we gain two benefits: 1) pruning non-similar vertices from the score matrix S', and 2) speeding up the maximization step. Experimentally, a good value for $\hat{k}$ is $10 \times \max\{|G|, |Q|\}$. Notice that $\hat{k}$ does not refer to the size of an active subgraph. In other words, k refers to the size of an active subgraph and $\hat{k}$ denotes the number of similar vertices to a given vertex that are selected for the candidate set.

2. **Solving the assignment problem.** Munkres' algorithm [85] is then used to select the best match from the candidate set.

Since Munkres' algorithm solves the assignment problem in cubic time, pruning non-similar pairs of vertices makes the score matrix S' sparse. As a result, the size of the assignment problem will decrease. In addition to this, solving the assignment problem with a sparse matrix is much faster than with a dense one. To estimate the initial mixture model $M'^{(0)}$, the score matrix S' is defined based on an RBF kernel of the vertex distance function (Definition 3.5), which is defined based on the vertex edit distance function using the second set of edit operations.

The EM technique with these modifications can be considered a generalized expectation maximization (GEM) algorithm [36]. The maximization step that is used by the GEM selects $M'^{(t+1)}$ such that

$$\Lambda(M'^{(t+1)}|M'^{(t)}) \quad \geq \quad \Lambda(M'^{(t)}|M'^{(t)}). \tag{4.32}$$

The above equation means that the M-step selects a mixture model that improves the objective function but does not maximize it. To guarantee that our algorithm satisfies Equation 4.32, all pairs of vertices such that $m_{ij}^{(t)} = 1$ are added to the candidate set. In other words, the match selected at iteration t is added to the candidate set at iteration $t + 1$. In this case, Munkres' algorithm finds the optimal solution to the assignment problem. As a consequence, the score of the new match $M'^{(t+1)}$ will be for sure higher than or equal to the score of $M'^{(t)}$. Let $score^{t+1} = \Lambda(M'^{(t+1)}|M'^{(t)})$, normally, the GEM algorithm converges when $score^{t+1} - score^{t} < \alpha$. However, for our framework, we normalize the score used in the convergence test by the size of the match $M'^{(t+1)}$ as follows:

$$score^{t+1} = \frac{score^{t+1}}{\sum\limits_{m_{ij} \in M'^{(t+1)}} m_{ij}} \tag{4.33}$$

The intuition behind this is to force the GEM algorithm to converge when the size of the match is not improved. We show later that a good value for α is 0.5.

4.3.4 Complexity Analysis

In the following, we discuss the complexities of the main components of our proposed algorithm, which is summarized in Algorithm 4.1. Let $m = |G|$, $n = |Q|$, and $m \leq n$, the worst case complexity of the match initialization step is $O(n^3)$. This is a consequence of using Munkres' algorithm in line 4. In practice the complexity is less than that, because only the top-$\hat{k}$ similar pairs of vertices are selected before applying Munkres' algorithm.

The complexity of the expectation step, line 9, is $O(mncd^2)$, where $O(d^2)$ is the complexity of computing the conditional vertex similarity $P_{ij,kl}$ between two vertices such that d is the average degree of a vertex. Given two k-active subgraphs of two vertices, c represents the number of vertices from the first k-active subgraph that are matched to vertices from the other k-active subgraph. This makes $0 \leq c \leq k$, $k = 15$.

The complexity of computing the scoring matrix S', line 11, is $O(mnk^2d^2)$, where d is the average degree of a vertex and $k = 15$ is the size of a k-active subgraph. In practice, the complexity of computing S' can be reduced. This is done by only computing the score for the entries where the a posteriori value is high. In other words, since the a posteriori is considered as a weighting factor for the complete-likelihoods, one can skip computing the complete-likelihood for vertices where the a posteriori value is very low. In our implementation, we used a threshold of 0.1. This means that if the a posteriori of two vertices is less that 0.1, then their complete-likelihood value is set to 0. This is justified because, in all cases, low score entries will be removed from S' before applying Munkres' algorithm, line 13. The other component of the maximization step is solving the assignment problem, line 14, which has again a complexity of $O(n^3)$.

Where is the bottleneck of our algorithm in terms of complexity? The answer to this question depends on the two graphs to be compared. If the size of the match, i.e., $\sum_{m_{ij} \in M} m_{ij}$, reaches $\min\{|G|, |Q|\}$, then the bottleneck will be Munkres' algorithm. On the other hand, if the size of the match is much smaller than the smallest graph, then the highest complexity will be the computation of the scoring matrix S'.

Algorithm 4.1: Probabilistic graph matching

Input: Two geometric graphs $G = (V, E)$ and $Q = (U, T)$

Output: A match M that represents the correspondences between V and U

/* Compute the initial match */

foreach $i \in |V|$, $j \in |U|$ **do**

  /* RBF kernel of the vertex distance function */

  $S_{ij} \leftarrow \text{VertexSimilarity}(v_i, u_j)$

$S \leftarrow \text{Top-}\hat{k}(S)$ /* $\hat{k} \leftarrow 10 \times \max\{|V|, |U|\}$ */

/* solve the assignment problem */

$\{M\} \leftarrow Munkres(S)$

/* Start the GEM algorithm */

$oldscore \leftarrow 0$

$newscore \leftarrow 0$

$counter \leftarrow 0$

/* The GEM algorithm converges after a certain number of iterations */

while $counter < max\_iterations$ **do**

  /* Expectation step, Equation 4.30 */

  foreach $i \in |V|$, $j \in |U|$ **do**

    $R'_{ij} \leftarrow P(u_j | v_i, M)$

  /* Maximization step, Equation 4.31 */

  foreach $k \in |V|$, $l \in |U|$ **do**

    $$S'_{kl} \leftarrow \sum_{v_i \in V_{v_k}} \sum_{u_j \in U_{u_l}} R'_{ij} \left[G_{ik} Q_{jl} \ \ln\left(\frac{P_{ij,kl}}{P_g}\right) \ + \ \ln\left(\frac{F_{ik,jl}}{P_e}\right) \right]$$

  /* Solve the assignment problem */

  $S' \leftarrow \text{Top-}\hat{k}(S')$ /* $\hat{k} \leftarrow 10 \times \max\{|V|, |U|\}$ */

  $\{M, newscore\} \leftarrow Munkres(S')$

  $$newscore \leftarrow \frac{newscore}{\sum_{m_{ij} \in M} m_{ij}}$$

  /* The GEM algorithm converges when the change in the score is statistically insignificant, $\alpha \leftarrow 0.5$ */

  if $newscore - oldscore < \alpha$ **then**

    /* break from the while loop */

    stop

  $oldscore \leftarrow newscore$

  $counter \leftarrow counter + 1$

return M

4.4 Experimental Evaluation

The focus of this section is to empirically evaluate our proposed graph matching algorithm. Several criteria are considered by our evaluations. The first criterion is the matching quality. Algorithms with high matching quality are more resistant to changes in graph structure and spatial properties. For our experiments, the quality of the match is measured by the matching accuracy. For two graphs G and Q, the matching accuracy is estimated by the agreement between the match M that is computed by a graph matching algorithm, and the ground truth match $\hat{M}$, which formalized in Equation 3.10, page 64.

The second criterion that we study is scalability with respect to graph size. It is measured by the runtime required to match different graphs, in addition to the memory consumption. The third one is the recognition rate. This is evaluated by embedding the graph matching algorithm in a classification task. The higher the classification accuracy the better the graph matching algorithm.

In our experiments, we used geometric graphs that are extracted from different application domains. The first dataset is the CMU hotel/house datasets that consist of 212 geometric graphs [2]. The size of each graph is 30 vertices. The second dataset represents graphs extracted from road networks. Three road networks are used: California, City of Oldenburg, and North America, with sizes 1365, 3494, and 7517 vertices, respectively [4]. The third dataset is the COIL-100 dataset [87]. It consists of 3900 geometric graphs with an average graph size of 21 vertices. More details about the datasets will be outlined in the following sections.

We compare our graph matching algorithm **MixModel**, Section 4.3.3, to four other related graph matching algorithms. In the following we summarize them, and the reader may refer to Chapter 2 for more details.

1. **EMSoft** is the original probabilistic graph matching algorithm proposed by Sanromá *et al.* [109]. To compute the conditional likelihood function, they propose a multivariate Gaussian distribution based on an affine-invariant geometrical relationship between the vertices of different graphs.

2. The **GMS** algorithm [26] follows a feature extraction and embedding approach. Initially, a set of vertices from each graph is selected as *landmarks*. Then, every vertex from a graph is represented by a feature vector containing the distances to the landmarks. After that, the *Earth Mover's distance* is used to match two geometric graphs [106].

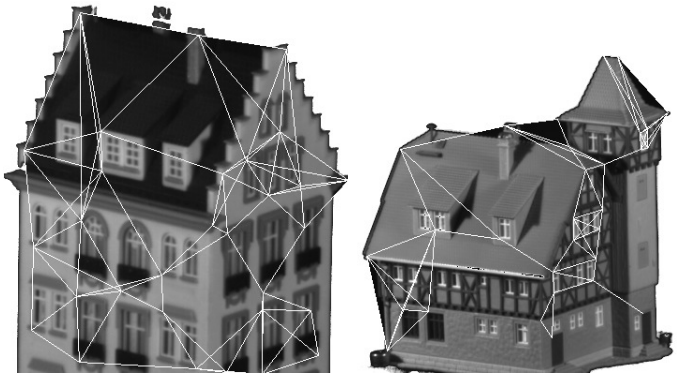


Figure 4.3: Sample images from the CMU house/hotel datasets and their corresponding geometric graphs represented by the white lines [2].

3. The **RRWM** algorithm [27] is a random walk-based graph matching algorithms. An associated graph is created from the two graphs to be compared. Then, the stationary distribution for a random walk over the associated graph gives a ranking for the possible correspondences between the vertices of two graphs.

4. **CSv2** is a graph edit distance algorithm that utilizes only the neighborhood of the vertices for graph matching. Initially, a vertex-to-vertex distance matrix is computed using vertex edit distance and the second set of edit operations. Then, Munkres' algorithm is used to select the best match. This algorithm is the same one that is used in our evaluation in Section 3.4.

All experiments were carried out on an Ubuntu 12.04 platform with an Intel Core i5 CPU with 8GB RAM, and all algorithms are written in C++. For matrix operations, we used the linear algebra library *Armadillo* [107]. To implement the related algorithms and for **EMSoft**, we used some matlab code from the author[1]. We also transfer the matlab code of **RRWM** into C++[2]. For **GMS**, we used the publicly available C code of the Earth Mover's distance[3].

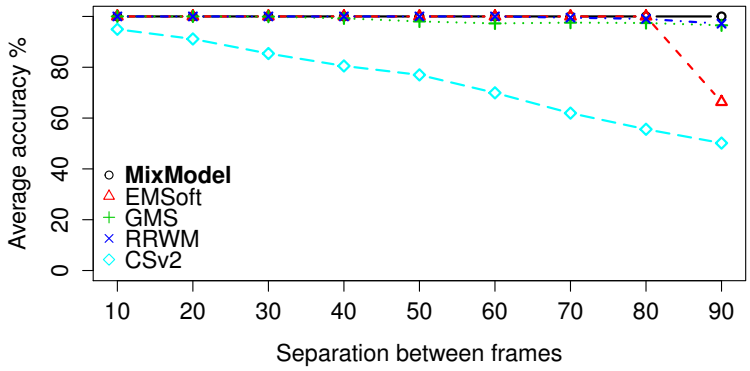
4.4.1 Matching Quality

We start our experiments with the CMU house and hotel datasets [2], which are used by almost all related graph matching algorithms. They demonstrate how in general

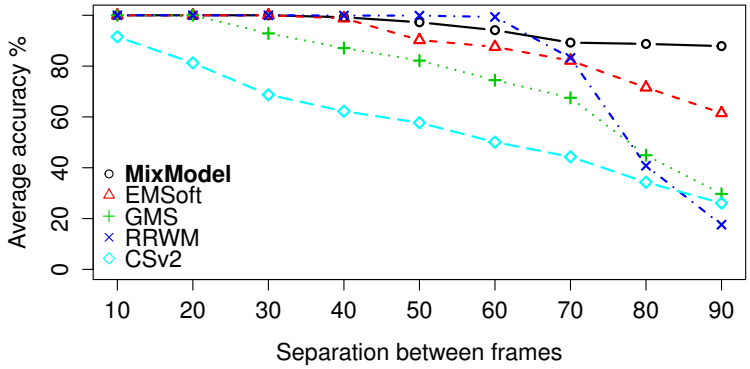
[1]http://www.mathworks.com/matlabcentral/fileexchange/35179-smooth-point-set-registration-using-neighboring-constraints, accessed 02.11.2013

[2]http://cv.snu.ac.kr/research/~RRWM/, accessed 02.11.2013

[3]http://robotics.stanford.edu/~rubner/emd/default.htm, accessed 02.11.2013



(a) House dataset



(b) Hotel dataset

Figure 4.4: The average matching accuracies for the CMU dataset and for different graph matching algorithms.

a graph matching algorithm can be used to track an object using images taken at different times or different viewing angles. They consist of images of a toy house and hotel, subjected to rotation in 3D. Figure 4.3 shows example images and their corresponding geometric graphs.

The house dataset consists of 111 images (snapshots taken during a rotation). The hotel dataset has 101 images. We match all images spaced at 10, 20, 30, 40, 50, 60, 70, 80, and 90 in the rotation sequence, and compute the average matching accuracy. We compare our algorithm **MixModel** with the related four graph matching algorithms: **EMSoft**, **GMS**, **RRWM**, and **CSv2**.

Figure 4.4(a) shows the matching accuracy for the CMU house dataset as the separation between snapshots increases in the rotation sequence. The more spaced

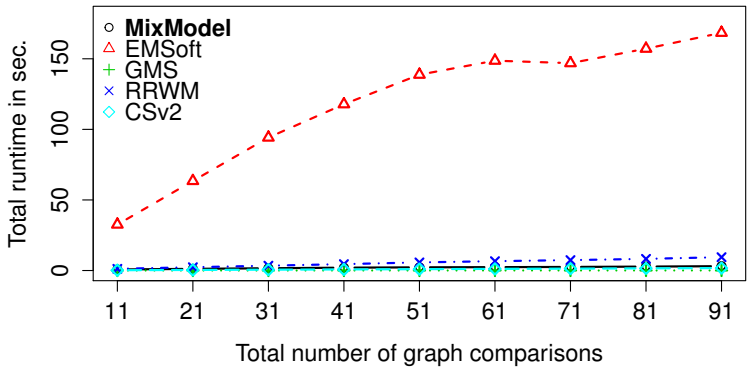


Figure 4.5: Runtime for the CMU hotel dataset.

in time the two graphs (snapshots) are, the more dissimilar they are. As a result, one can see that nearly for all algorithms, the matching accuracy decreases as the time between the snapshots increases. The matching accuracies for our algorithm **MixModel**, **GMS**, and **RRWM** are higher than the **CSv2** algorithm. This is because the overall graph connectivity is not considered by **CSv2**.

The matching accuracy for the CMU hotel dataset is shown in Figure 4.4(b). All the algorithms have less matching accuracy compared to their results for the house dataset (Figure 4.4(a)). This is because the spatial and structural differences between different snapshots of the hotel dataset are more than the differences between the snapshots of the house dataset. Our algorithm **MixModel** has the highest matching accuracy. **EMSoft** has higher matching accuracy than both **GMS** and **RRWM**, especially for graphs far apart in the snapshot sequence. This means that **EMSoft** is more tolerant to changes in the locations of the vertices and the structure of the graphs.

We use the CMU hotel dataset also to measure the scalability of the graph matching algorithms. Figure 4.5 shows the scalability measured by the runtime in seconds for all algorithms as the number of graph comparisons increases. Although **EMSoft** has good matching quality, it has a very high runtime even for such a small dataset. The reason for this is that at each iteration, **EMSoft** searches for the best affinity transformation parameters for each pair of vertices from the two graphs to be compared. This result is similar to the runtime analysis reported in [109]. Since **EMSoft** has very poor scalability with respect to the number of graphs and the graph size, we exclude it from the other experiments.

Figure 4.6: Sample images from the COIL-100 dataset [99].

4.4.2 Graph Similarity and Classification

In this experiment, the recognition rate for a graph matching algorithm is evaluated. Empirically, we test how good a graph matching algorithm is in finding graphs similar to a query graph. For all three algorithms **MixModel**, **CSv2**, and **RRWM**, the similarity between two graphs is quantified as the similarity of their common subgraph in addition to the similarity of their sizes. We ran the GEM algorithm for only one iteration for **MixModel**. For **GMS**, the similarity is quantified by the flow calculated by the Earth Mover's distance algorithm as proposed in [26]. To measure the recognition rate we conduct a graph classification experiment. The higher the classification accuracy, the higher recognition rate the algorithm has. For such an experiment, we use the COIL-100 dataset [87, 99], which consists of images of 100 different objects taken at different degrees. Figure 4.6 shows sample images from this dataset. For classification, we select 2900 graphs for training, 29 graphs for each object. For testing, we select 1000 graphs, 10 graphs for each object. We compute the similarity between each graph from the test dataset and all graphs from the training

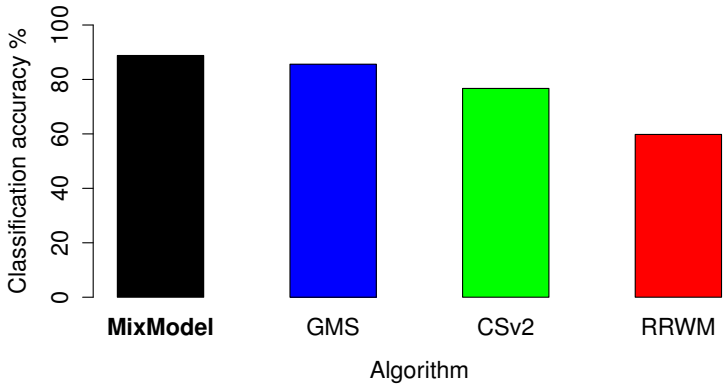
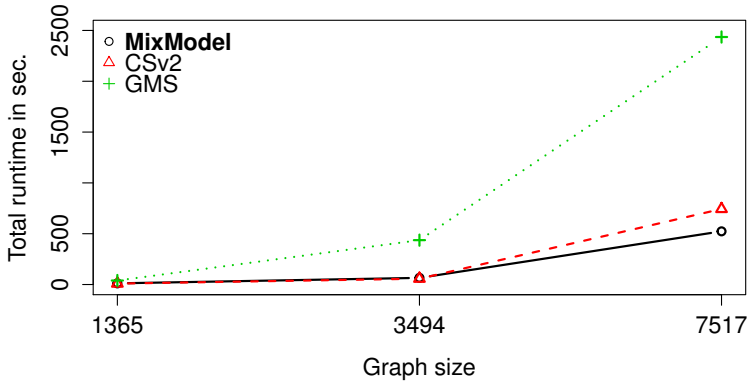


Figure 4.7: Classification accuracy for the COIL-100 dataset.

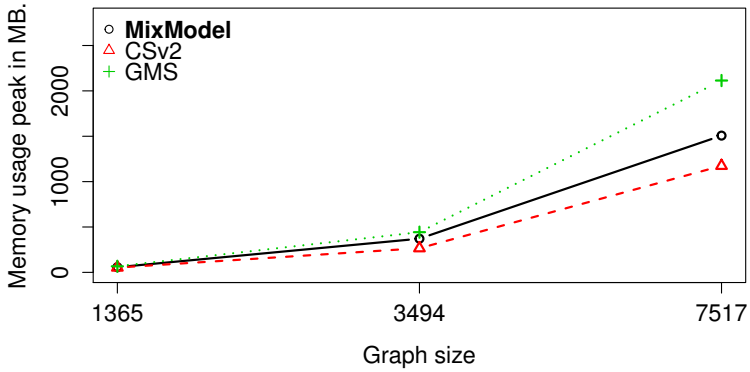
dataset. Then, a 1-NN classifier is used to find the nearest neighbor graph to the query graph. Figure 4.7 shows the classification accuracy for all four algorithms. Our algorithm **MixModel** and **GMS** has the highest classification accuracy followed by **CSv2** and then **RRWM**.

4.4.3 Scalability Study

In this experiment, we study the runtime and memory consumption for the algorithms **MixModel**, **GMS**, and **CSv2**. We use these algorithms to answer common subgraph queries. Three graphs are extracted from three road networks: California, the City of Oldenburg, and North America [4]. From each graph, we created another distorted graph [3]. Each of the graph matching algorithms is used to estimate the common subgraph between a graph and its distorted version. To create the distorted version, initially, the graph is clustered into groups of vertices. Distortion is applied to some of the clusters to make them non-similar. The rest of the clusters are left untouched without any distortion to create a common subgraph. In addition to this, rotation and translation are done randomly to some of the clusters. Such an experiment demonstrates the task of finding the differences in a road network over time or finding similar areas in two different road networks. From this experiment, we exclude the **RRWM** algorithm. This algorithm uses the Kronecker product to store the edge similarity between two graphs, i.e., it uses a matrix of size $|G||Q| \times |G||Q|$ to match the two graphs G and Q. For the California road network, which is the smallest graph in this experiment, the similarity matrix has 10^{12} entries, which requires a huge



(a) Runtime



(b) Memory Consumption

Figure 4.8: Scalability study in terms of runtime and memory consumption for different graph matching algorithms as graph size increases.

amount of RAM storage. This makes such an algorithm scale poorly with respect to memory consumption.

We report the runtime for all three algorithms in Figure 4.8(a). Our algorithm gives the best result. It scales better than **CSv2**, because we are selecting the top-$\hat{k}$ similar pairs of vertices before solving the assignment problem. **GMS** has the highest runtime. This is a consequence of using the Earth Mover's distance algorithm.

Figure 4.8(b) shows the memory consumption for the three algorithms, which is measured by the memory peak reported for each of them at different graphs. The **CSv2** matching algorithm has the lowest memory consumption. Our algorithm **MixModel** comes in the second place. It consumes more memory than **CSv2**, because it uses two different matrices in the E- and M-steps, i.e., $R^{'}$ and $S^{'}$, each of size $|G| \times |Q|$.

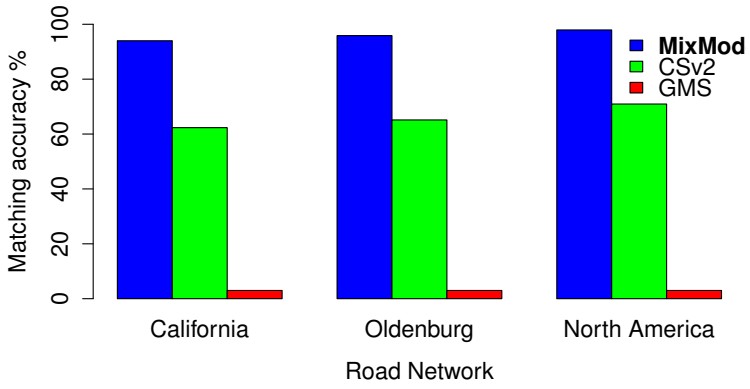


Figure 4.9: Matching quality for common subgraph matching using different road networks.

In addition to this, it needs another matrix of size $|G| \times |Q|$ in the candidate selection step. However, **CSv2** requires only one matrix of size $|G| \times |Q|$ to match the two graphs G and Q. Finally, **GMS** has the highest memory consumption. Even though our algorithm does not score best with respect to memory consumption, it scales in a way similar to the related graph matching algorithms, i.e., each algorithm has a memory complexity of $O(|G||Q|)$. On the other hand, our algorithm scores best in terms of the matching accuracy, as will be shown in the following.

We also report the matching accuracy for this test in Figure 4.9. Our algorithm has the highest matching accuracy with an average of 97%. On the other hand, the **GMS** algorithm has nearly zero matching accuracy for all three road networks. This is because the landmark selection method used by the algorithm is not suitable to match graphs that differ in the number of vertices. Finally, the **CSv2** algorithm has reasonable results even though the overall graph structure is not used.

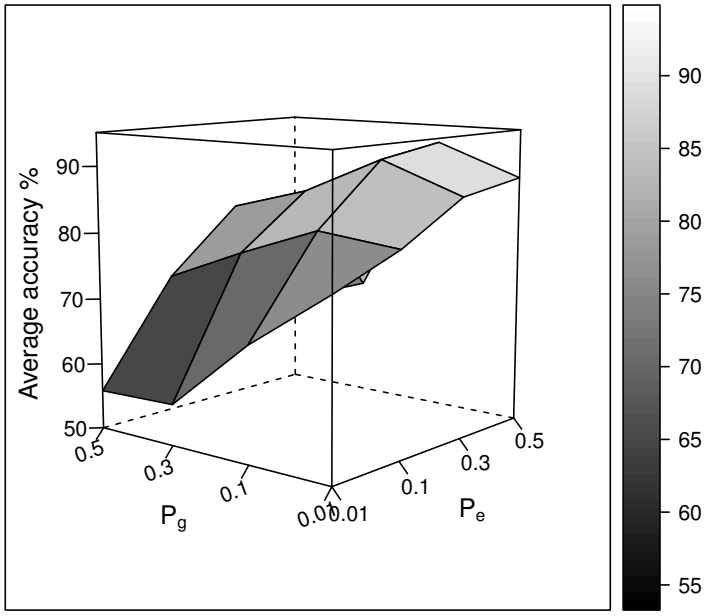
Next, we summarize our experiments. Both the **EMSoft** and **RRWM** algorithms give good matching accuracy for small geometric graphs. Both algorithms are able to match graphs that differ in the number of vertices, graph structure, and spatial properties. However, they scale very poorly in terms of memory consumption and runtime. The **GMS** algorithm is suitable for graphs that have the same number of vertices. In addition to that, such an algorithm is good in finding the similarity between graphs but not to determine common subgraphs. Even though **CSv2** does not consider the overall graph connectivity, it gives reasonable results. We conclude, that a good vertex similarity measure is sufficient to approximate the similarity between

different graphs. Finally, our algorithm scored the highest matching accuracy and the least runtime for all tests. It scales better than **EMSoft**, **RRWM**, and **GMS** in terms of memory consumption. **CSv2** outperforms our algorithm in terms of memory consumption. This is because our algorithm considers and stores more information about the connectivity of the graphs, which leads to a higher matching accuracy and a higher memory consumption.

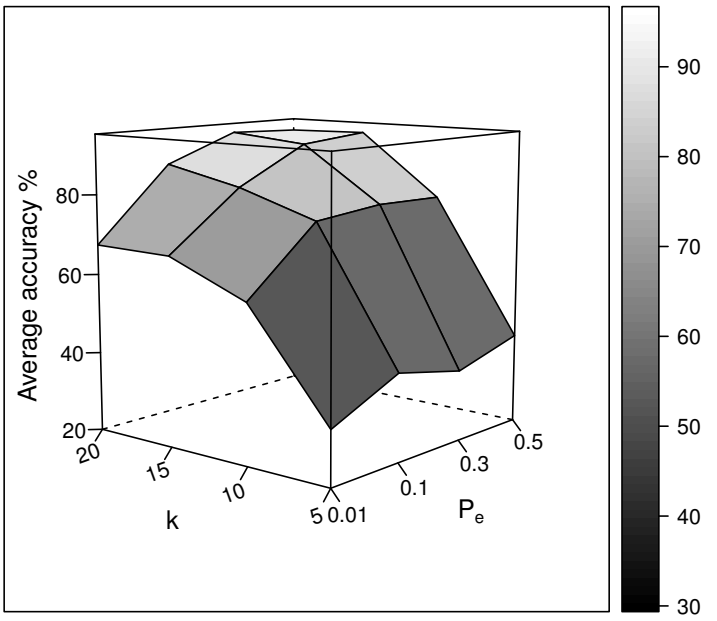
4.4.4 Parameters Analysis

In this section, we discuss how the parameters of our framework are chosen. For this, we use a dataset of 5 subgraphs extracted from the California road network. The average size of a subgraph is 100 vertices. Spatial and structural distortions are applied randomly to the subgraphs. Then, our framework is used to match each subgraph against the California road network. Figure 4.10(a) shows the relationship between the constants P_e, which is the minimum structural similarity between two vertices, and P_g, which is the minimum spatial similarity between two vertices, on one side and the average matching accuracy on the other side. One can see that the highest matching accuracy occurs when $0.3 \leq P_e \leq 0.5$ and $P_g = 0.1$. We fix P_g to a value of 0.1 and study the effect of the size k of an active subgraph. Figure 4.10(b) shows the average matching accuracy for different values of k and P_e. The highest matching accuracy occurs when P_e is set to 0.5 and k to 15. For higher values of k, the matching accuracy starts to decrease. This supports our argument about the relationship between the length of a shortest path and its reliability for graph matching, namely, the longer the path, the less reliable it is.

Next, we study the convergence speed of our algorithm in addition to the convergence threshold α. For this test, we use the graphs of the CMU hotel dataset that are spaced 70, 80, and 90 in the rotation sequence. In total there are 104 graph comparisons. Figure 4.11 shows the matching accuracy at each iteration. Iteration 0 gives the matching accuracy before we start the GEM algorithm, i.e., the accuracy of the match that is used to initialize GEM. Furthermore, the figure shows the change of the score at each iteration, i.e., $\Delta = score^{(t+1)} - score^{(t)}$ (Equation 4.33). Note that Δ can be only computed starting from the second iteration. One can see that on average our approach converges after 6 iterations. The change in the score is statistically insignificant starting from the 3^{rd} iteration. The change in the average matching accuracy is statistically insignificant starting from the 2^{nd} iteration. From the values of Δ at iteration 2 and 3, we conclude that a good value for α is 0.5. In



(a) The effect of P_e and P_g



(b) The effect of P_e and the size of a k-active subgraph.

Figure 4.10: The effect of different parameters of our algorithm, including P_e, P_g, and the size of the active subgraph, on the matching accuracy.

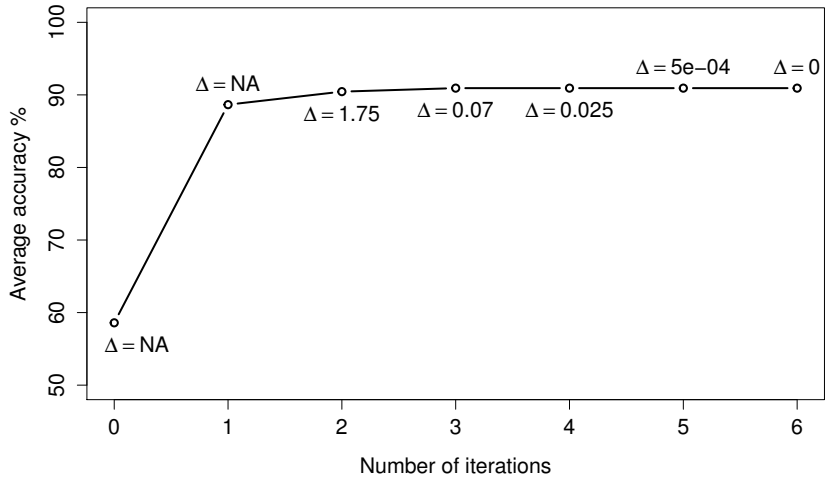


Figure 4.11: Average matching accuracy at different iterations of our approach, $\Delta = score^{t+1} - score^{t}$.

addition to this, the figure shows that one iteration of our approach is enough to give good results, i.e., only a marginal increase in the matching accuracy is achieved after the first iteration. We show this result visually in Figure 4.12 for two graphs extracted from the CMU hotel dataset. It shows the match computed before applying the GEM algorithm, i.e., at iteration 0. It also shows the match that is computed by the first iteration of the GEM algorithm. Notice that only one iteration is sufficient to increase the matching accuracy from 43% to 70%.

4.5 Summary and Discussion

In this chapter, we proposed an efficient graph matching framework for geometric graphs in 2D space. The main problem we solved is that how to estimate the match between two graphs based on both 1) the similarity of the vertices utilizing their neighborhoods, Chapter 3, and 2) the similarity based on the overall graph structure. Taking these two issues into consideration, we proposed a probabilistic graph matching algorithm. The main idea is to use maximum likelihood estimation to find the best match between two graphs. We define a mixture model of the set of possible correspondences between the vertices of two graphs. Then, the solution of the graph matching problem between two graphs is the mixture model that maximizes the likelihood function. For this, we propose a novel density function that considers

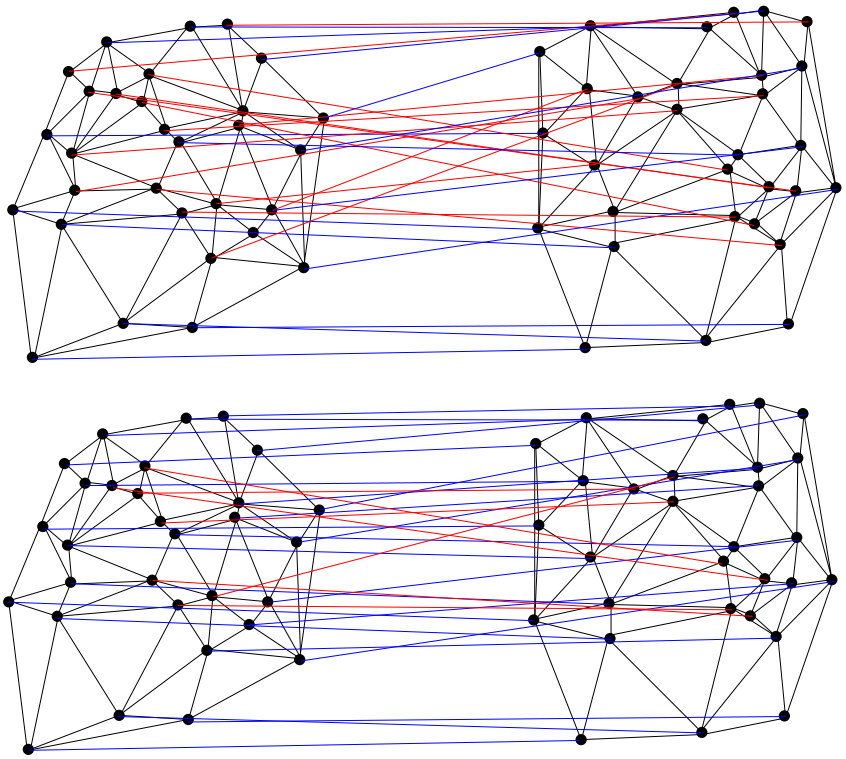


Figure 4.12: The match computed by our algorithm at iterations 0 and 1 with matching accuracy of 43 % and 70%, respectively. Blue lines show correct correspondence and red lines show the false ones.

labeling information, spatial properties, and structural compatibility. Based on such a density function, the likelihood is maximized using expectation maximization. Using representative geometric graphs extracted from several application domains, we showed that our approach outperforms existing graph matching algorithms in terms of matching quality, runtime, and memory consumption. In the following we summarize our results in this chapter.

1. Utilizing the overall graph structure for graph matching increases the matching quality. However, there is an inverse relationship between the length of the shortest path and its reliability for graph matching.

2. A major drawback of our algorithm is that it uses many parameters. Even though we estimate their optimal values experimentally, there is no guarantee that they will lead to good results for other datasets.

3. Even our algorithm scales better than other related algorithms, but it can be further improved. For example, till now we need to compute the conditional

vertex similarity between each vertex from one graph to all the vertices of the other one with a complexity of $O(d^2)$ where d is the average degree of a vertex. In addition to this, given a graph database, our algorithm linearly searches the database to extract similar graphs to a query graphs. These two issues can be further improved using pruning and indexing techniques as will be later discussed in Chapters 5 and 6.

Chapter 5

Efficient Geometric Graph Matching Using Vertex Embedding

The first step in designing any scientific application is to select a suitable data structure to model and abstract the objects under study. As we discussed in Chapter 1, mainly two representations are used, which are the vector- and the structure-based representations. For decades, the vector-based representation has been used in several domains such as in finance and physics. This because it enables us to use a broad variety of mathematical operations such as computing the sum, product, or the distance between two vectors, which can be efficiently computed.

Recently, the graph-based representation, which is a structure-based, has gained more and more interest. Even though graphs are representative data structures, they have some weakness compared to the vector-based representation. An example is the complexity of computing the distance between two graphs. Whereas the distance between two vectors can be computed easily by simple Euclidean distance, computing the optimal distance between two graphs is NP-hard [20].

In this chapter, we bridge the gap between the vector- and the graph-based representations. Our main goal is to estimate the similarity of two vertices using the Euclidean distance function, which in turn runs in constant time. We propose a graph matching framework that has two novel components. First, we propose a novel vertex embedding scheme to represent the vertices of different graphs in the vector-based representation. Initially, a set of representative vertex signatures called *prototypes* is provided. Then, each vertex is embedded into the Euclidean space using the distances between its vertex signature and the set of prototypes. Second, we propose an iterative graph matching algorithm such that the structure of the graphs is used to merge highly similar vertices into similar connected common subgraphs.

Our algorithm starts with an initial match between two given graphs and iteratively improves and expands the match. In each iteration, the similarity between two vertices is refined using a voting scheme. The vertices of the match that is computed in a previous iteration are used to estimate the similarity between any pair of vertices from the two graphs to be compared.

The remainder of this chapter is organized as follows. In Section 5.1, we discuss existing techniques that have been used to embed graphs and vertices into vector spaces. Section 5.2 presents our approach for embedding the vertices of different graphs into the Euclidean space. After that, we introduce our iterative graph matching algorithm in Section 5.3. Our experimental evaluations are discussed in Section 5.4. Finally, we summarize the chapter in Section 5.5.

5.1 Embedding into Vector Spaces

To bridge the gap between the vector- and the graph-representations, several authors have proposed to embed graphs into vector spaces [49, 100]. Based on this, a wide variety of algorithms and techniques can be used, and the complexity of several graph-based problems can be reduced. In the following, we first introduce some mathematical foundations for the embedding into vector spaces, and then we discuss how different graph-based algorithms utilize such embedding schemes.

A *finite metric space* is normally denoted by a tuple (S, d) where S is a finite set of objects and $d : S \times S \rightarrow \mathbb{R}^+$ is a distance metric [92]. Such a metric space is defined in general to adapt to different application domains. For example, the objects could be images, graphs, trees, and the distance function d is designed to fulfill the notion of similarity for each application domain. To bridge the gap between the vector- and structure-based representations, we are interested in the embedding of finite metric spaces into real-valued normed spaces.

Definition 5.1. (Embedding into Normed Spaces) *The embedding of a finite metric space (S, d) into a real-valued normed space $(\mathbb{R}^m, \delta)$ is a mapping $\varphi : S \hookrightarrow \mathbb{R}^m$, where m is the number of dimensions of the embedded space and δ is the distance function in the embedded space such that $\delta : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}^+$.*

Normally, the distance metric δ is defined based on the norm of the embedded space. For example, by using the L_2 norm, $\delta(x, y)$ is defined as $\parallel x - y \parallel_2 = \sqrt{\sum_{i=1}^{m}(x_i - y_i)^2}$.

The key idea of using such an embedding scheme for similarity search is that the distance in the embedded space $\delta(\varphi(x), \varphi(y))$ is closely related to the distance in the original space $d(x, y)$. In other words, when two objects in the finite metric space are highly similar, their embeddings in the normed space are also highly similar. However, the accuracy of a search query in the embedded space does not equal the accuracy of the same query in the original metric space. Suppose that the objects retrieved by a similarity query in the original metric space is R_o and the set of objects retrieved by the same similarity query in the embedded space is R_e, then, the embedded space can be used to search for similar objects only when $R_o \subseteq R_e$, i.e., there are no false dismissals. This condition can be fulfilled when the embedded space is *contractive*.

Definition 5.2. (Contractive Embedded Space) *Given the finite metric space (S, d), and let (U, d) be an infinite metric space that includes (S, d), then the embedded space that is induced by the mapping function $\varphi : S \hookrightarrow \mathbb{R}^m$ is said to be contractive if $\forall o_1, o_2 \in U, \delta(\varphi(o_1), \varphi(o_2)) \leq d(o_1, o_2)$.*

The above condition simply means that the distance between any two objects in the embedded space is a lower bound of their distance in the original metric space. This condition guarantees that there are no false dismissals for range similarity queries, i.e., finding all objects that are within a certain distance from a query graph [55]. For nearest neighbor queries, one can use a filter and verification technique to get the k nearest neighbors utilizing the embedded space [73]. This is done in three steps. First, get the k nearest neighbors to a query q using the similarity in the embedded space. Second, compute the actual distances between the k nearest neighbors and q using the original distance function d. Third, use such distances to issue a range query to get the k nearest neighbors.

A famous embedding method that guarantees the contractive property is the *Lipschitz embedding* [15]. It creates a normed space $\mathbb{R}^m$ such that each dimension corresponds to a reference subset of S.

Definition 5.3. ***(Lipschitz embedding)*** *Given the metric space (S, d) and a set $S_A = \{A_1, A_2, \ldots, A_m\}$ of subsets of S. Suppose the distance between an object o_i and a set $A_j \subset S_A$ is defined as $d(o_i, A_j) = min_{x \in A_j} d(o_i, x)$, then the embedding of object o_i is defined as $\varphi(o_i) = (d(o_i, A_1), d(o_i, A_2), \ldots, d(o_i, A_m))$.*

The Lipschitz embedding can be seen as a dissimilarity-based embedding. The distances between an object o_i and a set of reference objects S_A define a vector-based representation, which in turn defines the coordinates of o_i in the embedded space. For

graph classification, Riesen and Bunke [102] use the Lipschitz embedding to speed up nearest neighbor queries. They use the graph edit distance to compute the distance between each graph and the set of reference graphs S_A. To select the reference set of graphs they propose to use the well-known k-medoids cluster algorithm to split a training dataset into m disjointed subsets. Each cluster represents the set of graphs used to define a dimension of the embedded space.

A major drawback of the Lipschitz embedding is the high number of distance computation that is needed to embed an object. To overcome this problem, a special case of the Lipschitz embedding is used where each subset A_i is represented by only one object [119]. This approach was also used for graph classification [100].

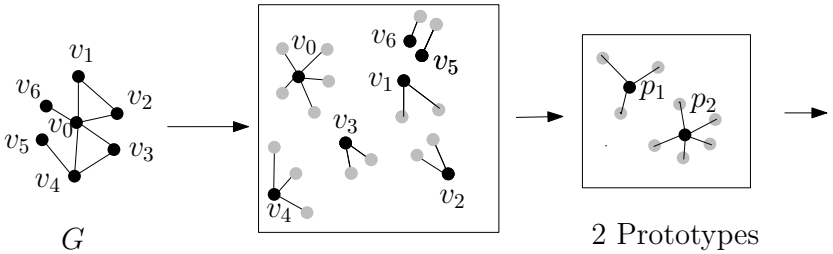
Another direction to create a vector-based representation for a graph is by extracting *features* where each feature represents a dimension of the embedded space. Gibert *et al.* [49] propose to embed a graph into a vector space using the statistics of the labels of the vertices and edges. This work is dedicated to labels that belong to the real-valued space. The graph is embedded into a vector space by using a set of representative labels $W = \{w_1, w_2, \ldots, w_n\}$ such that $w_i \in \mathbb{R}^d$. Each vertex is assigned to one representative label by taking the nearest neighbor distance. Then, the embedding of the graph is the vector that represents how many vertices from that graph are assigned to each representative.

Another well-known technique for feature extraction is by utilizing the spectra of the adjacency or the Laplacian matrix [118]. However, one of the major drawbacks of this technique is that each graph is embedded in its own vector space. In other words, to determine the similarity between vertices of two graphs based on their spectral features, the dimensions of the two embedded spaces must first be mapped, which is as complex as the graph matching problem itself [118].

In the following section, we propose a method for embedding the vertices of a graph into a vector space. To this end, we follow the special case of the Lipschitz embedding such that each dimension of the embedded space is spanned by only one reference object. Such an approach is successfully used for image similarity and graph classification [100, 119].

5.2 Vertex Embedding

In this section, we propose our vertex embedding scheme. The main idea is to use both the vertex signature and the spatial feature concepts (Section 3.3.2) to create a dissimilarity representation for the vertices of different graphs. To embed all



Vertex	p_1	p_2
v_0	0.68	0.34
$\mathbf{v_1}$	**1.10**	**1.25**
$\mathbf{v_2}$	**1.12**	**1.35**
$\mathbf{v_3}$	**1.12**	**1.35**
v_4	1.38	1.86
v_5	2.17	2.33
v_6	2.17	2.33

Embedding coordinates in 2D space

Figure 5.1: The embedding of graph G into a 2D space using two prototypes.

vertices from a graph, the graph is first decomposed into a multi-set of vertex signatures. Then, a representative set of vertex signatures, which are called *prototypes*, is provided. For a vertex, the distances between its vertex signature and the set of prototypes form a vector. This vector is considered the embedding of that vertex in the Euclidean space.

Definition 5.4. (Vertex Embedding) *Let $G = (V, E, l, c)$ be a geometric graph with the multi-set of vertex signatures $S = \{s_1, s_2, \ldots, s_n\}$, $n = |V|$. Given a set of prototypes $P = \{p_1, p_2, \ldots, p_m\}$, s.t. $m < n$, the embedding using the prototype set is a mapping $\varphi : S \hookrightarrow \mathbb{R}^m$, defined as:*

$$\varphi(s_i) := (d(s_i, p_1), d(s_i, p_2), \ldots, d(s_i, p_m))$$

where $d(s_i, p_j)$ is the vertex edit distance between the vertex signature s_i and the prototype p_j based on Definition 3.5.

To compute the distance between two vertex signatures, we use the cyclic string edit distance between their spatial features as discussed in Section 3.3.2. We illustrate the proposed embedding scheme in Figure 5.1. Using two prototypes p_1 and p_2, graph G is embedded into a two-dimensional space using the distances to the prototypes. Notice that the vertices v_1, v_2, and v_3, which are similar, are embedded close to each other.

Based on Definition 5.4, each axis of the Euclidean space corresponds to a single prototype $p_i \in P$. In other words, the prototypes span the whole embedded space. This can be interpreted as a special case of the Lipschitz embedding, as discussed in the previous section.

So the question is how much speedup one can gain by using this embedding scheme? Notice that after the embedding of the vertices from different graphs, the distance between two vertices is computed by the Euclidean distance, which is computed in linear time with respect to the size of the vector. However, without embedding, the complexity of computing the distance between two vertices is $O(d^3)$ where d is the average degree of all vertices. To give a feeling about the speedup, let us compare the complexity of building the distance matrix between the vertices of two graphs once by using the embedding scheme and once without. Notice that such a distance matrix is frequently used by several graph matching algorithms including our algorithms that were proposed in Chapters 3 and 4.

Given two graphs G and Q with $|G|$ and $|Q|$ vertices, respectively. Suppose that the degree of a vertex is d, and the size of the prototype set is m. The complexity of embedding the two graphs is $O((|G|+|Q|)md^3)$. This means that for every vertex from the two graphs, we compute its coordinates by applying the cyclic string edit distance with respect to each of the prototypes. Based on the vector-based representation, the complexity of computing the distance matrix between the vertices of G and Q is $O(|G||Q|m)$, assuming that the Euclidean distance is computed in linear time with respect to the length of the vector. On the other hand, the complexity of computing the distance matrix using the vertex edit distance is $O(|G||Q|d^3)$. By removing the asymptotic notation, the speedup is computed as follows:

$$\text{speedup} = \frac{|G||Q|d^3}{(|G|+|Q|)md^3 + |G||Q|m} \tag{5.1}$$

Without loss of generality, let us assume that the size of a graph is n. Then, the previous equation is simplified as:

$$\begin{aligned} \text{speedup} &= \frac{n^2 d^3}{2nmd^3 + n^2 m} \\ &= \frac{1}{\frac{2m}{n} + \frac{m}{d^3}} \end{aligned} \tag{5.2}$$

Based on Equation 5.2, the speedup that is gained by using the embedding scheme is defined as the ratio of the size of the vector, i.e., the number of dimensions of the embedded space, to the graph size and the vertex degree. Once the number of dimensions of the Euclidean space becomes greater than d^3, the embedded scheme becomes a bottleneck and the performance gets worse. Also, it is not recommended to use the embedding scheme for small graphs where the graph size equals the number

of dimensions of the embedded space. As a result, we conclude that we gain the best speedup of our approach when using large and dense graphs and we do not recommend to use it for small and sparse graphs.

Now we turn our attention to formalize the relationship between the cyclic string edit distance between two vertices and their Euclidean distance [103]. The Euclidean distance between two vectors $\varphi(s_1)$ and $\varphi(s_2)$ is defined as:

$$\| \varphi(s_1) - \varphi(s_2) \|_2 = \sqrt{\| \varphi(s_1) \|^2 + \| \varphi(s_2) \|^2 - 2\varphi(s_1) \cdot \varphi(s_2)} \tag{5.3}$$

Given that the cyclic string edit distance, denoted as $d(.)$, is a metric, and due to the triangle inequality $|d(s_1, p_i) - d(s_2, p_i)| \leq d(s_1, s_2)$, we have:

$$\begin{aligned} \| \varphi(s_1) - \varphi(s_2) \|_2 &= \left(\sum_{i=1}^{m} d(s_1, p_i)^2 + \sum_{i=1}^{m} d(s_2, p_i)^2 - 2 \sum_{i=1}^{m} (d(s_1, p_i) d(s_2, p_i)) \right)^{\frac{1}{2}} \\ &= \left(\sum_{i=1}^{m} (d(s_1, p_i) - d(s_2, p_i))^2 \right)^{\frac{1}{2}} \\ &\leq (m \cdot d(s_1, s_2)^2)^{\frac{1}{2}} \\ &= \sqrt{m} \cdot d(s_1, s_2) \end{aligned} \tag{5.4}$$

As a result, $\frac{\|\varphi(s_1) - \varphi(s_2)\|_2}{\sqrt{m}}$ is a lower bound to the cyclic string edit distance between the two vertex signatures s_1 and s_2. This relationship, as we discussed in the previous section, guarantees that the embedded space is contractive. This means that no false dismissals occur for a similarity query in the embedded space.

5.2.1 Prototype Selection

Matching two graphs based on their vector-based representation relies on the embedding of similar vertices close to each other. To guarantee this, a representative set of prototypes must be used. This makes the prototypes the basis and the crucial part of the proposed embedding scheme. Furthermore, in addition to the prototypes themselves, the number of prototypes is also a crucial decision. In this section, we discuss how prototypes are generated or selected from a graph training dataset.

Prototype selection is studied in the literature to improve well-known problems for the k-NN classifier [93]. Examples for such problems are low tolerance to noise and less efficiency due to distance computation. For vertex embedding, a prototype selection method should also take care of these issues.

In the following, we discuss three prototype selection methods: *random selection* (RS), *medoids selection* (MS), and *spanning selection* (SP).

• **Random Selection.** For graphs where a training sample is not available, the random selection method is used. Prototypes of vertex signatures are artificially created. The user specifies the number of prototypes, the size of each, and the labeling alphabet. Then, labels are assigned randomly to the edges and the neighboring vertices. In addition to this, the lengths of the edges and the values of the angles are randomly assigned. Then, they are normalized in the interval $[0, 1]$, as proposed in Equation 3.4.

Algorithm 5.1: Medoids selection method

Input: A set of vertex signatures $S = \{s_1, s_2, \ldots, s_n\}$ and the size of the prototype set m

Output: A set of prototypes $P = \{p_1, p_2, \ldots, p_m\} \subseteq S$

/* D is a distance matrix s.t. $D \in \mathbb{R}^{n \times n}$ */
/* $d(.)$ is the cyclic string edit distance function */
1 **foreach** $s_i, s_j \in S$ **do**
2 $\quad D_{ij} \leftarrow d(s_i, s_j)$
/* P represents the medoids of the clusters, m is the number of prototypes */
3 $P \leftarrow MedoidsCluster(D, m)$
4 **return** P

• **Medoids Selection.** The medoids selection method is used to select prototypes from a graph training dataset. As described in Algorithm 5.1, it utilizes the well-known k-medoids clustering algorithm to select k prototypes. Each prototype represents the center of a cluster of vertex signatures. The k-medoids algorithm uses a distance matrix that is created from vertex signatures extracted from the training dataset. The distance between two vertex signatures is computed based on the cyclic string edit distance. The medoids selection can be interpreted as a frequent prototype selection method. Each cluster extracted by the k-medoids represents a group of a frequent vertex signature.

• **Spanning Selection.** We adopt the spanning prototype selection method proposed in [103] to select prototypes of vertex signatures. This prototype selection method finds vertex signatures as uniformly distributed as possible from a training dataset. The median vertex signature is first selected, which is the one with the minimal sum of distances to all other vertex signatures.

Definition 5.5. ***(Median Vertex Signature)*** *Given a set of vertex signatures* $S = \{s_1, s_2, \ldots, s_n\}$*, the median vertex signature* s_{median} *is defined as:*

$$s_{median} := \arg\min_{s_i \in S} \sum_{s_j} d(s_i, s_j) \tag{5.5}$$

where the distance $d(s_i, s_j)$ is the cyclic string edit distance between the two vertex signatures s_i and s_j. To select the remaining prototypes iteratively, the vertex signature with the farthest distance from the already selected prototypes is selected, as illustrated in Algorithm 5.2. To further improve the spanning selection method, we restrict the prototype set to have only one prototype that has only one edge. In other words, we allow only one pendent vertex to exist in the prototype set. This way, the prototype set will be more discriminative and leads to less dependencies between the different dimensions of the embedded space.

Algorithm 5.2: Spanning selection method

Input: A set of vertex signatures $S = \{s_1, s_2, \ldots, s_n\}$ and the size of the prototype set m

Output: A set of prototypes $P = \{p_1, p_2, \ldots, p_m\} \subseteq S$

```
/* initialize P with the median vertex signature */
P ← {s_median}
S ← S \ {s_median}
while |P| < m and S ≠ φ do
    p = arg max_{s_i∈S} min_{p_j∈P} d(s_i, p_j)
    S ← S \ {p}
    /* size(p) denotes number of edges in the prototype p */
    if (size(p) ≥ 2) or (size(p) = 1 and ∀s_i ∈ S, size(s_i) ≥ 1) then
        P ← P ∪ {p}
return P
```

In Figure 5.2, we show the different prototype selection methods. The RS method follows an unsupervised approach by selecting the prototypes randomly. On the other side, the MS and SP methods follow a supervised approach and select prototypes from a training dataset. MS selects the prototypes as medoids of clusters of frequent prototypes. SP selects the prototypes uniformly to cover the whole training dataset. The advantage of the spanning selection method over the medoids method is that it follows a deterministic scheme for selecting the prototypes. However, the medoids method creates different prototype sets for different executions of the algorithm even when using the same training dataset. This is because the k-medoids clustering

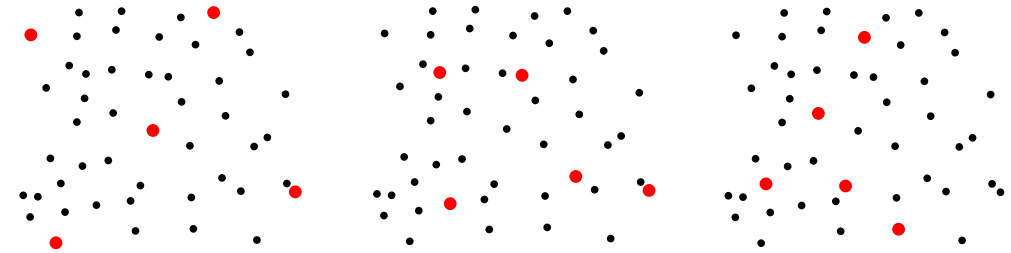


Figure 5.2: Different prototype selection methods.

algorithm is sensitive to the initial assignments of the clusters, i.e., different initial assignments lead to different clustering results.

5.3 Iterative Graph Matching

After the embedding of all graphs into the Euclidean space, the k-nearest-neighbor algorithm can be used to find similar vertices. However, matching two graphs based on the similarity in the Euclidean space creates a match that is structurally inconsistent, i.e., two neighboring vertices from one graph may be mapped to non-neighboring vertices from another one. This is because the distance function used in the embedding process considers only a vertex and its direct neighbors. As a result, graph connectivity is not preserved in the Euclidean space.

We solved this problem in Chapter 4 by formalizing the graph matching problem as maximum likelihood estimation. Then, the generalized expectation algorithm is used to select the parameters of the mixture model that maximize the likelihood function. Such a solution gives good results and scales better than the related graph matching algorithms, as shown in Section 4.4. Unfortunately, the algorithm proposed in Chapter 4 cannot utilize the embedding scheme. This is because such an algorithm utilizes the conditional vertex similarity between two vertices at the E- and M-steps, which is not defined in the Euclidean space.

To solve this problem, in this section, we propose an iterative graph matching algorithm that integrates the similarity between the vertices in the Euclidean space with structural compatibility. Our proposed algorithm follows the continuous optimization style and iteratively refines the match between two graphs $G = (V, E)$ and $Q = (U, T)$ to create similar connected common subgraphs. Each iteration consists of three steps:

1. solving the assignment problem,

2. similarity refinement, and

3. candidate selection.

In an iteration t, the best match $M^t = \{0,1\}^{|V|\times|U|}$ from a set of candidate matches $C^t = \mathbb{R}^{|V|\times|U|}$ is selected by solutions to the assignment problem. In the similarity refinement step, the similarities between the vertices of one graph to the vertices of the other one are updated in a voting scheme. Vertices from M^t vote to quantify structural compatibility between each pair of vertices from the two graphs. Then, a set of k candidate matches C^{t+1} is selected to be used in the next iteration.

In the following section, we discuss how to initialize the match between two graphs. Section 5.3.2 discusses a greedy algorithm to solve the assignment problem. Then, in Section 5.3.3, we formalize a probabilistic approach to refine the similarity between the vertices of two graphs based on structural compatibility. Finally, in Section 5.3.4, we propose a pruning technique to exclude non-similar pairs of vertices from the match between two graphs. Through the rest of this section, the distance between two vertices refers to the Euclidean distance between their coordinates in the embedded space.

5.3.1 Candidate Initialization

The set of candidates used in the first iteration of the proposed graph matching algorithm is called the *initial candidate set*. The vertices from the initial candidate set are considered seeds for the iterative algorithm. We call such seeds of vertices *anchor vertices* or anchors. The anchors affect the number of iterations needed for the convergence. Two issues should be considered when selecting them: 1) the similarity of the anchors from one graph to another, and 2) the degree of the anchors, i.e, the vertex degree. Since our matching algorithm iteratively increases the match between two graphs, anchors that are highly similar and have a higher vertex degree guarantee a faster convergence rate and produce good matching results. We propose two anchor selection methods guided by requirements from different application domains.

• **Common Subgraph Matching.** The size of a common subgraph between two graphs G and Q is less than or equal to $\min\{|G|, |Q|\}$. To speed up the convergence, vertices from the non-common subgraph should be avoided and not added to the

initial candidate set. For this type of matching, the similarity of the vertices is used to prune the vertices that belong to the non-common subgraph. To do this, a similarity matrix is created from the vertices of G and Q. The similarity between two vertices is defined as the similarity of their embedding in the vector space. Then, for each vertex $v_i \in G$, the candidate pair of vertices (v_i, u_j) is added to the candidate set when u_j is the most similar vertex to v_i and their similarity exceeds a certain threshold c. To compute the similarity threshold, we first create a vector $\vec{x} \in \mathbb{R}^{|G|}$ such that $x_i = \max_{u_j \in Q} s(v_i, u_j)$ and $s(v_i, u_j)$ is the similarity in the Euclidean space between the two vertices v_i and u_j. Based on this, the threshold $c = \mu(\vec{x})$ where μ is the mean function.

• **(Sub)graph Matching.** To increase the tolerance to changes in graph structure and spatial probabilities, anchors for subgraph matching are not filtered based on the similarity of the vertices but based on the vertex degree. To do this, first, a similarity matrix is created from the vertices of Q and G. The similarity between two vertices is defined as the similarity of their embedding in the vector space. Then, a candidate pair of vertices (v_i, u_j) is added to the candidate set when u_j is the most similar vertex to v_i and their degrees exceed 3. This number guarantees that the match between two graph is refined and expanded faster than picking vertices of lower degree, an aspect that will be discussed later in the similarity refinement section.

5.3.2 Solving the Assignment Problem

Given a candidate set of similar vertices, the question is how to select the best assignment from the vertices of one graph to the vertices of another one such that the selected pairs of vertices have the highest similarity? This problem is normally referred as the assignment problem, which can be solved by different techniques. Mostly, the Hungarian algorithm is used to solve the assignment problem. However, such an algorithm runs in cubic time with respect to graph size. To overcome this complexity, we propose to use a greedy algorithm that has less runtime complexity and computes an approximate solution to the assignment problem. The greedy algorithm iteratively selects the highest similar pair of vertices from the candidate set and adds it to the match. Specifically, it adds the pair of vertices (v_i, u_j) when $s(v_i, u_j) = \max_{v_k} s(v_k, u_j)$ and $s(v_i, u_j) = \max_{u_l} s(v_i, u_l)\}$ where s represents the similarity between two vertices.

5.3.3 Similarity Refinement

The match between graphs based on the initial candidate set utilizes only the similarity in the Euclidean space. Thus, in an iterative approach, the match is expanded and improved by utilizing more information about the structure of the graphs. The main idea is that the computed match in iteration t is used to refine the similarity between the vertices, which is then utilized in the next iteration. Given two graphs G and Q with their vertex sets V and U, respectively, the similarity between the vertices is updated based on a probabilistic voting scheme. For this, we adopt a Bayesian formulation similar to the one presented in [28]. Given a match $M^t = \{0,1\}^{|V|\times|U|}$ computed at an iteration t, the similarity between two vertices $v_k \in V$ and $u_l \in U$ is updated as a conditional joint probability distribution $P(V,U|M^t)$. To compute such a probability distribution, an auxiliary variable of a match $M \in M^t$ is used. The probability distribution is computed by marginalizing $P(V,U,M|M^t)$ over M. By using the chain rule, the similarity between v_k and u_l is defined as:

$$\begin{aligned} P(v_k,u_l|M^t) &:= \sum_{\mathrm{m}_{ij}\in M^t} P(v_k,u_l,\mathrm{m}_{ij}|M^t) \\ &= \sum_{\mathrm{m}_{ij}\in M^t} P(v_k|u_l,\mathrm{m}_{ij},M^t)\ P(u_l|\mathrm{m}_{ij},M^t)\ P(\mathrm{m}_{ij}|M^t) \end{aligned} \tag{5.6}$$

where $P(\mathrm{m}_{ij}|M^t)$ is a prior representing the probability of choosing the match $\mathrm{m}_{ij} \in M^t$. $P(u_l|\mathrm{m}_{ij},M^t)$ describes the probability of u_l being a neighbor of vertex u_j. $P(v_k|u_l,\mathrm{m}_{ij},M^t)$ represents the probability that v_k is similar to u_l given the status of the match m_{ij}. This marginalization can be seen as a probabilistic voting such that the voters are the matches $\mathrm{m}_{ij} \in M^t$. In the following, we detail the realization of the three probabilities in Equation 5.6.

$$P(m_{ij}|M^t) = \frac{\text{similarity}(v_i,u_j)\times m_{ij}}{\sum_{m_{kl}\in M^t}\text{similarity}(v_k,u_l)\times m_{kl}} \tag{5.7}$$

where the similarity between two vertices is computed in the previous iteration $t-1$.

$$P(u_l|m_{ij},M^t) = \begin{cases} \frac{1}{\deg(u_j)}, & \text{if } u_l \in \mathrm{N}(u_j) \text{ and } m_{ij}=1 \\ 0, & \text{otherwise} \end{cases} \tag{5.8}$$

for a vertex u_j, $\deg(u_j)$ denotes its degree and $\mathrm{N}(u_j)$ describes the set of its direct neighboring vertices.

$$P(v_k|u_l, m_{ij}, M^t) = \begin{cases} 1, & \text{if } m_{ij} = 1 \text{ and } m_{kl} = 1 \text{ and } v_k \in N(v_i) \\ \frac{\exp(-d(v_k,u_l))}{Z}, & \text{if } m_{ij} = 1 \text{ and } m_{kl} = 0 \text{ and } v_k \in N(v_i) \\ 0, & \text{otherwise} \end{cases} \tag{5.9}$$

where $d(v_k, u_l)$ is the Euclidean distance between the two vertices v_k and u_l. Z is a normalization factor which is defined as:

$$Z := \sum_{v_k \in N(v_i)} exp(-d(v_k, u_l)) \tag{5.10}$$

To summarize, the probability of matching any two vertices $v_k \in V$ and $u_l \in U$ increases when 1) they are similar in the Euclidean space and 2) their neighbors exist in the match M^t. The more neighbors exist in M^t, the higher the probability that v_k is matched to u_l.

Algorithm 5.3: Candidate selection

Input: A match M and a similarity matrix S between the vertices of two graphs

Output: The candidate set C that is going to be used in the next iteration

```
1  c ← 0
2  if common subgraph matching then
3      x̂ ← 0
4      foreach v_k ∈ V do
5          x̂ ← x̂ ∪ max_{u_j ∈ U} s(v_i, u_j)
       /* c is the similarity threshold, μ and stdev are the mean and standard deviation
          functions, Equation 5.11 */
6      c ← μ(x⃗) − stdev(x⃗)
   /* %*% is the element-wise multiplication between two matrices */
7  C ← M %*% S
   /* Add all neighboring vertices */
8  foreach m_ij ∈ M do
9      foreach v_k ∈ V, u_l ∈ U do
10         if m_ij = 1 ∧ v_k ∈ N(v_i) ∧ u_l ∈ N(u_j) ∧ S(k,l) > c then
11             C_kl ← S(k,l)
12 return C
```

5.3.4 Candidate Selection

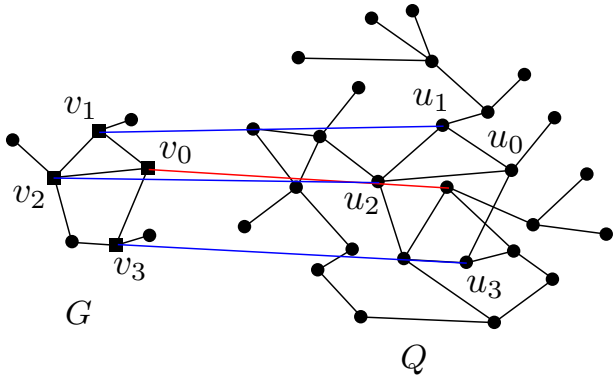
Once the similarity between the vertices is updated following the previous probabilistic approach, the candidate set C^{t+1} is selected to be utilized by the graph matching step at the next iteration. We propose to prune non-similar pairs of vertices before solving the assignment problem. By this, we guarantee that our heuristic does not falsely select non-similar vertices and does not add them to the match. We select the candidate set C^{t+1} as follows. First, all the vertices v_i and u_j s.t. $m_{ij}^t = 1$ are used to initialize C_{t+1}. Then, all neighboring vertices to the vertices in C^{t+1} are selected as candidates. For common subgraph matching, we further refine the candidate set to prune the vertices from the non-common subgraph. We remove all pairs of vertices from the candidate set where their similarity is under a certain threshold c. To compute this similarity threshold c, we first create a vector $\vec{x} \in \mathbb{R}^{|G|}$. An element x_i represents the maximum similarity between vertex $v_i \in V$ and all the vertices of U. It is defined as $x_i = \max_{u_j \in U} s(v_i, u_j)$, where $s(v_i, u_j)$ is the similarity between the two vertices v_i and u_j as computed in the previous iteration. Then,

$$c = \mu(\vec{x}) - stdev(\vec{x}) \tag{5.11}$$

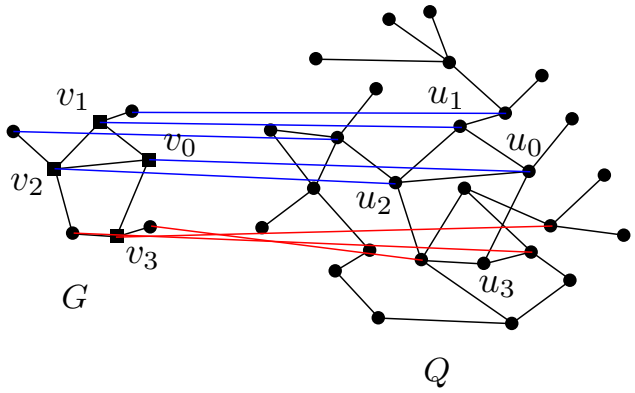
where μ and $stdev$ are the mean and standard deviation of the vector $\vec{x}$, respectively. Our candidate selection method is outlined in Algorithm 5.3.

5.3.5 Convergence

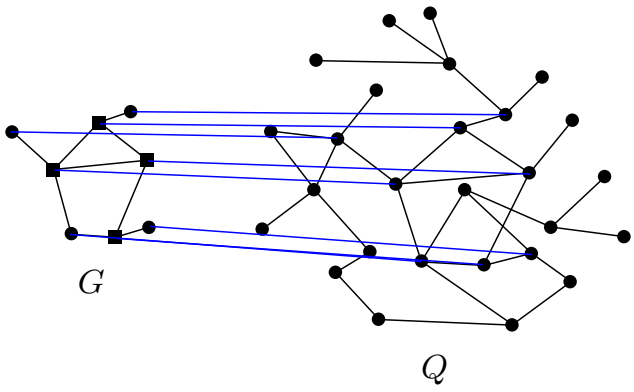
Given a scoring function $f : \mathcal{M} \to \mathbb{R}$ that quantifies the quality of any match from the matching space $\mathcal{M}$, our algorithm converges when the change in the score for two consecutive iterations is statistically insignificant. However, the score reported at different iterations is not monotonically increasing. In other words, the score may decrease in a future iteration. This is because the match between two graphs is iteratively expanded. As a result, new pairs of vertices may be added to the match such that they vote negatively to the similarity of other pairs of vertices. For our algorithm, the convergence criterion is $|S(M_{t+1}) - S(M_t)| < \alpha$, where α is a threshold defined by the user. Our experiments (Section 5.4) show that a good value of α is 10^{-4} for subgraph matching and 10^{-2} for common subgraph matching. These values guarantee a good matching accuracy and a fast convergence time. We outline our iterative approach in Algorithm 5.4. Notice that our algorithm returns the match with the highest matching score and does not return the match at the convergence iteration.



a) iteration $t = 1$, accuracy $= 37.5\%$.



b) iteration $t = 2$, accuracy $= 62.5\%$.



c) iteration $t = 4$, accuracy 100%.

Figure 5.3: The iterative graph matching between two graphs G and Q. The anchor vertices for G are represented by squares. A correct correspondence is drawn in blue and a false one is drawn in red. a), b), and c) represent the matching results at the 1^{st}, 2^{nd}, and 3^{rd} iteration, respectively.

Algorithm 5.4: Iterative graph matching

Input: Two geometric graphs $G = (V, E)$ and $Q = (U, T)$
Output: The match $M^{'}$

/* D is a dissimilarity matrix using the Euclidean distance s.t. $D \in \mathbb{R}^{|V|\times|U|}$ */
foreach $v_i \in V$, $u_j \in U$ **do**
  $d_{ij} \leftarrow d(v_i, u_j)$
$min\_value \leftarrow \min(D)$
$max\_value \leftarrow \max(D)$
/* convert the distance matrix D to a normalized similarity matrix S */
/* $S \in \mathbb{R}^{|V|\times|U|}$ */
foreach $v_i \in V$, $u_j \in U$ **do**
  $s_{ij} \leftarrow 1 - \frac{d_{ij} - min\_value}{max\_value - min\_value}$
/* candidate initialization, Section 5.3.1, C is a candidate set s.t. $C \subseteq S$ */
$C \leftarrow CandiateInit(G, Q, S)$
$oldscore \leftarrow 0$
$newscore \leftarrow 1$
$highestscore \leftarrow 0$
/* The algorithm converges when the change in the score is statistically insignificant */
while $|newscore - oldscore| > \alpha$ **do**
  /* solve the assignment problem, Section 5.3.2 */
  $\{M, newscore\} \leftarrow SolveAssignment(C)$
  /* vertex-to-vertex similarity refinement, Section 5.3.3 */
  $S \leftarrow Similarity(G, Q, M)$
  /* C is a candidate set s.t. $C \subseteq S$, k is size of the candidate set, Section 5.3.4 */
  $C \leftarrow CandidateSelection(G, Q, S, M, k)$
  $oldscore \leftarrow newscore$
  /* save the match with the highest score */
  if $oldscore \geq highestscore$ **then**
    $M^{'} \leftarrow M$
    $highestscore \leftarrow oldscore$
return $M^{'}$

Table 5.1: The three geometric graphs that are used in our evaluations. $|V|$ and $|E|$ denote the number of vertices and the number of edges, respectively.

Road network	$\|V\|$	$\|E\|$
California	1365	1990
City of Oldenburg	3494	4348
North America	7517	10088

We finish this section by an example that gives a feeling about our iterative graph matching algorithm, which is shown in Figure 5.3. At the end of the 1^{st} iteration, the similarity between $v_0 \in G$ and $u_0 \in Q$ increases since all the neighbors of v_0 are matched to the neighbors of u_0, whereas the similarity between v_3 and u_3 decreases since there is no neighbor of v_3 that is matched to a neighbor of u_3.

5.4 Experimental Evaluation

In this section, our proposed solution to the graph matching problem is empirically evaluated. To this end, we use geometric graphs that are extracted from three road networks: California, North America, and the City of Oldenburg [4]. Latitude and longitude are used as the x and y coordinates of the vertices. Since a road segment between two intersections is represented by several nodes in the road network, we simplified them using the Douglas Peucker algorithm [41]. This algorithm is used to reduce the number of points representing a curve. Table 5.1 shows the number of vertices and edges for the simplified road networks. We also show an example from our dataset in Figure 5.4. It represents the geometric graph that is extracted from the road network of the City of Oldenburg.

The focus of this section is to empirically evaluate two criteria for a graph matching algorithm. The first criterion is the scalability with respect to graph size. It is measured by the runtime required to match different graphs. The second criterion is the quality of the match computed by the graph matching algorithm. Algorithms with high matching quality are more resistant to changes in graph structure and spatial attributes. For our experiments, the quality of the match is measured by the matching accuracy. For two graphs G and Q, the matching accuracy is estimated by the agreement between the match M that is computed by a graph matching algorithm, and the ground truth match $\hat{M}$, which is formalized in Equation 3.10, page 64.

We compare our graph matching algorithm, called **VEM**, which is proposed in this chapter, with the following related algorithms:

Figure 5.4: The geometric graph of the road network of the City of Oldenburg.

1. **MixModel** (Chapter 4). The **MixModel** algorithm follows a probabilistic approach to solve the graph matching problem. It computes the similarity of two vertices based on 1) the similarity of their neighborhoods and 2) the similarity of the paths that connect them to other vertices in the graphs. The main idea of this approach is to use maximum likelihood estimation to find the best match between two graphs. A mixture model of the set of possible correspondences between the vertices of two graphs is defined. Then, the solution of the graph matching problem between two graphs is the mixture model that maximizes the likelihood function, which is optimized using the expectation maximization technique.

2. **CSv1** (Chapter 3). The **CSv1** algorithm follows a graph edit distance approach and utilizes only the neighborhoods of the vertices for graph matching. This algorithm initially creates a vertex-to-vertex distance matrix based on the cyclic string edit distance of the spatial features of the vertices. To this end, the first set of edit operations are used, which is proposed in Section 3.3.2. Then, the Hungarian algorithm is used to select the best match between the two graphs.

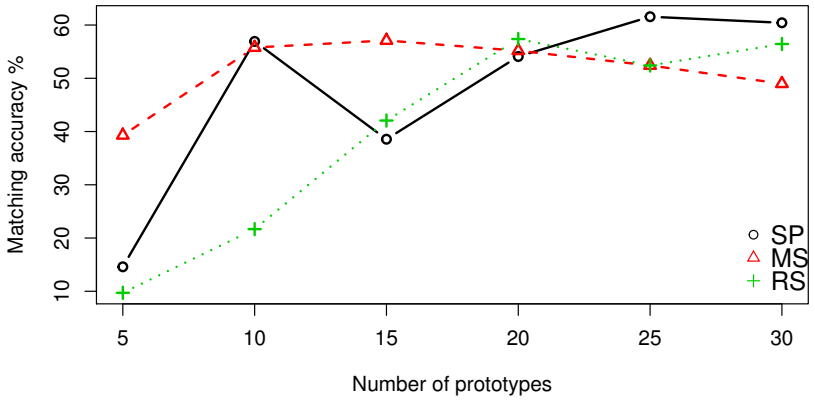


Figure 5.5: The effect of the number of prototypes and the prototype selection method on graph matching accuracy.

3. **Heat**. It is the heat kernel embedding algorithm proposed by Xiao *et al.* [130]. Such an algorithm initially uses the spectra of the normalized Laplacian matrix to embed the vertices of a graph into the Eigenspace. Then, the match between two graphs is estimated following a variation of the Scott and Longuet-Higgins algorithm [110]. This algorithm uses SVD to create a similarity matrix between the vertices of the two graphs based on their embeddings in the Eigenspace. Then, two vertices are matched if their similarity is the maximum among all similarities of the matches containing any of them.

In the following section, we study the effects of various parameters of our graph matching algorithm. This includes the prototype selection method, the size of the prototype set, and the convergence threshold α. After that, in Section 5.4.2, we evaluate the matching quality followed by a scalability study in Section 5.4.3.

5.4.1 Parameters Analysis

In this section, we first study the effect of the prototype selection method and the number of prototypes on the matching accuracy. Then, we discuss the matching accuracy with different values of the convergence threshold α for both subgraph and common subgraph matching.

Two datasets are used for the parameter analysis: one for (sub)graph matching and the other for common subgraph matching. The subgraph matching dataset

contains 5 subgraphs extracted from the California road network. We applied random spatial and a few structural distortions to these subgraphs [3]. The sizes of the subgraphs are 60, 92, 116, 123, and 128. We matched each of them against the California road network and averaged the matching accuracy. The common subgraph dataset consists of 5 different geometric graphs created from the California road network. Each geometric graph has a common subgraph with the original California road network in addition to a non-common subgraph. To do this, we first partition the California road network into different clusters. The number of clusters used varies from 3 to 7. We apply spatial distortion to some clusters to make them non-similar [3]. For the five graphs, the number of distorted clusters are 1, 1, 2, 3, and 3, respectively. The remaining of the clusters are considered as the common subgraph between the new created graph and the original California road network. The sizes of the common subgraphs are 451, 334, 546, 592, and 689 such that some common subgraphs are disconnected. We matched all 5 graphs against the California road network and averaged the matching accuracy.

Prototype Selection

We empirically evaluate the three prototype selection methods: random selection (RS), medoids selection (MS), and spanning selection (SP), see Section 5.2. For this test, we use the dataset of subgraph matching. Since the result of the MS method is affected by the initial medoids assignment, we run the k-medoids several times with different initial assignments. The prototypes that create the lowest within-cluster sum of distances are chosen. As shown in Figure 5.5, the spanning selection method has the highest matching accuracy. On the other side, the RS method has nearly the same matching accuracy as MS. This result confirms the analysis reported in [93], which says that a random prototype selection method gives good results in many cases. Figure 5.5 also shows the effect of the number of prototypes on the matching accuracy. A small number of prototypes does not cover the diversity of the vertex signatures extracted from the graphs, which gives low matching accuracy for all three selection methods. Also, a high number of prototypes decreases the matching accuracy. This is because more non-representative vertex signatures are selected as prototypes, especially for the MS method. From this experiment, we conclude that 10 prototypes give a good matching accuracy. We observe that the increase in the matching accuracy is statistically insignificant for values more than 10. On the other hand, a higher number of prototypes decreases the scalability of the matching algorithm. This is because the algorithm computes the cyclic string edit

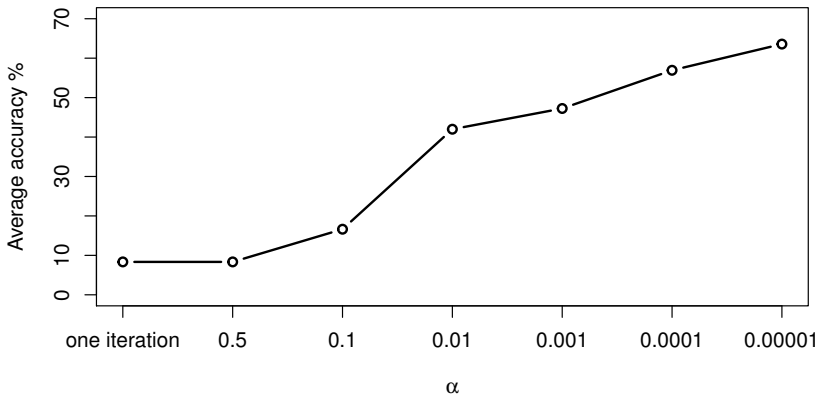


Figure 5.6: The effect of the convergence threshold α on the accuracy of subgraph matching.

distance between a vertex and each of the prototypes, which runs in cubic complexity with respect to the vertex degree.

The Convergence Threshold α

We test the convergence threshold α for both subgraph and common subgraph matching. We also test the matching accuracy that is achieved by running our algorithm for only one iteration. For subgraph matching as seen in Figure 5.6, a higher value of α achieves better matching quality. For subgraph matching, the task is normally to locate a query graph in a larger one, i.e., which subgraph from the larger graph is highly similar to the query graph. A higher value for α means that our algorithm requires more iterations to find such a subgraph. For subgraph matching, we assign α a value of 0.0001. This value is a compromise between the matching accuracy and the scalability.

Figure 5.7 shows the effect of α on common subgraph matching. Notice that one iteration of our algorithm gives a matching accuracy of 96.5 %. The best matching accuracy occurs when $\alpha = 0.01$, which is the value used for the rest of our experiments and for common subgraph matching. When comparing Figures 5.7 and 5.6, we see a different effect of α on the matching accuracy. This is because the common subgraph between the California road network and its distorted version has very little spatial and structure differences. As a result, the initial match computed at the first iteration gives high matching accuracy. However, for subgraph matching, there are significant

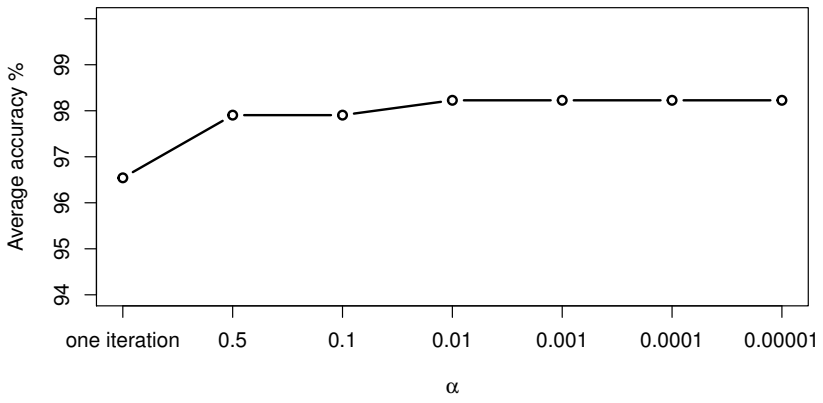


Figure 5.7: The effect of the convergence threshold α on the accuracy of common subgraph matching.

spatial and structure differences between the two compared graphs. For subgraph matching, a higher matching accuracy is achieved when running the algorithm for more iterations using higher values for α.

5.4.2 Matching Quality

In this section, we test the matching quality of our algorithm against the three matching algorithms **MixModel**, **CSv1**, and **Heat**. First, we discuss the matching accuracy for subgraph matching, then, we analyze the results for common subgraph matching.

Subgraph Matching

We use the California road network for subgraph matching. From this road network, we extracted 5 initial subgraphs. From these 5 subgraphs, we created two datasets such that each one has 20 graphs. The first dataset was created by applying only structural distortion for the initial 5 subgraphs. The second dataset was created by using only spatial distortion [3]. Distortion is applied at an increasing level. We computed the amount of distortion needed to make an initial subgraph non-similar to the distorted one. Then, we divide this amount to create four levels of distortion such that each distortion level is represented by 5 subgraphs. We compare the graphs

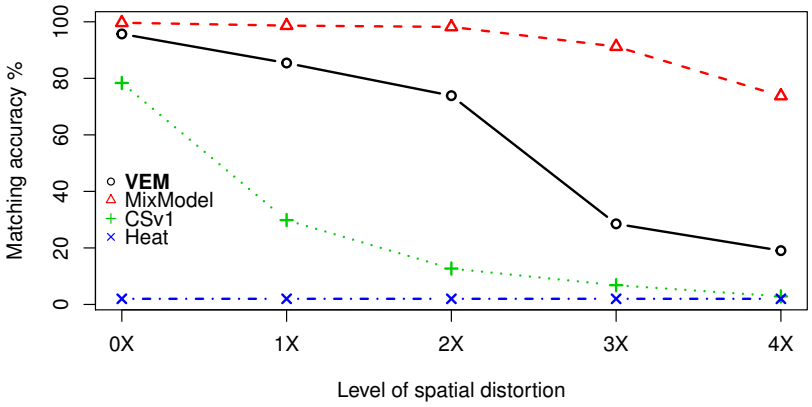


Figure 5.8: The effect of spatial differences on the matching accuracy for subgraph matching.

at each distortion level against the California road network and average the matching accuracy.

Figure 5.8 shows the effect of spatial distortion on the matching accuracy for different matching algorithms. The value reported under $0X$ is the accuracy of matching the initial 5 subgraph against the California road network. This can be seen as an instance of the exact matching problem since the initial 5 subgraphs do not have any distortion with respect to the California road network. As one can see, our proposed algorithm **VEM** comes in the second place. The best matching accuracy is for our algorithm **MixModel**, Chapter 4. This is because the latter algorithm uses more information about the overall graph structure. The **Heat** algorithm has nearly zero matching accuracy for all levels of distortions. This demonstrates the weakness of spectral approaches for matching graphs that differ in their sizes. On the other side, **CSv1** gives better results than **Heat** even though the global connectivity of the graph is not considered by textbfCSv1. We conclude that our proposed vertex similarity metric, Section 3.3.2, which is used by both **VEM** and **CSv1**, is very efficient in finding similar vertices.

Figure 5.9 shows the relationship between the structural differences and the matching accuracy for different matching algorithms. Additionally to this test, **MixModel** gives the best result and **Heat** gives the worst result. Even though **MixModel** gives higher matching accuracy than **VEM**, it does not scale well as **VEM**, as will be shown later.

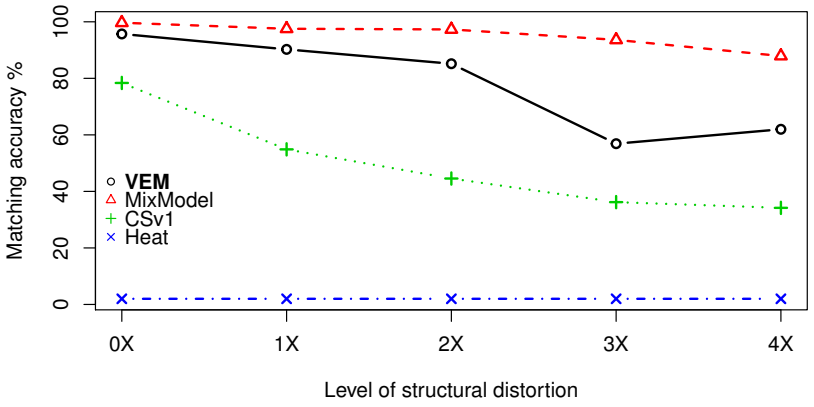


Figure 5.9: The effect of structural differences on the matching accuracy for subgraph matching.

Common Subgraph Matching

The accuracy for common subgraph matching is tested by using all three road networks: California, North America, and the city of Oldenburg. From each road network we created another distorted graph [3]. To this end, initially, each road network is clustered into groups of vertices. Then, distortion is applied to some of the clusters to make them non-similar. The remaining clusters are left without any distortion to create a common subgraph. We matched each graph with its distorted version and computed the matching accuracy. We show an example in Figure 5.10 for the California road network and its distorted graph.

Figure 5.11 shows the accuracy for common subgraph matching for the three road networks. The results show that our proposed algorithm outperforms the related graph matching algorithm. The matching accuracy for our algorithm is nearly 100% for the three road networks. The difference in the matching accuracy between **MixModel** and **VEM** is statistically insignificant. This is because the common subgraph between any two compared graphs has very little distortion. On the other hand, the distortion between the graphs was higher for subgraph matching. As a result, **MixModel** gives the best result. Even for common subgraph matching, **heat** performs the worst. This shows again that spectral approaches are sensitive to structural and spatial changes.

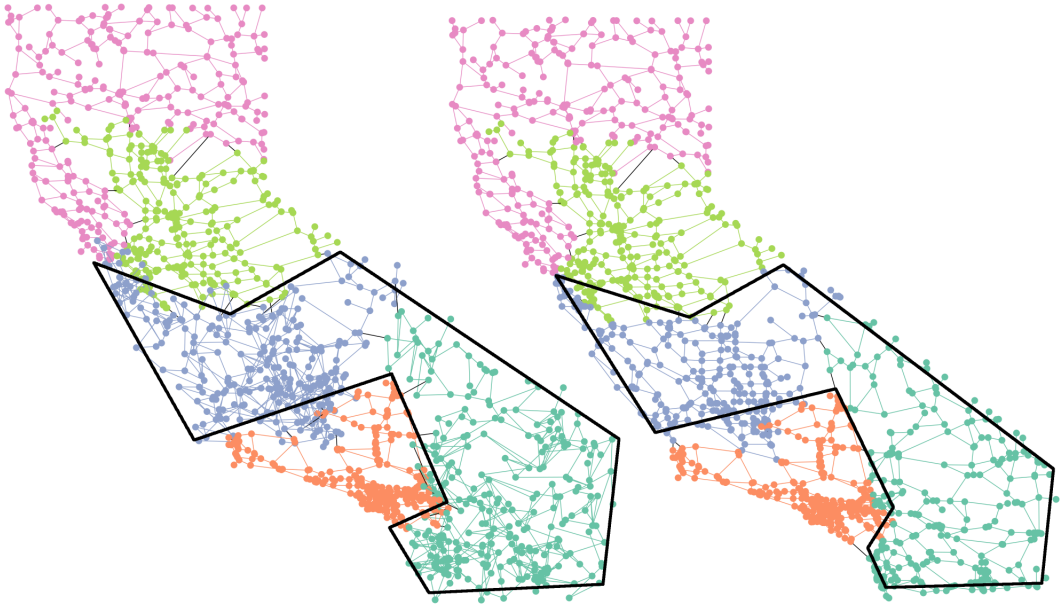


Figure 5.10: The California road network and its distorted graph. The subgraph in the black polygon highlights the non-common subgraph between them.

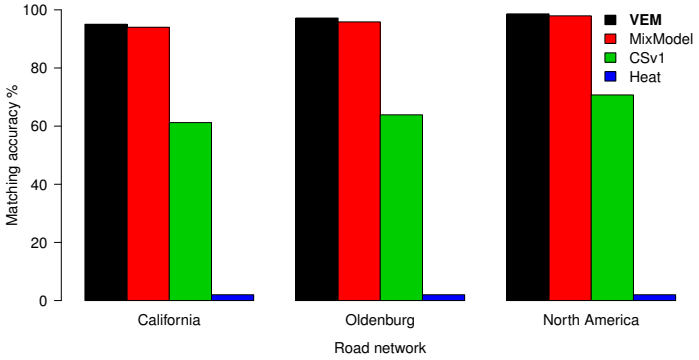


Figure 5.11: Common subgraph matching accuracy by using different road networks.

5.4.3 Scalability Study

In this section, we compare the scalability with respect to graph size for the four graph matching algorithms. We report the runtime for common subgraph matching from Section 5.4.2. All experiments were carried out on an Ubuntu 12.04 platform with an Intel Core i5 CPU with 8GB RAM. For matrix operations we used *Armadillo*, which is a C++ linear algebra library [107].

Figure 5.12 shows all three road networks with the number of vertices for each. As one can see, our proposed algorithm **VEM** scales well with respect to the graph

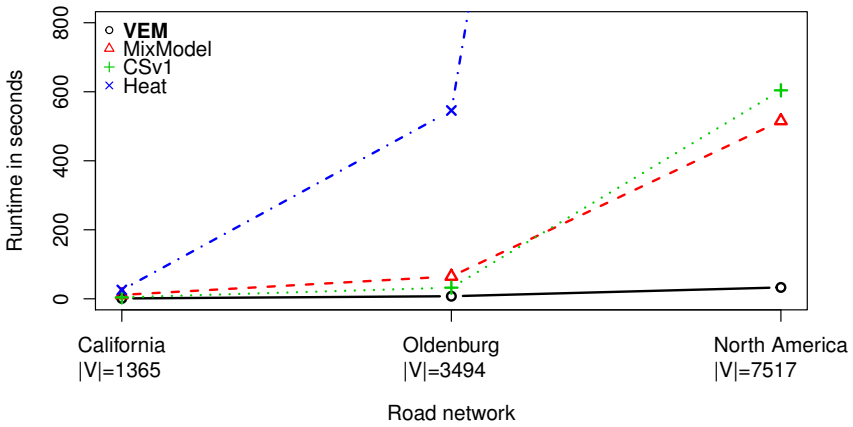


Figure 5.12: Scalability with respect to graph size reported as the runtime required to match different graphs.

size. It scales better than both **MixModel** and **CSv1** because the latter two algorithms use Munkres' algorithm, which runs in cubic time with respect to the graph size. The **Heat** algorithm requires the longest runtime. This algorithm uses matrix decomposition to compute the Eigenvectors for each of the graphs followed by SVD for their similarity matrix. However, matrix decomposition and SVD face scalability problems with dense and large matrices making **Heat** the slowest algorithm [107].

5.5 Summary and Discussion

In this chapter, we proposed a novel graph matching algorithm that has high scalability with respect to graph size and high matching accuracy. To improve scalability, we proposed to embed the vertices of different graphs into vector-spaces. This enables us to find the similarity between two vertices in constant time compared to a cubic runtime complexity in the case of using the cyclic string edit distance. Our embedding scheme is considered a special case of the Lipschitz embedding where each axis of the embedded space is spanned by the distances to a reference object. To embed the vertices of a graph into a vector space, initially, the graph is decomposed into a multi-set of vertex signatures such that each vertex signature represents the neighborhood of a vertex. Then, prototypes of vertex signatures are provided. The coordinate of a vertex in the embedded space is defined by the vector of distances between that vertex and the set of prototypes. To select a representative set of prototypes, we

discussed three different prototype selection methods. After embedding the vertices of different graphs into the vector space, we proposed a probabilistic graph matching algorithm that combines both the similarity in the embedded space with structural compatibility. Our algorithm starts with an initial match between the two graphs and iteratively improves and expands it. At each iteration, the similarity between two vertices is refined in a voting scheme. The vertices of the match that is computed in a previous iteration are used to estimate the similarity of any pair of vertices from the two graphs to be compared.

Using representative geometric graphs that are extracted from road networks, our algorithm outperforms related algorithms in terms of scalability and matching accuracy. However, our algorithm proposed in Chapter 4 outperforms the algorithm that is proposed in this chapter for subgraph matching.

All the previous chapters discuss the problem of matching geometric graphs in 2D space. However, a crucial task for many scientific applications is to search a graph database for similar graphs. Since computing the similarity between two graphs runs in cubic time complexity, as proposed in this chapter and Chapter 4, a linear search of the graph database takes too much time. As a result, filtering and verification techniques are used to prune non-similar graphs before solving the graph matching problem, which will be detailed in the following chapter.

Chapter 6

Geometric Graph Similarity Search

In the previous chapters, we discussed the matching problem for geometric graphs in 2D space. We proposed algorithms that approximate the optimal solution and run in cubic time complexity with respect to graph size. In this chapter, we turn our attention to the geometric graph similarity search problem, which formalizes the task of discovering all graphs in a database that are similar to a query graph. For example, in bioinformatics, the task is to search a database to find proteins that are similar to a newly discovered one; in image analysis, users search an image database for similar images or search a database for similar fingerprints.

6.1 Problem Definition

A well-known approach to estimate the similarity of graphs is by utilizing solutions to the graph matching problem. The more similar vertices the two graphs have the more similar the graphs are. The common subgraph concept adopts this approach such that the similarity between two graphs increases when the size of their maximum common subgraph increases [54]. Unfortunately, such a solution is based on the concept of exact graph matching, which cannot estimate the similarity between graphs that differ in structure and labeling information. As a result, *graph edit distance* is proposed to estimate the similarity between graphs in several scientific applications, such as computer vision, character recognition, and many more [47].

A typical task for many applications is to retrieve all graphs, from a graph database, that are highly similar to a query graph Q. Such a task can be classified into two types: 1) *k-nearest-neighbor queries* and 2) *range queries*. In this chapter, we are interested in **range queries with graph edit constraints**.

Definition 6.1. (Graph Range Query Problem) *Given a graph database* $D = \{G_1, G_2, \ldots, G_{|D|}\}$ *and a query graph* Q*, the range query problem is to find all graphs* $G_i \in D$ *such that the graph edit distance between* G_i *and* Q *is less than a threshold* τ*.*

Since computing the exact graph edit distance is an NP-hard problem [134], a sequential search of a graph database takes too much time. Even a graph matching heuristic, such as the one we proposed in the previous chapter, takes too much time when applied to all the graphs in a database. As a result, *filtering and verification* techniques are used to initially filter non-similar graphs and later verify the candidate similar graphs using graph edit distance. There are three criteria that should be guaranteed by such a filtering approach:

1. There are no *false negatives*. This means that there are no filtered graphs that satisfy the range threshold τ. To guarantee this condition, the filtering step utilizes a graph distance function $d(G, Q)_{lower}$ that is a *lower bound* to the graph edit distance $d(G, Q)$. The main idea is that if $d(G, Q)_{lower} > \tau$ then for sure $d(G, Q) > \tau$. As a result, graph G does not satisfy the range query threshold and can be safely filtered out.

2. The runtime complexity of the filter step is less than the complexity of the verification step. Otherwise, using the filter step becomes a bottleneck and not an advantage.

3. The higher the percentage of the pruned graphs the better the filtering approach. It is always true that the tighter the lower bound $d(G, Q)_{lower}$ to the graph edit distance $d(G, Q)$, the higher the percentage of pruned graphs.

Most of the pruning approaches with graph edit distance constraints suppose the vertices or/and the edges are labeled based on a discrete alphabet. Also, most of the lower bound distances are defined based on the number of edit operations that are used to make two graphs identical [53, 123]. In other words, a Dirac function of the labels of the vertices and edges is used to estimate the costs of their edit operations, where the distance is 1 if the labels of two vertices/edges are different and 0 otherwise. Unfortunately, these two assumptions are not valid in the case of solving range queries for geometric graphs. First, the vertices are assigned coordinates as real-valued labels and not discrete. Second, using the number of edit operations does not capture the spatial similarity between two graphs. As a result, the real-valued cost of the edit path must be used instead. Such requirements make it hard to formalize a lower

bound to the graph edit distance for geometric graphs, which is stated explicitly by Cheong *et al.* [26].

In this chapter, we study range queries for geometric graphs in 2D space. Since it is very hard to formalize a tight lower bound to the graph edit distance, we use another graph distance measure that has the flavor of a graph edit distance. Such a distance measure is built based on the similarity of the vertices of two geometric graphs and the Hungarian algorithm. To speed up the search for similar vertices to the vertices of the query graph, we propose a vertex-based index structure. For this, we adopt the vertex embedding technique proposed in Chapter 5. First, all the vertices of a graph database are represented as vectors of their distances to a set of prototypes. Then, the vectors are indexed by the well-known R-tree structure. We also propose three distance functions that are considered lower bounds to the geometric graph distance. The first one utilizes highly similar vertices, which are retrieved from the R-tree, to estimate the distance between different graphs. The second one is based on the concept of graph norm, which is defined based on the insertion costs of the vertices of a graph. The third uses a lower bound to the substitution cost of two vertices and the Hungarian algorithm to estimate the overall graph similarity.

The remainder of this chapter is organized as follows. Section 6.2 discusses related work. In Section 6.3, we propose our geometric graph indexing framework. In Section 6.4, we empirically evaluate our proposed solution. Section 6.5 summarizes the chapter.

6.2 Related Work

A crucial task for many graph-based applications is to retrieve relevant graphs from a graph database given a query graph. Since graph problems, such as graph isomorphism or frequent subgraph discovery, are very complex, a sequential search of a graph database takes too much time. Several graph indexing structures and pruning techniques that support graph query processing have been proposed, which can be classified into two main categories: *containment queries* and *similarity queries*.

A containment query searches for a graph G from a graph database such that there is a subgraph isomorphism between G and a query graph Q. Actually, most of the indexing structures proposed in the literature support containment queries. The main idea of such approaches is to index and prune non-similar graphs based on frequent or discriminative graph substructures. This includes path-based approaches such as GraphGrep [113], frequent subgraph-based approaches such as gIndex [132] and

FG-Index [25], frequent and discriminate subtrees-approaches such as TreePi [135], Swift-Index [111], and Tree+Δ [137], and coding-based approaches such as GCoding [142] and GString [60].

The second class of indexing structures is dedicated to support graph similarity search. Given a similarity threshold τ and a query graph Q, the user is interested in retrieving every graph G from a graph database such that the graph edit distance between Q and G is less than τ.

A naive approach to prune non-similar graphs is by using the difference in the number of vertices and edges [134], or the difference in the labels of the vertices and edges [131]. However, the pruning power of such naive approaches is very limited since the structure of the graphs is not considered. *Substructure-based* approaches are proposed to solve this problem. The main idea is to extract substructures as features from each graph. Then, a count-based lower bound to the graph edit distance is used to prune non-similar graphs. To retrieve features similar to the features of a query graph, different indexing approaches have been proposed based on the structure of the features. In the following, we categories such approaches based on the extracted graph feature.

- **(Tree-based)**: a **tree** feature approach has been introduced by Wang *et al.* [123]. Each graph is decomposed into a multi-set of k-adjacent trees (***k*-AT**), which is the breadth first search tree rooted at each vertex with hight k. The vertices at the same level of the tree are sorted based on the canonical order of their labels. The authors propose a lower bound to the graph edit distance based on the number of common k-adjacent trees. To index and organize the k-AT of all graphs, they introduce a k-AT lattice.

- **(Star-based)**: Wang *et al.* [124] propose a **star** feature approach called **SEGOS**. The star structure is defined by a vertex, its incident edges, and the set of direct neighboring vertices [134]. They give a lower bound to the graph edit distance based on the best assignments of stars using the Hungarian algorithm [134]. For indexing, they suggest a two level indexing structure. The first level maps a label to star structures, and the second level maps a star structure to graphs.

- **(Path-based)**: Zhao *et al.* propose **GSimJoin**, which is a **path** feature approach [131, 138]. They give a count-based lower bound to the graph edit distance based on the common paths of two graphs. To organize the paths extracted from the graph database, they use an inverted index that maps each

path to the graphs that contain it. Since some paths may exist in nearly every graph of the database, they suggest a prefix filtering technique to prune graphs that do not guarantee the range query threshold. To obtain a smaller candidate set, they choose rare labels as the prefixes of the paths. They also use a pruning technique based on the non-common paths of two graphs.

- **(Branch-based)**: recently, a branch approach has been presented [139], which is a star structure that does not include the direct neighboring vertices and only includes the incident edges. To efficiently search for similar branches, they propose an indexing structure called **branch-tree** where all leaves are graphs and non-leaves represent the information union of their child nodes. Such a concept is similar to the closure-tree concept [53] except that the internal nodes store branch unions instead of graph unions. They also give a lower bound to the graph edit distance by solving the assignment problem between the branches of two graphs.

There are two common assumptions made by the above indexing and pruning techniques. First, most of the lower bounds proposed are based on the number of edit operations between two graphs and do not consider the difference in the labeling information. Second, most of the indexing structures suppose a discrete label alphabet with a canonical order. Unfortunately, geometric graphs do not guarantee such assumptions. Spatial properties are real-valued numbers and are not discrete. On the other hand, the number of edit operations cannot estimate the spatial similarity between two graphs. As a result, most of the previous approaches are not applicable and cannot handle geometric graphs.

For geometric graphs, one can only find a very few indexing structures that support similarity search. Remember that geometric graph similarity should handle invariance under geometric transformation, as we already stated before. As a result, using a spatial index structure, such as a Quad-tree, cannot retrieve similar coordinates under geometric transformation.

To the best of our knowledge, *geometric hashing* is the only index structure that supports similarity between the coordinates of different vertices taking into consideration geometric transformations [125]. The basis of this approach is to create several local frames for the neighborhood of each vertex, which is defined again by that vertex and its direct neighbors. Then, the coordinates of the vertices in the neighborhood of a vertex are measured with respect to each local frame. After that, hashing is used to speed up the search for the local frame that best estimates the distance between two

vertices. Geometric hashing is efficient in the case of matching vertices that have a homogeneous transformation, i.e., a affine transformation. But, in the case of inexact matching, such an approach fails to correctly estimate the similarity of the vertices.

6.3 Geometric Graph Indexing

In this section, we propose our novel framework to tackle the range query problem for geometric graphs in 2D space. We follow a star-based approach utilizing the vertex signatures concept. We propose several pruning techniques that guarantee no false negatives and a recall of 100%. Our proposed framework consists of two layers. The first layer searches for vertices similar to the vertices of a query graph. For this, we propose a vertex-based indexing structure that allows to efficiently retrieve similar vertices. Based on the common similar vertices, the first layer generates the first set of candidate similar graphs, which is then passed to the second layer of our framework. The second layer consists of two filtering approaches that prune non-similar graphs before computing the geometric graph similarity. In the following, we outline our framework before detailing its components.

1. **First Layer**. The core of the first layer is an index structure that supports the search for vertices similar to the vertices of a query graph. Initially, all vertices of the graphs of in a database are embedded into a higher dimensional space using our previously proposed method (Chapter 5). Then, an R-tree is used to store and index all vertices using their vector-based representations. For querying, all vertices of the query graph are embedded into the same higher dimensional space. Then, the R-tree is used to retrieve all vertices that are similar, within a certain threshold, to the vertices of the query graph.

   Based on the set of candidate similar vertices, the first layer builds the first set of candidate similar graphs to be processed by the second layer. This can be done through an inverted index that links each vertex to its graph. The first layer estimates a lower bound to the geometric graph distance by utilizing 1) the set of common vertices, 2) the set of non- common vertices from the smaller graph, and 3) the set of non-common vertices from the larger graph.

2. **Second Layer**.

   Two pruning approaches are used by the second layer to filter non-similar graphs. The first approach employs the concept of *graph norm*, which is the

sum of the insertion costs of all the vertices of a graph. We prove later that the difference between the norms of two graphs is a lower bound to the distance between the two graphs. The second pruning approach is a tighter lower bound of the distance between geometric graphs. However, the computational complexity of such an approach is more than the complexity of the norm-based approach. To compute the second lower bound between two graphs G and Q, first, a distance matrix is created such that each entry $d_{i,j}$ represents a lower bound to the vertex edit distance between the two vertices $v_i \in G$ and $u_j \in Q$. In other words, each entry represents a lower bound to the substitution cost between two vertices from the two graphs. Then, the Hungarian algorithm is used to find the cost of the best assignment between the vertices of the two graphs. Such cost is eventually considered a lower bound to the distance between the two graphs.

Figure 6.1 outlines our geometric graph search framework. The two layers are highlighted inside the rectangles in bold. In the first layer, the inverted index is used to link each vertex to its graph id and its embedding coordinate. Notice that we maintain a global id among all the vertices from the graphs. Given a vertex v and its embedding, an R-tree is used to retrieve vertices highly similar to v. In the second layer, the two pruning approach are used to prune non-similar graphs to the query graph. Notice that the second layer also uses the inverted index to access the embedding of the vertices of different graphs.

6.3.1 Geometric Graph Distance Measure

In this section, we formalize our graph distance measure that is used to estimate the similarity between two geometric graph in 2D space. Since it is hard to formalize a lower bound to the geometric graph edit distance, we propose a distance measure that is highly related to the graph edit distance concept and easier to analyze. Several concepts that were introduced in Chapter 3 are going to be used to formalize our graph distance measure. Mainly, we are interested in the vertex edit distance, and our solution uses the polar distance-based edit operations (Section 3.3.2).

Definition 6.2. (Vertex Substitution) *Given two vertices v and u with their vertex signatures $S(v)$ and $S(u)$, respectively, the substitution cost between the two vertices $c(v \rightarrow u)$ is computed by the vertex edit distance between $S(v)$ and $S(u)$ using the polar coordinates edit operations.*

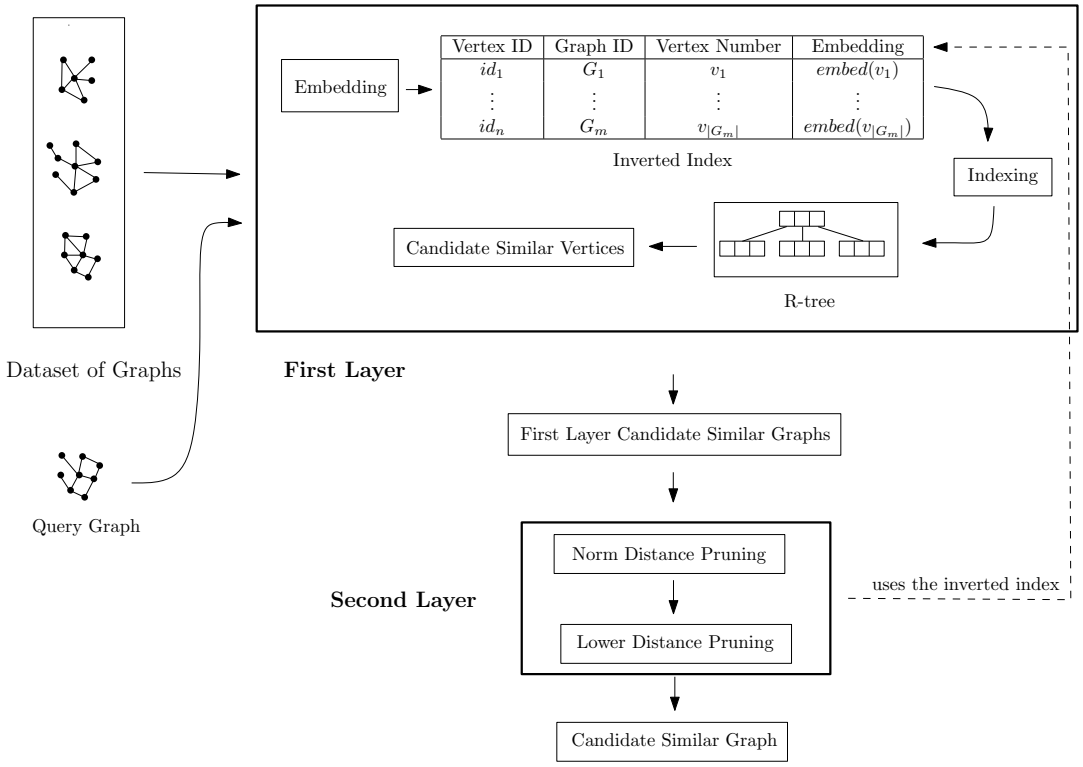


Figure 6.1: Proposed geometric graph search framework consisting of two layers. The first combines both an inverted index and an R-tree. The second layer has two pruning techniques to filter non-similar graphs.

Definition 6.3. (Vertex Insertion/Deletion) *Given a vertex v and its spatial feature $F_v = [f_1, f_2, \ldots, f_{deg(v)}]$, the insertion cost of vertex v $c(\varepsilon \rightarrow v)$ and the deletion cost $c(v \rightarrow \varepsilon)$ are computed as:*

$$c(\varepsilon \rightarrow v) = c(v \rightarrow \varepsilon) = \sum_{i}^{deg(v)} c(\lambda \rightarrow f_i)$$

such that $c(\lambda \rightarrow f_i)$ is the insertion cost of f_i as proposed in Equation 3.8.

Definition 6.4. (Vertex Distance Matrix) *Given two geometric graphs G and Q, $m = |G|$, $n = |Q|$, and $m \leq n$. The vertex distance matrix C is defined as:*

$$C = \begin{array}{c} \\ 1 \\ \\ m \\ m+1 \\ \vdots \\ n \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \dots & n \end{array} \\ \begin{pmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m,1} & c_{m,2} & \dots & c_{m,n} \\ c_{\varepsilon,1} & c_{\varepsilon,2} & \dots & c_{\varepsilon,n} \\ \vdots & \vdots & \vdots & \vdots \\ c_{\varepsilon,1} & c_{\varepsilon,2} & \dots & c_{\varepsilon,n} \end{pmatrix} \end{array}$$

where $c_{i,j}$ denotes the substitution cost $c(v_i \rightarrow u_j)$ of the two vertices $v_i \in G$ and $u_j \in Q$ and $c_{\varepsilon,j}$ denotes the insertion cost $c(\varepsilon \rightarrow u_j)$ of vertex $u_j \in Q$.

Since different graphs have different numbers of vertices, the vertex distance matrix is made a square matrix by padding it with the insertion costs of the vertices of the larger graph. This can be interpreted as if the smaller graph is padded with the null, i.e., a non-existent vertex, to make its size equal the larger one.

Definition 6.5. (Geometric Graph Distance) *Given two geometric graphs G and Q with their vertex distance matrix C, the distance between the two graphs $d(G, Q)$ equals the cost of solving the assignment problem using C.*

Normally, the solution of the assignment problem is represented by the bijective function $\phi : \{V(G), \varepsilon\} \rightarrow V(Q)$ that maps each vertex from G to a vertex from Q and maps the unmatched vertices from Q to the (null) non-existent vertex ε. Based on such a mapping function, the geometric graph distance $d(G, Q)$ is simply computed as:

$$d(G, Q) = \sum_{i=1}^{m} c(v_i \rightarrow \phi(v_i)) + \sum_{i=m+1}^{n} c(\varepsilon \rightarrow \phi(\varepsilon)) \tag{6.1}$$

Figure 6.2 shows an example of how we compute the distance between two geometric graphs. In the figure we see the vertex signatures of two graphs G and Q. Solving the assignment problem produces the match between the two graphs, which is represented by the dashed lines. In total there will be 4 vertex substitutions and one vertex insertion. If we suppose the cost of such edit operations the values on the arrows, then the distance between the two graphs is 50.

Our geometric graph distance (Definition 6.5) is a metric function. This is because a) the vertex edit distance, i.e., vertex substitution, b) the vertex insertion and deletion costs are metric functions, and c) the Hungarian algorithm computes the optimal solution to the assignment problem.

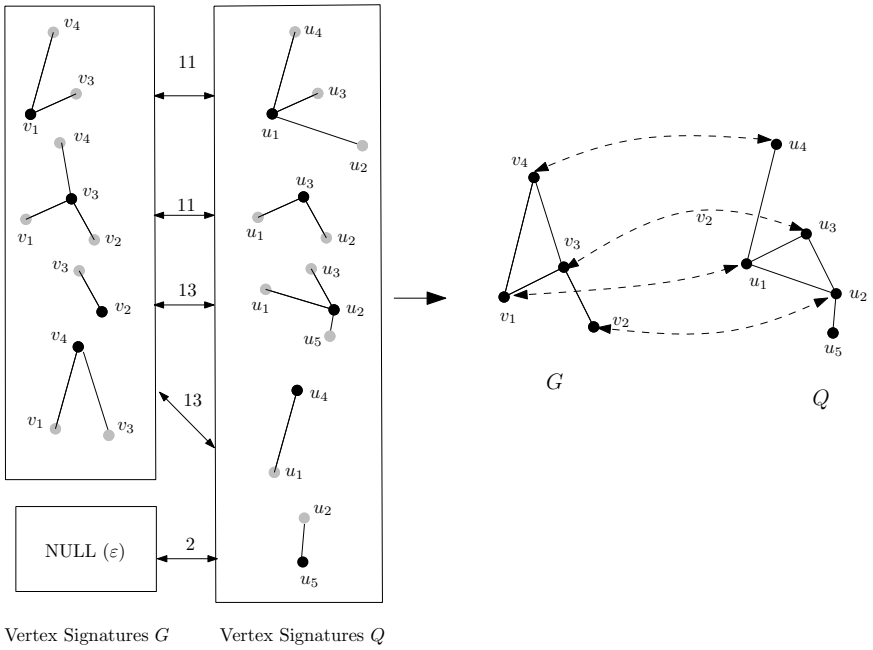


Figure 6.2: The geometric graph distance based on solving the assignment problem using the vertex distance matrix.

It is worth to notice that the distance function proposed in Definition 6.5 has been used to approximate the graph edit distance for non-geometric graphs. Zeng *et al.* prove that such a distance function can be used as a lower bound to the non-geometric graph edit distance with certain assumptions [134]. On the other hand, Justice and Hero prove that under different assumptions such a graph distance measure is an upper bound to the non-geometric graph edit distance [65]. In our case, it is worth to put more efforts and try to study the relation between our geometric graph distance and the exact solution to the geometric graph edit distance. In the following, we are going to define the range query problem based on the geometric graph distance measure given in Definition 6.5.

After introducing our geometric graph distance metric, in the following sections, we detail our framework that helps in solving the geometric graph range query. Section 6.3.2 outline the first layer of our framework. We formalize the vertex range query problem in Section 6.3.3 and propose our solution that combines the idea of vertex embedding and the well-known R-tree structure. Then, in Section 6.3.4, we discuss how to prune non-similar graphs based on the set of candidate similar vertices. Section 6.3.5 details the second layer of our framework with its two pruning approaches.

6.3.2 Indexing Structure

The main task of the first layer is to use the common similar vertices between a query graph and the graphs in a database to generate a set of candidate similar graphs. Notice that the similarity between two vertices is estimated by the similarity of their vertex signatures. To fulfill such a task, two questions should be answered:

1. How to efficiently retrieve vertices similar to the vertices of a query graph, which is what we call the *vertex range query problem* and will be detailed in Section 6.3.3.

2. Given a set of common similar vertices between two graphs, how to estimate the overall graph similarity, taking into consideration the lack of information about the similarity of the non-common vertices. We will discuss this problem in Section 6.3.4.

6.3.3 Vertex Range Query Problem

Since our framework follows a substructure-based approach, i.e., vertex signatures in our case, an efficient indexing structure is needed to retrieve vertex signatures similar to the vertex signatures of the query graph. For this, we formalize the vertex range query problem as follows:

Definition 6.6. ***(Vertex Range Query Problem)*** *Given a database* $D_s = \{S(v_1), S(v_2), \ldots, S(v_{|D_s|})\}$ *that contains vertex signatures from all the graphs of a graph database* D*, and a query vertex signature* $S(v_q)$*, the vertex range query problem is to find all vertex signatures* $S(v_i) \in D_s$ *such that the vertex edit distance between* $S(v_i)$ *and* $S(v_q)$ *is less than a threshold* μ*.*

Our framework does not explicitly index the database of vertex signatures D_s. Instead, we convert such an indexing problem to the indexing of higher dimensional data by utilizing our vertex embedding scheme (Chapter 5). The first layer of our framework first embeds all the vertex signatures in D_s into a higher dimensional space using a pre-defined set of prototypes. To select the prototypes, we follow the spanning prototype selection method as defined in Algorithms 5.2 (page 115). It initializes the prototype set by the median vertex signature from a training dataset. Then, the vertex signature that has the farthest distance from the already selected prototypes is selected. We also add the null prototype to the prototypes set, i.e., the null non-existent vertex. The result of measuring the distance between a vertex signature and

the null prototype equals the cost of inserting that vertex, which helps to efficiently computed our lower bounds as will be detailed later.

After converting all the vertex signatures to the vector-based representation, an R-tree is used to index and store all the vectors. For a query graph, first, all its vertices are embedded into higher dimensional space. Then, given a vertex range threshold μ, the R-tree is used to efficiently retrieve vertices similar to a query vertex.

A critical issue is definition of the value of the threshold μ. Suppose a query graph Q and a graph G such that $|Q| \leq |G|$. If the distance between the two graphs is within τ, then there are at least two vertices from the two graphs $v \in G$ and $u \in Q$ such that their substitution cost $c(v \rightarrow u) \leq \frac{\tau}{|Q|}$. On the other side, if $\forall v_i, u_j \; c(v_i \rightarrow u_j) > \frac{\tau}{|Q|}$, then the two graphs are not within a τ distance.

Unfortunately, the previous observation is only valid in the case where $|Q| \leq |G|$. On the other side, i.e., $|Q| > |G|$, using $\mu = \frac{\tau}{|Q|}$ later leads to false negatives in terms of similar graphs. In other words, there will be a graph G such that its distance with respect to the query graphs is $d(Q, G) \leq \tau$ and at the same time $\forall v_i \in G, c(v_i \rightarrow \phi(v_i)) > \mu$. We give an example in Figure 6.2 to clarify this point. Suppose graph Q is the query graph and $\tau = 50$. If we consider $\mu = 10$, then all the vertices of graph G will be filtered by the R-tree. However, the distance between the two graphs is 50 and satisfies the range threshold τ.

Actually, the cause of this problem is that we do not know the size of graph G in advance, other than that the problem is solved by dividing τ by the minimum of both graph sizes. To overcome this problem, we utilize the insertion costs of the vertices of the query graph to guarantee a value of μ without any false negatives. The intuition is that if $|Q| = |G| + 1$, then there are G substitutions and exactly one insertion. If we assume the insertion cost x s.t. $x < \frac{\tau}{|Q|}$, then, it is possible that $|G|-1$ substitutions have a total cost less than or equal $(|G| - 1)\frac{\tau}{|Q|}$, and one substitution has a cost less than or equal $\frac{\tau}{|Q|} + (\frac{\tau}{|Q|} - x)$. In this case, the geometric graph distance is $x + (|G| - 1)\frac{\tau}{|Q|} + \frac{\tau}{|Q|} + (\frac{\tau}{|Q|} - x)$, which is less than or equal τ. As a result, the upper bound to the substitution cost becomes $\frac{\tau}{|Q|} + (\frac{\tau}{|Q|} - x)$. Since we do not know the size of graph G, we utilize the upper bound of the number of insertions, which is $|Q| - 1$. To define μ, we consider the minimum $|Q| - 1$ insertions of Q as follows:

$$\mu = \frac{\tau}{|Q|} + \sum_{\substack{j=1 \\ c(\varepsilon \rightarrow u_j) < \frac{\tau}{|Q|}}}^{|Q|-1} \left(\frac{\tau}{|Q|} - c(\varepsilon \rightarrow u_j) \right) \tag{6.2}$$

where $c(\varepsilon \rightarrow u_j)$ is the insertion cost of vertex $u_j \in Q$. Notice that the summation includes the cost of an insertion $c(\varepsilon \rightarrow u_j)$ only if it is less than $\frac{\tau}{|Q|}$. On the other side, an insert edit operation does not affect the upper bound of μ if its cost is less than or equal $\frac{\tau}{|Q|}$. Based on the above equation, the value of μ is considered as the upper bound to the substitution cost between any two vertices from G and Q such that $d(G, Q) \leq \tau$. As a result, we guarantee no false negatives.

6.3.4 From Similar Vertices to Similar Graphs

The first layer uses the set of similar vertices to generate a candidate set of similar graphs. Notice that the set of similar vertices contains vertices from many graphs. For this, the first layer uses an inverted index that links each vertex to its graph id and its embedding vector-based representation. The first layer uses the inverted index to group similar vertices that belong to the same graph. Given the vertices of a query graphs Q and their most similar vertices from another graph G, how do we know if the two graphs have a distance within the threshold τ?

To solve this problem, we divide the vertices of the two graphs G and Q into three overlapping groups of vertices to define three different distances between them: $C_{insertion}$, $C_{similar}$, and $C_{unmatched}$, as shown in Figure 6.3. We use such three distances to create the first layer graph distance function, which is a lower bound to the geometric graph distance and is used to prune non-similar graphs.

Let $m = |G|$ and $n = |Q|$. Without loss of generality, suppose $m \leq n$. Let G_s be the set of vertices from G that have similar vertices in Q, which is represented by the vertices v_1 and v_3 in Figure 6.3. Also, let Q_s be the set of vertices from Q that have similar vertices in G, which is represented by the vertices u_1, u_2, and u_3 in Figure 6.3. Notice that a vertex from G_s may be similar to more than one vertex from Q_s, and vise versa. Following these assumptions, we define the three graph distances as follows:

1. $C_{insertion}$. Let the set $S = \{s_1, \ldots, s_n\}$ contain the insertion costs of all the vertices of Q in increasing order, then $C_{insertion}$ is the sum of the insertion costs of the vertices of Q that do not have a corresponding vertex in G:

$$C_{insertion} = \sum_{i=1}^{n-m} s_i \tag{6.3}$$

Notice that when computing the graph distance $d(G, Q)$, there will be $|G|$ vertex substitutions and $n-m$ vertex insertions. For the graphs in Figure 6.3, $C_{insertion}$

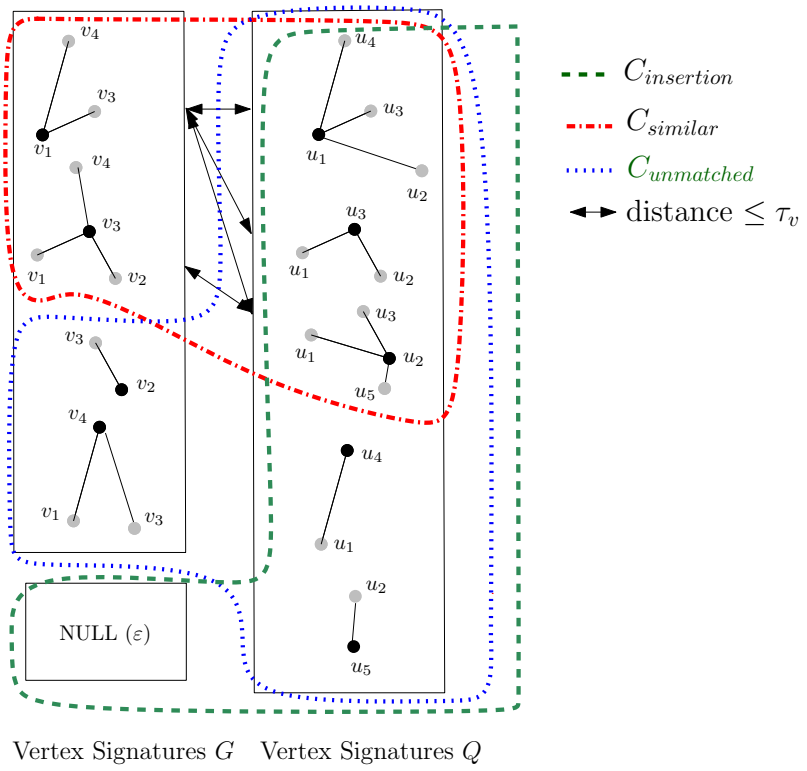


Figure 6.3: Three groups of the vertices of two graphs are used to compute the first layer graph distance.

equals the minimum insertion cost of the vertices of Q. This is because the difference between the sizes of both graphs is 1. The insertion cost $C_{insertion}$, as defined above, is a lower bound to the insertion cost computed by the graph distance $d(G, Q)$. This is because we took the smallest $n - m$ values of them. However, when computing $d(G, G)$, the Hungarian algorithm may select any of the $n - m$ insertions from the set S.

2. $C_{similar}$. Without loss of generality, suppose $|G_s| \leq |Q_s|$, then $C_{similar}$ is computed as the sum of the minimum distances between the vertices of G_s and Q_s.

$$C_{similar} = \sum_{v \in G_s} d_{min}(v, Q_s) \tag{6.4}$$

where $d_{min}(v, Q_s)$ is the minimum distance between v and the vertices of Q_s as retrieved from the R-tree. Notice that such a distance is computed based on the Euclidean distance of the embedding of two vertices, which is a lower bound

to their substitution cost as proved in Equation 5.4. As a result, $C_{similar}$ is a lower bound to the sum of substitution cost between the vertices of G_s and Q_s.

3. $C_{unmatched}$. Let $h = \min\{|G|, |Q|\} - \min\{|G_s|, |Q_s|\}$ define the number of vertices from the smaller graph that have no similar vertices from the larger one, which is represented by the vertices v_2 and v_4 in Figure 6.3. Then we define $C_{unmatched}$ as:

$$C_{unmatched} = h * \mu \tag{6.5}$$

Since we use the vertex similarity threshold μ to filter non-similar vertices, any pair of vertices that are not added to the candidate of similar vertices must have an Euclidean distance greater than μ. As a result, any filtered pair of vertices has a substitution cost greater than μ. This is because the Euclidean distance between the embedding of two vertices is a lower bound to their substitution cost. This makes $C_{unmatched}$ a lower bound to the sum of the exact substitution costs of the same h vertices.

Definition 6.7. ***(First Layer Graph Distance)*** *The first layer graph distance $d_{fl}(G, Q)$ is defined as:*

$$d_{fl}(G, Q) = C_{insertion} + C_{similar} + C_{unmatched} \tag{6.6}$$

Lemma 6.1. *The first layer graph distance $d_{fl}(G, Q)$ between the two graphs G and Q is a lower bound to the geometric graph distance $d(G, Q)$.*

Proof. $C_{insertion}$ is a lower bound to the sum of insertion costs of the vertices of the larger graph as computed by the geometric graph distance $d(G, Q)$. $C_{similar} + C_{unmatched}$ is a lower bound to the sum of substitution costs of the vertices of the smaller graph. As a result, $d_{fl}(G, Q) \leq d(G, Q)$. □

By using the first layer graph distance, a set of candidate similar graphs is determined from the set of candidate similar vertices. Then, such candidate graphs are passed to the second layer for further processing.

6.3.5 Lower Bound Distances

The second layer of our framework takes a set of candidate similar graphs and the embedding of their vertices from the first layer. It uses two lower bounds to the

geometric graph distance measure to prune non-similar graphs. The first one is not tight and runs in linear time complexity with respect to the graph size. The second lower bound is a tighter one and runs in cubic time complexity.

Definition 6.8. ***(Graph Norm)*** *The norm of a graph G, $m = |G|$, is defined as:*

$$\parallel G \parallel = \sum_{i}^{m} c(\varepsilon \rightarrow v_i)$$

where $c(\varepsilon \rightarrow v_i)$ is the insertion cost of vertex $v_i \in G$ based on Definition 6.3.

The complexity of computing the insertion cost of a vertex v_i, i.e., $c(\varepsilon \rightarrow v_i)$, is linear with respect to the degree of v_i. As a result, the complexity of computing the norm of a graph is linear with respect to its size.

Definition 6.9. ***(Graph Norm Distance)*** *Given two graphs G and Q, their norm distance is defined as:*

$$d_{no}(G, Q) = | \parallel G \parallel - \parallel Q \parallel | \tag{6.7}$$

The norm distance between two graphs is defined as the Manhattan distance between their norms. It is obvious from the above definition that the complexity of computing such a distance is linear with respect to the graph size.

Lemma 6.2. *The graph norm distance $d_{no}(G, Q)$ is a lower bound to the geometric graph distance $d(G, Q)$.*

Proof. Given two geometric graphs G and Q, $m = |G|$, $n = |Q|$, $m \leq n$. The geometric graph distance between G and Q as a result of solving the assignment problem can be written as:

$$d(G, Q) = \sum_{i=1}^{m} c(v_i \rightarrow \phi(v_i)) + \sum_{i=m+1}^{n} c(\varepsilon \rightarrow \phi(\varepsilon))$$

For simplicity, we define $v_i^{'} := \phi(v_i)$, and $c(v_i, v_i^{'}) := c(v_i \rightarrow \phi(v_i))$. Using this, $d(G, Q)$ is written as:

$$d(G, Q) = \sum_{i=1}^{m} c(v_i, v_i^{'}) + \sum_{i=m+1}^{n} c(\varepsilon, \varepsilon^{'})$$

From the triangle inequality, we have $|c(\varepsilon, v) - c(\varepsilon, u)| \leq c(v, u)$ for any two vertices v and u. As a result, $d(G, Q)$ is rewritten as:

$$d(G,Q) \geq \sum_{i=1}^{m} |c(\varepsilon, v_i) - c(\varepsilon, v_i^{'})| + \sum_{i=m+1}^{n} c(\varepsilon, \varepsilon^{'})$$

Notice that for a finite set of real numbers, $\sum_i |r_i| \geq |\sum_i r_i|$ holds. Utilizing this, we write the above equation as:

$$d(G,Q) \geq \left| \sum_{i=1}^{m} c(\varepsilon, v_i) - c(\varepsilon, v_i^{'}) \right| + \sum_{i=m+1}^{n} c(\varepsilon, \varepsilon^{'})$$

Since the insertion cost is positive, i.e., $c(\varepsilon, \varepsilon^{'}) \geq 0$, $\sum_{i=m+1}^{n} c(\varepsilon, \varepsilon^{'}) = \left| \sum_{i=m+1}^{n} c(\varepsilon, \varepsilon^{'}) \right|$. Also, for any two numbers a and b, $|a|+|b| \geq |a+b|$. By using these two observations, $d(G,Q)$ is rewritten as :

$$\begin{aligned} d(G,Q) \geq & \left| \sum_{i=1}^{m} c(\varepsilon, v_i) - c(\varepsilon, v_i^{'}) + \sum_{i=m+1}^{n} c(\varepsilon, \varepsilon^{'}) \right| \\ \geq & \left| \sum_{i=1}^{m} c(\varepsilon, v_i) - \sum_{i=1}^{m} c(\varepsilon, v_i^{'}) + \sum_{i=m+1}^{n} c(\varepsilon, \varepsilon^{'}) \right| \end{aligned}$$

Since $c(\varepsilon, \varepsilon^{'}) \geq 0$, $\sum_{i=m+1}^{n} c(\varepsilon, \varepsilon^{'}) \geq -\sum_{i=m+1}^{n} c(\varepsilon, \varepsilon^{'})$. Using this, we rewrite the above equation as:

$$\begin{aligned} d(G,Q) \geq & \left| \sum_{i=1}^{m} c(\varepsilon, v_i) - \sum_{i=1}^{m} c(\varepsilon, v_i^{'}) - \sum_{i=m+1}^{n} c(\varepsilon, \varepsilon^{'}) \right| \\ \geq & \left| \sum_{i=1}^{m} c(\varepsilon, v_i) - \left(\sum_{i=1}^{m} c(\varepsilon, v_i^{'}) + \sum_{i=m+1}^{n} c(\varepsilon, \varepsilon^{'}) \right) \right| \\ \geq & \left| \sum_{i=1}^{m} c(\varepsilon, v_i) - \sum_{j=1}^{n} c(\varepsilon, u_j) \right| \\ \geq & | \parallel G \parallel - \parallel Q \parallel | \\ \geq & d_{no}(G,Q) \end{aligned}$$

□

The norm distance lower bound is used to prune non-similar graphs. It runs in linear time complexity with respect to graph size. In the following, we first define a lower bound to the vertex substitution cost. Then, we use such a lower bound to propose another tighter lower bound to the geometric distance measure that runs in

cubic time complexity. Notice that in our framework we use different graph pruning approaches. We apply them consecutively following their runtime complexity, i.e., in increase order. So even though the last approach uses the tightest lower bound distance, it has the highest runtime complexity. Using such a pipeline of approaches reduces the candidate set that is processed by the tightest lower bound and speeds up query processing.

Definition 6.10. ***(Vertex Substitution Lower Bound)*** *Given two vertices v and u, a lower bound to their substitution cost, i.e., $c(v_i \rightarrow u_j)$, is defined as:*

$$lb_{v_i,u_j} = \max \left\{ \frac{\parallel \varphi(v_i) - \varphi(u_j) \parallel}{\sqrt{k}}, |c(\varepsilon \rightarrow v_i) - c(\varepsilon \rightarrow u_j)| \right\} \tag{6.8}$$

where $\varphi : \{V(G), V(Q)\} \rightarrow \mathbb{R}^k$ is the embedding function of a vertex into a higher dimensional space using our embedding scheme in Chapter 5. $\parallel \varphi(v_i) - \varphi(u_j) \parallel$ is the Euclidean distance between the embeddings of two vertices v_i and u_j, k is the dimension of the embedding space, $|c(\varepsilon \rightarrow v_i) - c(\varepsilon \rightarrow u_j)|$ is the Manhattan distance between the insertion costs of two vertices v_i and u_j.

Notice that we have two lower bounds to the substitution cost between two vertices. The first one is $|c(\varepsilon \rightarrow v_i) - c(\varepsilon \rightarrow u_j)|$, which is based on the insertion cost of the two vertices. The second lower bound uses our embedding scheme. After the embedding of the vertices of two graphs into a higher dimensional space using the same set of prototypes, we proved in Equation 5.4 that $\frac{\|\varphi(v_i)-\varphi(u_j)\|}{\sqrt{k}}$ is a lower bound to the substitution cost. To have a tighter lower bound to the vertex substitution cost, we take the maximum of both vertex lower bounds. Notice that computing the substitution cost between two vertices runs in cubic time complexity with respect to the vertex degree. However, both vertex lower bounds are computed in linear time complexity with respect to the vertex degree.

Definition 6.11. ***(Lower Bound Vertex Distance Matrix)*** *Given two geometric graphs G and Q, $m = |G|$ with $n = |Q|$ and $m \leq n$, the lower bound vertex distance matrix LB is defined as:*

$$LB = \begin{array}{c} \\ 1 \\ \\ m \\ m+1 \\ \vdots \\ n \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \dots & n \end{array} \\ \begin{pmatrix} lb_{1,1} & lb_{1,2} & \dots & lb_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ lb_{m,1} & lb_{m,2} & \dots & lb_{m,n} \\ c_{\varepsilon,1} & c_{\varepsilon,2} & \dots & c_{\varepsilon,n} \\ \vdots & \vdots & \vdots & \vdots \\ c_{\varepsilon,1} & c_{\varepsilon,2} & \dots & c_{\varepsilon,n} \end{pmatrix} \end{array}$$

where $lb_{i,j}$ is the lower bound to the substitution cost between the two vertices $v_i \in G$ and $u_j \in Q$, and $c_{\varepsilon,j}$ denotes the insertion cost $c(\varepsilon \rightarrow u_j)$ of vertex $u_j \in Q$.

Using such a distance matrix, in the following, we define the second lower bound to the geometric graph distance.

Definition 6.12. ***(Graph Lower Mapping Distance)*** *Given two graphs G and Q with their distance matrix LB, the distance $d_{lm}(G, Q)$ between the two graphs equals the cost of solving the assignment problem using LB.*

Notice that the way we compute the graph lower mapping distance is similar to the way we compute the geometric graph distance. The only difference is that instead of using the exact vertex substitution cost we use its lower bound.

Lemma 6.3. *The graph lower mapping distance $d_{lm}(G, Q)$ between the two graphs G and G is a lower bound to the geometric graph distance $d(G, Q)$.*

Proof. Given the vertex distance matrix $C \in \mathbb{R}^{n \times n}$, the solution of the assignment problem is the permutation matrix $P \in \{0, 1\}^{n \times n}$ that minimizes the following objective function:

$$\min_{P \in \mathcal{P}} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\ p_{ij} \tag{6.9}$$

where $\mathcal{P}$ is the space of all permutation matrices of size $n \times n$ subject to the following conditions:

$$\begin{gathered} \forall i \in \{1, \dots, n\},\ \sum_{j=1}^{n} p_{ij} = 1 \\ \forall j \in \{1, \dots, n\},\ \sum_{i=1}^{n} p_{ij} = 1 \\ \forall i \in \{1, \dots, n\}, j \in \{1, \dots, n\},\ p_{ij} \in \{0, 1\} \end{gathered} \tag{6.10}$$

Let the permutation matrix P^C be the solution to the assignment problem using the distance matrix C and P^{LB} be the solution to the assignment problem using the lower bound vertex distance matrix LB. Then, the distance between two graphs $d(G, Q)$ is written as

$$d(G, Q) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\ p_{ij}^{C} \tag{6.11}$$

Since $\forall i, j \in \{1, \ldots, n\}, lb_{ij} \leq c_{ij}$, we rewrite $d(G, Q)$ as:

$$d(G, Q) \geq \sum_{i=1}^{n} \sum_{j=1}^{n} lb_{ij}\ p_{ij}^{C} \tag{6.12}$$

The solution to the assignment problem is the permutation matrix that minimizes the objective function, as a result

$$\forall P \in \mathcal{P}, \left(\sum_{i=1}^{n} \sum_{j=1}^{n} lb_{ij}\ p_{ij}^{LB} \right) \leq \left(\sum_{i=1}^{n} \sum_{j=1}^{n} lb_{ij}\ p_{ij} \right) \tag{6.13}$$

which means that

$$\left(\sum_{i=1}^{n} \sum_{j=1}^{n} lb_{ij}\ p_{ij}^{LB} \right) \leq \left(\sum_{i=1}^{n} \sum_{j=1}^{n} lb_{ij}\ p_{ij}^{C} \right) \tag{6.14}$$

Using this inequality, we rewrite $d(G, Q)$ as

$$\begin{aligned} d(G, Q) &\geq \sum_{i=1}^{n} \sum_{j=1}^{n} lb_{ij}\ p_{ij}^{C} \\ d(G, Q) &\geq \sum_{i=1}^{n} \sum_{j=1}^{n} lb_{ij}\ p_{ij}^{LB} \\ d(G, Q) &\geq d_{lm}(G, Q) \end{aligned} \tag{6.15}$$

□

To summarize, given a query graph Q and a graph database, our framework first finds vertices from the graph database that are highly similar to the vertices of the query graph. Then, it uses three lower bound distances to the geometric graph distance to filter out non-similar graphs. The first one is used by the first layer and uses the set of candidate similar vertices to generate a set of candidate similar graphs. Then, the other two lower bounds are used by the second layer, which are the graph

norm distance and the graph lower mapping distance. A graph that is not filtered by any of these approaches is considered part of the final set of candidate similar graphs and then verified using our geometric graph distance, i.e., Definition 6.5.

6.4 Experimental Evaluation

In this section, our proposed framework to tackle the search problem for 2D geometric graphs is empirically evaluated. We evaluate our framework with respect to the following criteria:

1. Pruning performance, which is measured by the average number of pruned graphs over the total number of graphs in the database.

2. First layer vertex pruning performance, which is measured by the average number of pruned vertices, given a query vertex, over the total number of vertices indexed by the R-tree.

3. The pruning performance of each of the proposed lower bound distances. This includes $d_{fl}(G, Q)$, which is used by the first layer, $d_{no}(G, Q)$ and $d_{lm}(G, Q)$, which are used in sequence by the second layer. For this, the pruning performance of a lower bound distance is measured by the average number of pruned graphs over the total number of graphs processed by that distance.

4. Time and space used to build our indexing framework as the size of the graph database increases.

5. The response time of our framework. It is measured by the number of seconds needed to retrieve the set of relevant similar graphs that satisfy the range query threshold τ. This includes both the filtering time in addition to the verification time. We also report the number of seconds each layer and each pruning approach takes for a given query.

For our evaluations, we use three different datasets that come from a variety of applications domains. This includes image processing, chemoinformatic, and character recognition. We show some statistics of our datasets in Table 6.1 and detail each of them in the following.

1. COIL-100 image dataset has 7200 images for 100 different objects [87]. Each object creates a set of images by rotating the camera around that object in 3D.

Dataset	COIL-100	Chinese	AIDS
Type	Images	Characters	Chemical Compounds
Database size	6200	7820	42,234
Query set size	1000	1564	1000
Number of vertices in the R-tree	133,224	156,684	1,074,868
Average query size	21	31	33

Table 6.1: Statistics of the datasets being used to evaluate our indexing framework.

In other words, the dataset has images of objects taken from different angle views. Notice that only a subset of this dataset has been used in the previous chapters. For this test, we select randomly 1000 images as a query set, 10 images for each object. The remaining images create a database of 6200 graphs.

2. The Chinese character dataset contains a total of 9384 characters that belongs to 6 different fonts, i.e., 1564 characters from each font [1]. A query data set of 1564 graphs is extracted from the Dotum Korean font. The remaining five fonts build a dataset of 7820 graphs.

3. DTP AIDS Antiviral Screen chemical compound dataset [42]. It consists of 42,234 chemical compounds. In addition to the coordinates of the atoms, labeling information is assigned to the atoms indicating their types. Such a dataset is considered the standard benchmark for the non-geometric graph indexing approaches [25, 53, 123, 124, 132, 135, 138, 139]. We select randomly 1000 queries such that the query size is at least 24 vertices. This number is the upper limit for the query size used by the non-geometric graph indexing approaches. For our experiments, we consider only the spatial properties of the graphs and ignore labeling information.

For all three datasets, to define the range threshold τ, we compute the exact distance between every query and all the graphs in a database. Then, we assign τ for a query Q to its minimum distance to all the graphs in the database. By this, the graph G from the graph database such that $d(G, Q) \leq \tau$ is the first nearest neighbor graph to Q.

6.4.1 Index Construction

In this section, we elaborate on the performance of creating our indexing structure. We measure this by the number of seconds needed to construct the index structure

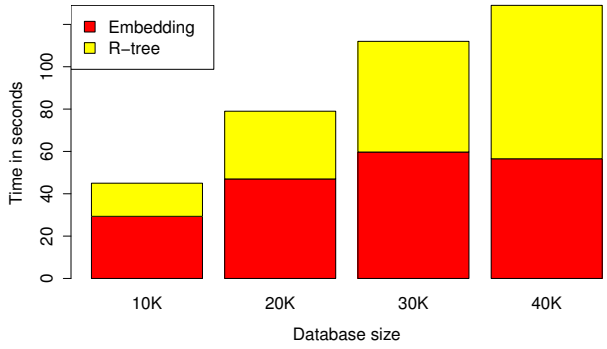


Figure 6.4: Number of seconds used to construct our proposed index structure. "Embedding" refers to the time spent to embed the vertices in a higher dimensional space. "R-tree" is the time spent to construct the R-tree.

and its size in megabytes (MB). Remember that our indexing structure includes the embedding of the vertices in a higher dimensional space, constructing an R-tree, and constructing the inverted index. Since the time and space that are needed to construct the inverted index are marginal compared to the other two steps, we remove it from the figures. For this test, we use the AIDS dataset to create four different graph databases with sizes of 10K, 20K, 30K, and 40K graphs.

Figure 6.4 shows the time needed to construct our indexing structure as the database size increases. Since building the indexing structure consists of two steps, embedding the vertices and constructing the R-tree, we report the time consumed by each of them separately. The total time used to construct the index structure is the sum of the times consumed by both of them. One can see from the figure that the construction time increases linearly as the database size increases. This also applies to both the embedding step and the R-tree construction step. From our experiments, we see that each of the two steps consumes the same number of seconds. In addition to this, our experiments show a linear relationship between the increase of the database size and the increase of the index size.

6.4.2 Pruning Performance

In this section, we test the pruning performance of our indexing framework. The smaller the size of similar candidate graphs the more pruning performance we have. Figure 6.5 shows the pruning performance of our proposed solution. One can see that on average 94% of the graphs are pruned by our framework. The figure also shows the

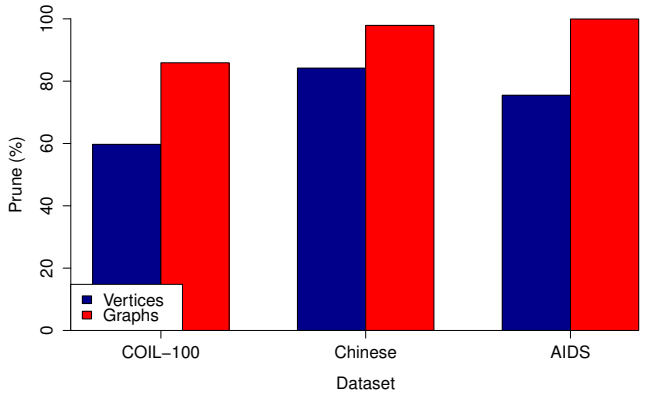


Figure 6.5: Pruning performance for different datasets. "Vertices" means the number of vertices pruned by the R-tree. "Graphs" indicates the number of graphs pruned by our framework.

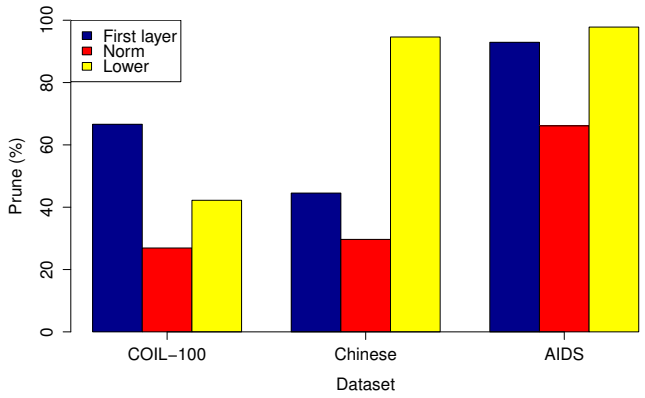


Figure 6.6: The pruning performance for different layers and lower bound distances. "Norm" denotes the norm distance $d_{no}(G, Q)$. "Lower" denotes the lower mapping distance $d_{lm}(G, Q)$.

percentage of the pruned vertices by the R-tree. We notice that on average more than 70% of the vertices indexed by the R-tree are pruned. This percentage is less than the percentage of pruned graphs because we only use one lower bound distance to the vertex substation cost to prune non-similar vertices. However, we use three different lower bounds to prune non-similar graphs. Also, the more frequent vertex signatures the graph database has, the less vertex pruning performance we get. An example for this observation is the result of the AIDS dataset, which has a high percentage of frequent vertex signatures.

We also show the pruning performance for each of the proposed lower bound distances in Figure 6.6. It is true that the overall pruning performance of our framework is more than 94%. But we want to know which layer and pruning approach of our framework performs the best. The pruning performance of the first layer is a result of first pruning non-similar vertices and then uses the lower bound $d_{fl}(G, Q)$ between the query graph and candidate similar graphs.

From the figure, we see that in general the best pruning performance is for the lower mapping distance. This is because it is the tightest lower bound to the geometric graph distance. On the other side, the least pruning performance is for the norm lower bound distance. This is because only the vertex insertion cost is used to estimate such a distance, where neither the difference in the graphs' sizes nor the substitution costs of the individual vertices are considered.

We notice that our systems performs better for the AIDS dataset compared to COIL-100 and Chinese datasets. This is because the edges in a chemical compound have nearly the same length. As a result, the vertex degree is some how encoded by the cost of vertex insertion, which is the sum of the lengths of the edges incident to that vertex. However, graphs in the COIL-100 and Chinese datasets have edges with high variation in their lengths. As a result, the insertion cost of two vertices maybe similar even though their degrees are highly different. This happens when a vertex has a low degree and edges with high lengths and the other vertex has many edges and small edge length. As a result, the difference between their insertion costs is low even though they are highly different.

We are also interested in measuring the pruning performance of our framework as the range threshold τ increases. For this test, we use the Chinese characters dataset. We used 5 different variations of the range threshold such that each one is defined by increasing τ by a certain amount starting from $0.2 \times \tau$ till $1.0 \times \tau$. For our analysis, we average the size of the candidate similar graphs for different queries and measure the overall pruning performance.

Figure 6.7 shows the pruning performance of each component of our framework as the search threshold increases. This includes the first layer, the norm distance, and the lower mapping distance. All are measured with respect to the percentage of pruned graphs, as discussed previously. In the figure, +0X means that the threshold is the minimum distance between a query graph and the graphs of the database. To increase the threshold, we add to τ a percentage starting from $0.2 \times \tau$, which is indicated by +0.2X in the figure, till $1.0 \times \tau$, which is indicated by +1.0X in the figure. In other words, the last value means twice the minimum distance between

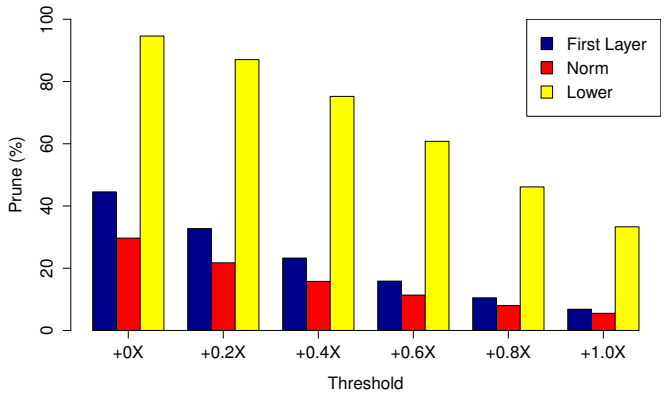


Figure 6.7: The pruning performance of the different components as the range threshold increases.

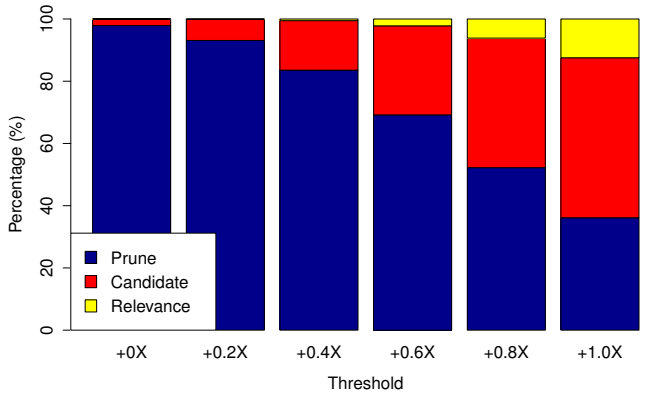


Figure 6.8: The percentage of the pruned graphs, candidate set, and relevant graphs to the query graph as the range query threshold increases τ.

a query graph and the graphs of the database. Our results show that the lower mapping distance has the highest pruning percentage with an average of 66%. The norm distance prunes on average 15% of the set of candidate similar graphs produced by the first layer. The first layer of our framework prunes on average 22% of the graphs in the database.

Figure 6.8 compares the percentage of the pruned graphs to both the candidate similar graphs retrieved by our framework and the percentage of the graphs relevant to the query graph. One can see that most of the graphs in the database are pruned by our framework. Notice that when we double the range query, which is indicated

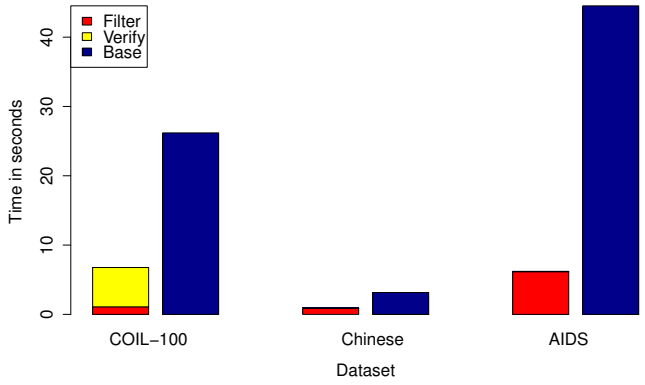


Figure 6.9: The response time of our approach compared with the base approach.

by +1.0X in the figure, still more than 36% of the graphs in the database are pruned by our framework.

6.4.3 Response Time

In this section, we report the response time of our framework for the different datasets. It is measured by the average number of seconds needed to retrieve the similar graphs that satisfy the range threshold. This consists of both the filtering and the verification times as stacked in Figure 6.9. Notice that the verification times for AIDS and Chinese are marginal and does not appear in the figures. We also compare with the response time of the base approach, which does not adopt any indexing nor pruning technique. In average, we see that our approach reduces the response time by an amount of 77%. We notice that the less the pruning power the more time the framework takes in the verification step, which is clear for the COIL-100 dataset. On the other hand, our system filters nearly 97% of the graphs for AIDS and Chinese, which leads to a high filtering time and a low verification time.

We also detail the response time and zoom in to see how many seconds each layer of our framework consumes to process a query graph in Figure 6.10. The first layer denotes the time consumed by 1) accessing the R-tree, 2) merging similar vertices to generate candidate similar graphs, and 3) applying the first layer lower bound distance. The second layer denotes the time consumed by applying both the norm and the lower mapping distances. Theoretically, the smaller the set of candidate similar graphs generated by an approach the more time it takes. For example, for

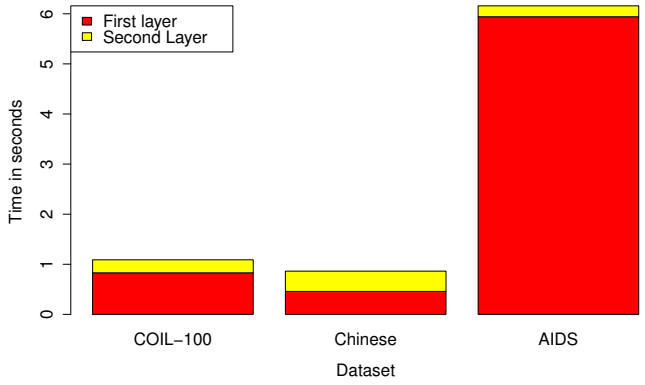


Figure 6.10: Time in seconds that is consumed by each component of our framework.

the AIDS dataset, we saw that the first layer prunes most of the graphs in the graph database. As a result, the first layer takes more time than the second layer.

6.5 Summary

In this chapter, we discussed the geometric graph search problem. Given a geometric graph database and a query graph, the search problem is to efficiently retrieve graphs from a database that are similar to the query graph within a certain range threshold. For geometric graphs, we propose a distance function that estimates the similarity between two graphs, which is defined based on concepts of the graph edit distance. Initially, a vertex distance matrix is constructed from two graphs. Then, the Hungarian algorithm is used to select the best mapping between the vertices of the two graphs. Such a mapping is then used to estimate the overall graph distance. Even with such a heuristic, a linear search of a graph database takes too much time. To solve this problem, we propose a novel framework to efficiently prune non-similar graphs before applying the geometric graph distance. Our framework consists of two layers. The first one efficiently retrieves vertices highly similar to the vertices of the query graph. For this, we propose to index the vertices of the graphs by an R-tree after embedding them in a higher dimensional space. Then, the set of candidate similar vertices is used to generate a set of candidate similar graphs. This is a accomplished by using both an inverted index that maps each vertex to its graph and a lower bound distance function that utilizes the set of candidate similar vertices. The second layer of our framework consists of two consecutive pruning approaches. The

first one utilizes the concept of graph norm, which is defined by the insertion costs of the vertices of a graph. The second pruning approach uses a lower bound to the substitution cost of two vertices and the Hungarian algorithm to prune non-similar graphs.

Using representative geometric graphs extracted from several application domains, we showed the time and space needed to construct and store our index structure, the pruning performance, and the response time to produce the set of candidate similar graphs.

Chapter 7

Conclusions and Future Work

Searching for similar objects is a vital task in many scientific applications, such as in biology, chemistry, computer vision, and pattern recognition. For such applications and many more, graphs have been used to model complex objects and relationships. As a result, the task of finding similar objects is answered by finding similar graphs. Basically, the similarity between two graphs consists of the similarity of the labels that are assigned to the vertices and edges, in addition to the similarity of their structure. Whereas the similarity between the labels can be easily computed, for example, using the Euclidean distance, finding the structural similarity between two graphs is a complex problem. The notion of structural similarity is typically formalized as the graph matching problem, which searches for correspondences between the vertices of two graphs such that the mapped vertices have similar structure.

Unfortunately, the inexact graph matching problem, which considers graphs that differ in the number of vertices, structure, and labeling information, is proved to be NP-hard [20, 26]. As a result, several authors proposed heuristics that approximate the optimal solution. However, such approaches still face problems related to accuracy and scalability.

7.1 Summary

In this thesis, we have proposed a novel framework to estimate the similarity and matching of geometric graphs in 2D space. Such types of graphs have been used by several application domains, such as drug discovery, protein analysis, image processing, rout planning, and many more. Compared to state-of-the-art approaches, our framework scales better, in terms of time and space, as graph size increases. It has a higher matching accuracy when comparing graphs that differ in the number of

vertices, labeling information, and the spatial properties. And finally, it efficiently supports a database search for similar graphs utilizing novel indexing and pruning techniques.

Since the vertex similarity problem is the basis of any graph matching algorithm, we start our contributions by tackling this problem. We proposed the vertex edit distance to tackle such a problem, which we proved to be NP-hard. As a result, we proposed an approach that approximates the optimal solution by utilizing the concept of cyclic string edit distance.

To tackle the graph matching problem, we utilized the concept of vertex similarity and proposed a probabilistic graph matching algorithm that is tolerant to the differences between two graphs. It is based on a novel density function that estimates the probability of matching any two vertices utilizing the vertex-to-vertex similarity concept and the structure of both graphs. Then, a match between two graphs is computed by maximizing a graph likelihood function using the well-known expectation maximization technique.

To enable scalability with respect to graph size, we proposed a novel vertex embedding scheme that bridges the gap between the vector and the graph representations. For this, the vertices are represented by their distances to a given set of prototypes. As a result, the similarity between two vertices is estimated by the Euclidean distance in constant time. Based on this, we proposed a second graph matching algorithm that iteratively improves and expands the match between two graphs. At each iteration, the vertices of the match vote for the similarity of any two vertices from the two graphs.

Finally, we proposed our novel indexing structure for geometric graphs in 2D space. It helps to efficiently search a database for similar graphs. For this, we utilized our approach to compute the similarity between different vertices, and our vertex embedding approach. We proposed three graph similarity functions that are considered lower bounds to the geometric graph distance function. Such lower bound distances help to efficiently prune non-similar graphs before applying a graph matching algorithm.

Throughout this thesis, we used different datasets that are extracted from a variety of applications domains, namely chemoinformatics, bioinformatics, geoinformatics, character recognition, and image analysis. Empirically, our proposed approaches outperformed state-of-the-art graph matching algorithms in terms of matching quality, classification accuracy, and scalability.

7.2 Future Work

Obviously, there are still open issues that could be considered to extend our work. In the following, we suggest directions for future work.

- **Matching geometric graphs in 3D space.** Till now, our graph similarity framework tackles geometric graphs in 2D space. For future work, we recommend to propose vertex similarity measures for graphs in 3D space. For this, vertex edit operations that handle graphs under geometric transformations in 3D space must be proposed. On the other hand, all the modules of our framework, i.e., the graph matching algorithms and the indexing structure, can be naively used to match and index 3D graphs.

- **Dimensionality reduction and vertex embedding.** To extend our vertex embedding scheme, it is worth to study the properties of the embedded space. From our experiments, we noticed that our prototype selection methods do not guarantee a space where all dimensions are mutually independent. In this direction, we recommend to improve such a space by utilizing dimensionality reduction techniques. The result is a reduced space with higher matching accuracy and less complexity.

- **Hierarchical vertex embedding.** Remember that our vertex embedding scheme utilizes only the neighborhoods of the vertices. As a result, the overall structure of the graphs is not preserved. For future work, we recommend to extend our vertex embedding scheme by considering more information about graph structure in a hierarchical style. The first level of the hierarchy is our vertex embedding scheme in Chapter 5. The second layer uses the information of both the first layer and the vertices that are within two hubs from each vertex, and so on. We believe that such a hierarchal embedding scheme can be further used for frequent subgraph discovery.

- **Parallelized graph matching.** Our proposed graph matching algorithms can be easily parallelized using, for example, MapReduce [35]. Remember that our algorithms estimate the similarity of any two vertices at each iteration, which can be easily computed in parallel. In the context of a map-reduce framework, any pair of vertices can be sent to a map job to estimate their similarity. Then, the reduce job simply joins the similarities of the vertices together.

- **Lower bounds to the geometric graph edit distance.** In this thesis, we proposed graph matching algorithms that, in general, adopt the concept of edit distance. However, till now, we do not have a mathematical analysis of the relationship between the geometric graph edit distance and our graph matching algorithms. In particular, in Chapter 6, we discussed a geometric graph matching algorithm based on Bipartite graph matching. For future work, we recommend to analytically study the relationship between such a graph matching algorithm and the optimal solution to the geometric graph edit distance.

Appendix A

Datasets

One of the main goals of this thesis is to build a framework for graph similarity and matching that can be used by several application domains. To empirically test this, through the previous chapters, we used several datasets that are extracted from a variety of application domains. Examples are chemoinformatic, bioinformatics, geoinformatics, character recognition, and image analysis. In the following, we discuss these datasets in more detail.

A.1 Road Networks

The field of road network analysis has been used as the basis for many geographic information systems. A road network is normally modeled as a graph such that cross roads are represented by vertices and road segments are modeled by edges. In addition to this, latitudes and longitudes are used as labels for the vertices and the lengths of the road segments are used as labels for the edges.

Our road network dataset contains three geometric graphs extracted from the California, North America, and the city of Oldenburg road networks [4]. Since a road segment between two intersections is represented by several nodes in the road network, we simplified them using the Douglas Peucker algorithm [41], which is used to simplify a curve by reducing the number of points that represent it. Table A.1 shows the number of vertices and edges for the simplified road networks.

From these three networks, we created two benchmarks. The first one is used to test common subgraph matching algorithms, and the second one is for subgraph matching. For the first benchmark, we create a distorted version from each road network. For this, we use *Itgrag* [3], which is a package used to generate and distort geometric graphs, to first partition the vertices of the road network into non-overlapping

Table A.1: The number of vertices and edges for the three road networks. $|V|$ and $|E|$ denote the number of vertices and the number of edges, respectively.

Road Network	$\|V\|$	$\|E\|$
California	1365	1990
City of Oldenburg	3494	4348
North America	7517	10088

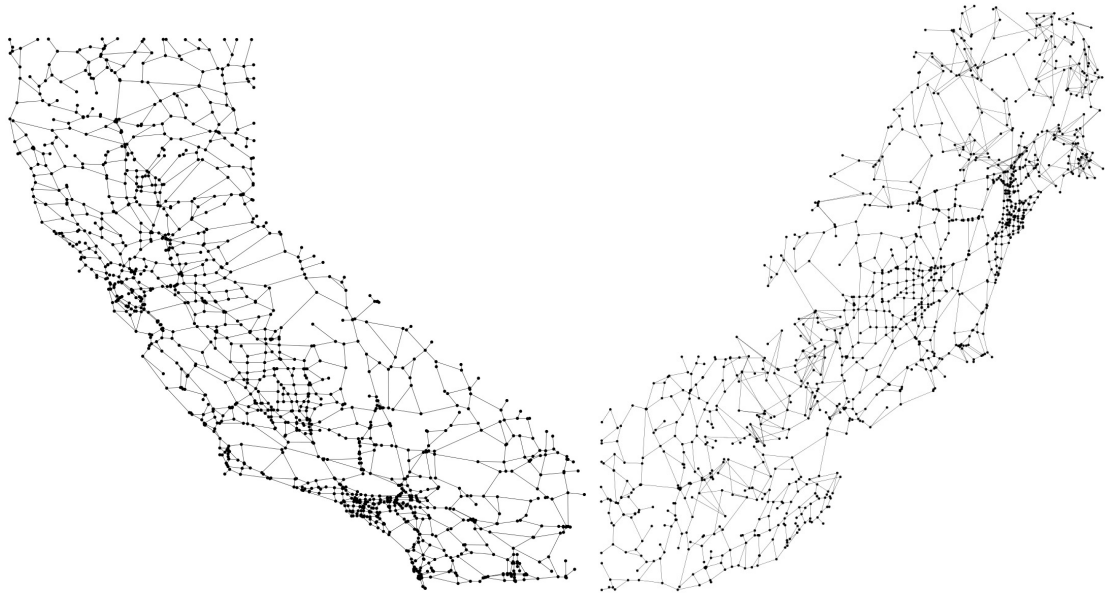


Figure A.1: The California road network on left and its distorted version on right.

clusters [68], and then randomly introduce noise to the structure of some clusters in addition to changing the locations of the vertices. Also, we change the location of some clusters manually so that adjacent clusters become far apart. Figure A.1 shows the California road network and its distorted version. For common subgraph matching, each road network is matched to its distorted version to estimate the matching accuracy.

For subgraph matching we created another benchmark from the California road network. From this road network, we extracted 5 initial subgraphs such that the average size of each subgraph is 104 vertices. From these 5 subgraphs, another dataset of 40 subgraphs is generated. To do this, structural and spatial distortions are applied to the 5 initial subgraphs using Itgrag [3]. Distortion is applied at an increasing level. We computed the amount of distortion needed to make an initial subgraph non-similar to the distorted one. Then, we divide this amount to create four levels of distortion such that each distortion level is represented by 10 subgraphs.

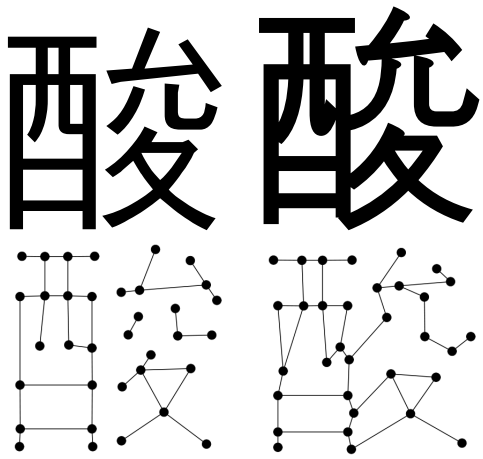


Figure A.2: Two glyphs from different fonts and their geometric graphs for the character with Unicode 9178.

A.2 Chinese Characters

The Chinese character dataset contains geometric graphs extracted from Chinese character glyphs that belong to different fonts [1]. We choose six different fonts: two Chinese, two Japanese, and two Korean. As seen in Figure A.2, the same Chinese character is drawn in different ways in different fonts. This dataset contains a total of 9384 characters, 1564 characters from each font. To extract a geometric graph from a glyph, the medial axis is computed. Small features are pruned and the Douglas Peucker algorithm is used to simplify the medial axis [41].

We divide the characters into two datasets to conduct a classification experiment. A test dataset of 1564 graphs is extracted from the Dotum Korean font. The remaining five fonts build a training dataset of 7820 graphs. Ideally, for a graph from the test dataset, its nearest graph from the train dataset should be a graph derived from the same Chinese character.

A.3 CMU dataset

The CMU house and hotel datasets consist of images for a toy house and hotel, subject to rotation in 3D [2]. The house dataset consists of a total of 111 images (snapshots taken during a rotation). The hotel datasets has 101 images. To extract graphs, 30 landmarks have been manually identified for each image representing graph vertices [19]. Then, edges are created by Delaunay triangulation. Figure A.3 shows two sample images with their geometric graphs. This dataset can be used to track

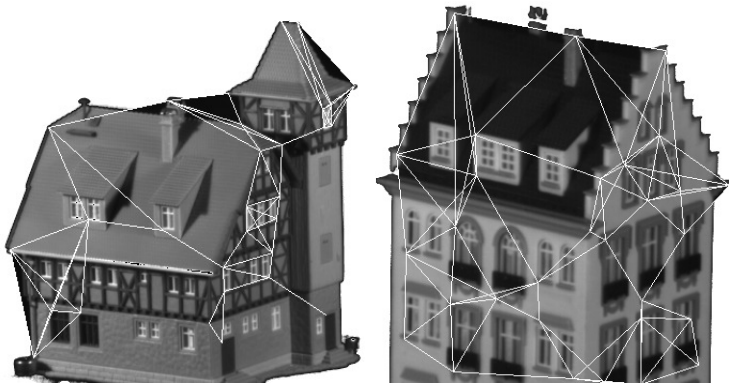


Figure A.3: Two sample images and their geometric graphs form the CMU house/hotel datasets [2]..

the same feature across different images taken at different time frames. Notice that the differences between two snapshots increase when the elapsed time between them increases.

A.4 IAM Graph Dataset

This dataset contains several benchmarks for graph matching and analysis [99]. For our work, we choose three benchmarks that contain geometric graphs.

A.4.1 COIL-100

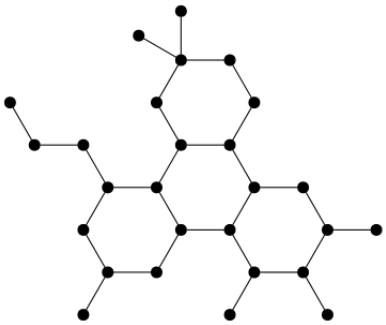
The COIL-100 dataset [87] consists of images of 100 different objects as shown in Figure A.4. Each object creates a sequence of 72 images by rotating the camera in 3D at pose intervals of 5 degrees. To extracted a geometric graph, the Harris corner detection algorithm is used to extract features representing the vertices, then, Delaunay triangulation is used to create the edges. The average sizes of the number of vertices and edges are 21 and 53, respectively. From the COIL-100 dataset we created two benchmarks. The first one is used to test graph indexing structures, which consists of a database of 6200 graphs and 1000 query graphs. The second benchmark is used for graph classification. It consists of a training dataset of 2400 graphs and a test dataset of 1000 graphs.

Figure A.4: Sample images from the COIL-100 dataset [99].

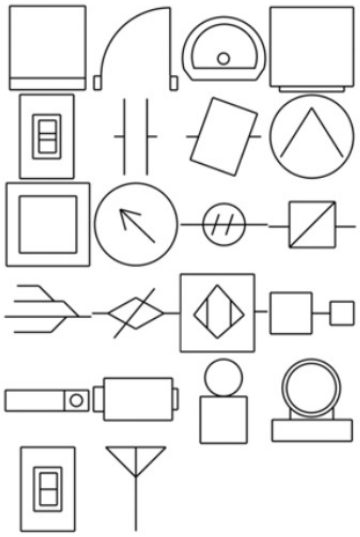
A.4.2 AIDS

The AIDS antiviral screen dataset of active compounds consists of chemical compounds that belong to one of two classes: active or inactive against HIV. A geometric graph is extracted from each chemical compound such that the atoms are represented by the vertices and the covalent bonds are represented by the edges. A vertex is also labeled by the atom type, in addition to its coordinates in 2D space. An edge is labeled by the valence of the linkage. We show an example of a geometric graph that is extracted from the AIDS dataset in Figure A.5(a).

From the AIDS dataset we created two benchmarks. The first one is used to test graph indexing structures and consists of 42,234 graphs. The second benchmark consists of 2000 graphs and is used for graph classification. From a total of 2000 graphs, we use a training set of 1500 graphs: 300 active and 1200 inactive. For testing we used 500 graphs.



(a) A sample geometric graph from the AIDS dataset.



(b) Sample images of the GREC data set [39, 99, 103].

Figure A.5: Sample geometric graphs that are extracted from the AIDS and the GREC datasets.

A.4.3 GREC

The GREC dataset consists of symbols of architectural and electronic drawings [39], which are shown in Figure A.5(b). From 22 drawings a dataset of 1100 images is created. Noise and distortions are used at different levels to create a set of 50 images from each drawing. To extract a geometric graph from each image, intersections and corners are used to create the vertices, which are assigned coordinates in 2D space. Lines and arcs are used to create the edges. This benchmark is used for graph classification. A training set of 814 and a testing set of 286 images are used. For each class, 37 images for training and 13 for testing are chosen.

Bibliography

[1] CJK Fonts: Chinese, Japanese, and Korean Fonts. `http://bookr-mod.googlecode.com/files/cjk-fonts-1.zip`. Accessed: 01/12/2011.

[2] CMU house and hotel datasets. `http://vasc.ri.cmu.edu//idb/html/motion`. Accessed: 16/02/2012.

[3] Distort geometric graphs. `https://code.google.com/p/itgrag/`. Accessed: 23/12/2012.

[4] Road networks dataset. `http://www.cs.utah.edu/~lifeifei/SpatialDataset.htm`. Accessed: 03/06/2013.

[5] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[6] Tatsuya Akutsu, Kyotetsu Kanaya, Akira Ohyama, and Asao Fujiyama. Point matching under non-uniform distortions. *Discrete Applied Mathematics*, 127(1):5–21, 2003.

[7] Helmut Alt and Leonidas J Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. *Handbook of computational geometry*, 1:121–153, 1999.

[8] Ayser Armiti and Michael Gertz. Efficient Geometric Graph Matching Using Vertex Embedding. In *Proceedings of the 21st International Conference on Advances in Geographic Information Systems, SIGSPATIAL'13*, pages 234–243, 2013.

[9] Ayser Armiti and Michael Gertz. Geometric Graph Matching and Similarity: A Probabilistic Approach. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, SSDBM '14, pages 27:1–27:12, 2014.

[10] Ayser Armiti and Michael Gertz. Vertex Similarity - A Basic Framework for Matching Geometric Graphs. In *Proceedings of the 16th LWA Workshops: KDML, IR and FGWM*, pages 111–122, 2014.

[11] Xiang Bai and Longin Jan Latecki. Path Similarity Skeleton Graph Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(7):1282–1292, 2008.

[12] E. Balas and C.S. Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal on Computing*, 15:1054, 1986.

[13] V.D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. Van Dooren. A measure of similarity between graph vertices: Applications to synonym extraction and web searching. *Siam Review*, pages 647–666, 2004.

[14] Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein Function Prediction via Graph Kernels. *Bioinformatics*, 21(1):47–56, January 2005.

[15] J. Bourgain. On lipschitz embedding of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52.

[16] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[17] H Bunke and G Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245–253, 1983.

[18] H. Bunke, P. Foggia, C. Guidobaldi, C. Sansone, and M. Vento. A comparison of algorithms for maximum common subgraph on randomly connected graphs. *Structural, Syntactic, and Statistical Pattern Recognition, SSSPR'02*, pages 85–106, 2002.

[19] T. S. Caetano, T. M. Caelli, D. Schuurmans, and D. A. C. Barone. Graphical Models and Point Pattern Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1646–1663, 2006.

[20] T.S. Caetano, J.J. McAuley, L. Cheng, Q.V. Le, and A.J. Smola. Learning graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(6):1048–1058, 2009.

[21] S.H. Cha. Comprehensive survey on distance/similarity measures between probability density functions. *International Journal of Mathematical Models and Methods in Applied Sciences*, 4(2):300–307, 2007.

[22] Hung-Hsuan Chen, Liang Gou, Xiaolong (Luke) Zhang, and C. Lee Giles. Discovering Missing Links in Networks Using Vertex Similarity Measures. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 138–143, 2012.

[23] Hwann-Tzong Chen, Horng-Horng Lin, and Tyng-Luh Liu. Multi-object tracking using dynamical graph matching. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages II–210–II–217 vol.2, 2001.

[24] Jin Chen, Wynne Hsu, Mong-Li Lee, and See-Kiong Ng. NeMoFinder: dissecting genome-wide protein-protein interactions with meso-scale network motifs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 106–115, 2006.

[25] James Cheng, Yiping Ke, Wilfred Ng, and An Lu. Fg-index: towards verification-free query processing on graph databases. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 857–872, 2007.

[26] O. Cheong, J. Gudmundsson, H.S. Kim, D. Schymura, and F. Stehn. Measuring the Similarity of Geometric Graphs. *Experimental Algorithms*, pages 101–112, 2009.

[27] Minsu Cho, Jungmin Lee, and Kyoung Mu Lee. Reweighted random walks for graph matching. In *Computer Vision–ECCV 2010*, pages 492–505, 2010.

[28] Minsu Cho and Kyoung Mu Lee. Progressive graph matching: Making a move of graphs via probabilistic voting. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition, CVPR'12*, pages 398–405, 2012.

[29] Haili Chui and Anand Rangarajan. A new point matching algorithm for non-rigid registration. *Computer Vision and Image Understanding*, 89(2-3):114–141, 2003.

[30] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[31] D. Conte, P. Foggia, and M. Vento. Challenging complexity of maximum common subgraph detection algorithms: A performance analysis of three algorithms on a wide database of graphs. *Journal of Graph Algorithms and Applications*, 11(1):99–143, 2007.

[32] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty Years Of Graph Matching In Pattern Recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298, 2004.

[33] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs. In *3rd IAPR-TC15 workshop on graph-based representations in pattern recognition*, pages 149–159, 2001.

[34] Eric H Davidson, Jonathan P Rast, Paola Oliveri, Andrew Ransick, Cristina Calestani, Chiou-Hwa Yuh, Takuya Minokawa, Gabriele Amore, Veronica Hinman, Cesar Arenas-Mena, et al. A genomic regulatory network for development. *science*, 295(5560):1669–1678, 2002.

[35] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[36] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistcal Society*, 39(B):1–38, 1977.

[37] Peter J. Dickinson, Horst Bunke, Arek Dadej, and Miro Kraetzl. Matching graphs with unique node labels. *Pattern Analysis and Applications*, 7(3):243–254, 2004.

[38] Reinhard Diestel. *Graph theory.* Springer-Verlag, New York, 4 edition, 2010.

[39] Ph. Dosch and E. Valveny. Report on the Second Symbol Recognition Contest. In *Graphics Recognition. Ten years review and future perspectives. Proceedings of 6th International Workshop on Graphics Recognition, GREC'05*, pages 381–397, 2005.

[40] Prashant Doshi, Ravikanth Kolli, and Christopher Thomas. Inexact matching of ontology graphs using expectation-maximization. *Journal of Web Semantics*, 7(2):90–106, 2009.

[41] D. H. Douglas and T. K. Peuker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Geographer*, 10(2):112–122, 1973.

[42] Development Therapeutics Program DTP. AIDS Antiviral Screen, 2004. Accessed: 07/07/2014.

[43] P.J. Durand, R. Pasari, J.W. Baker, and C. Tsai. An efficient algorithm for similarity analysis of molecules. *Internet Journal of Chemistry*, 2(17):1–16, 1999.

[44] Jack Edmonds and Richard M. Karp. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *Journal of the ACM*, 19(2):248–264, 1972.

[45] M. A. Fischler and R. A. Elschlager. The Representation and Matching of Pictorial Structures. *IEEE Transactions on Computers*, 22(1):67–92, January 1973.

[46] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. Image categorization: Graph edit distance+edge direction histogram. *Pattern Recognition*, 41:3179–3191, 2008.

[47] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and Applications*, 13(1):113–129, 2010.

[48] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[49] Jaume Gibert, Ernest Valveny, and Horst Bunke. Graph embedding in vector spaces by node attribute statistics. *Pattern Recognition*, 45(9):3072–3083, 2012.

[50] S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377–388, 1996.

[51] Edwin R. Hancock and Josef Kittler. Discrete relaxation. *Pattern Recognition*, 23(7):711–733, 1990.

[52] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[53] Huahai He and Ambuj K Singh. Closure-tree: An index structure for graph queries. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE'06.*, pages 38–38, 2006.

[54] D. Hidovic and M. Pelillo. Metrics for attributed graphs based on the maximal similarity common subgraph. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):299–313, 2004.

[55] Gísli R. Hjaltason and Hanan Samet. Properties of embedding methods for similarity searching in metric spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):530–549, 2003.

[56] John E. Hopcroft and J. K. Wong. Linear Time Algorithm for Isomorphism of Planar Graphs (Preliminary Report). In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 172–184, 1974.

[57] B. Huet and E.R. Hancock. Inexact graph retrieval. In *Proceedings of the IEEE Workshop on Content-Based Access of Image and Video Libraries, CBAIVL'99*, pages 40–44, 1999.

[58] Benoit Huet and Edwin R. Hancock. Relational Object Recognition from Large Structural Libraries. *Pattern Recognition*, 35:1895–1915, 2002.

[59] Glen Jeh and Jennifer Widom. SimRank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 538–543, 2002.

[60] Haoliang Jiang, Haixun Wang, Philip S. Yu, and Shuigeng Zhou. GString: A Novel Approach for Efficient Search in Graph Databases. In *Proceedings of the IEEE 23rd International Conference on Data Engineering, ICDE'07*, pages 566–575, 2007.

[61] Xiaoyi Jiang and Horst Bunke. Optimal quadratic-time isomorphism of ordered graphs. *Pattern Recognition*, 32(7):1273–1283, 1999.

[62] S. Jouili, I. Mili, and S. Tabbone. Attributed graph matching using local descriptions. In *Advanced Concepts for Intelligent Vision Systems*, pages 89–99, 2009.

[63] Salim Jouili and Salvatore Tabbone. Graph Matching Based on Node Signatures. In *Graph-Based Representations in Pattern Recognition*, GbRPR '09, pages 154–163, 2009.

[64] Tommi A. Junttila and Petteri Kaski. Engineering an Efficient Canonical Labeling Tool for Large and Sparse Graphs. In *Proceedings of the Ninth Workshop on Algorithm Engineeringand Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics*, 2007.

[65] Justice and Hero. A Binary Linear Programming Formulation of the Graph Edit Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28, 2006.

[66] U. Kang, Martial Hebert, and Soonyong Park. Fast and scalable approximate spectral graph matching for correspondence problems. *Information Sciences*, 220:306–318, 2013.

[67] Richard M. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[68] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[69] Duck Hoon Kim, Il Dong Yun, and Sang Uk Lee. Attributed relational graph matching based on the nested assignment structure. *Pattern Recognition*, 43(3):914–928, 2010.

[70] Jon Michael Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of ACM*, 46(5):604–632, 1999.

[71] D. Knossow, A. Sharma, D. Mateus, and R. Horaud. Inexact Matching of Large and Sparse Graphs Using Laplacian Eigenvectors. In *Graph Based Representation for Pattern Recognition, GbR'09*, pages 144–153, 2009.

[72] Ina Koch. Enumerating all connected maximal common subgraphs in two graphs. *Theory of Computer Science*, 250(1-2):1–30, 2001.

[73] Flip Korn, Nikolaos Sidiropoulos, Christos Faloutsos, Eliot L. Siegel, and Zenon Protopapas. Fast Nearest Neighbor Search in Medical Image Databases. In *Proceedings of the 22th International Conference on Very Large Data Bases, VLDB'96*, pages 215–226, 1996.

[74] H.W. Kuhn. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[75] Michihiro Kuramochi and George Karypis. Discovering Frequent Geometric Subgraphs. In *Proceedings of the 2nd IEEE Conference on Data Mining, ICDM'02*, pages 258–265, 2002.

[76] A.N. Langville and C.D. Meyer. Deeper Inside PageRank. *Internet Mathematics*, 1(3):335–380, 2004.

[77] E. A. Leicht, Petter Holme, and M. E. J. Newman. Vertex similarity in networks. *Physical Review E*, 73, Feb 2006.

[78] Marius Leordeanu and Martial Hebert. A Spectral Technique for Correspondence Problems Using Pairwise Constraints. In *Proceedings of the Tenth IEEE International Conference on Computer Vision, ICCV'05*, pages 1482–1489, 2005.

[79] Eugene M. Luks. Isomorphism of Graphs of Bounded Valence can be Tested in Polynomial Time. *Journal of Computer and System Sciences*, 25(1):42–65, 1982.

[80] Bin Luo and Edwin R. Hancock. Structural graph matching using the EM algorithm and singular value decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1120–1136, 2001.

[81] M. Maes. On a cyclic string-to-string correction problem. *Information Processing Letters*, 35(2):73–78, 1990.

[82] J.J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software: Practice and Experience*, 12(1):23–34, 1982.

[83] BD McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[84] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112, 2014.

[85] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[86] Richard Myers, Richard C. Wilson, and Edwin R. Hancock. Bayesian Graph Edit Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):628–635, 2000.

[87] S. A. Nene, S. K. Nayar, and H. Murase. Columbia Object Image Library (COIL-100). Technical report, Feb 1996.

[88] Michel Neuhaus, Kaspar Riesen, and Horst Bunke. Fast Suboptimal Algorithms for the Computation of Graph Edit Distance. In *Structural, Syntactic, and Statistical Pattern Recognition, SSPR'06*, pages 163–172, 2006.

[89] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, 1999.

[90] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[91] Odysseas Papapetrou, Ekaterini Ioannou, and Dimitrios Skoutas. Efficient discovery of frequent subgraph patterns in uncertain graph databases. In *Proceedings of the 14th International Conference on Extending Database Technology, EDBT'11*, pages 355–366, 2011.

[92] Elzbieta Pekalska and Robert P. W. Duin. *The Dissimilarity Representation for Pattern Recognition: Foundations And Applications (Machine Perception and Artificial Intelligence)*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2005.

[93] Elzbieta Pekalska, Robert P. W. Duin, and Pavel Paclík. Prototype selection for dissimilarity-based classifiers. *Pattern Recognition*, 39(2):189–208, 2006.

[94] Adolfo Piperno. Search Space Contraction in Canonical Labeling of Graphs (Preliminary Version). *The Computing Research Repository*, abs/0804.4881, 2008.

[95] R. Raveaux, J.C. Burie, and J.M. Ogier. A graph matching method and a graph matching distance based on subgraph assignments. *Pattern Recognition Letters*, 31(5):394–406, 2010.

[96] John W. Raymond, Eleanor J. Gardiner, and Peter Willett. RASCAL: Calculation of Graph Similarity using Maximum Common Edge Subgraphs. *Computer Journal*, 45(6):631–644, 2002.

[97] J.W. Raymond and P. Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of computer-aided molecular design*, 16(7):521–533, 2002.

[98] Ronald C. Read and Derek G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1(4):339–363, 1977.

[99] K. Riesen and H. Bunke. IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning. In *Structural, Syntactic, and Statistical Pattern Recognition, SSSPR'08*, pages 287–297, 2008.

[100] K. Riesen and H. Bunke. Graph classification based on vector space embedding. *International Journal of Pattern Recognitionand Artificial Intelligence*, 23(6):1053, 2009.

[101] Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27(7):950–959, 2009.

[102] Kaspar Riesen and Horst Bunke. Graph Classification by Means of Lipschitz Embedding. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 39(6):1472–1483, 2009.

[103] Kaspar Riesen and Horst Bunke. *Graph Classification and Clustering Based on Vector Space Embedding*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2010.

[104] Kaspar Riesen, Michel Neuhaus, Kaspar, and Horst Bunke. Bipartite Graph Matching for Computing the Edit Distance of Graphs. In *Graph-Based Representations in Pattern Recognition, GbRPR'07*, pages 1–12, 2007.

[105] João B. Rocha-Junior and Kjetil Nørvåg. Top-k spatial keyword queries on road networks. In *Proceedings of the 15th international conference on extending database technology, EDBT'12*, pages 168–179. ACM, 2012.

[106] Y. Rubner, C. Tomasi, and L.J. Guibas. A metric for distributions with applications to image databases. In *IEEE Sixth International Conference on Computer Vision*, pages 59–66, 1998.

[107] Conrad Sanderson. Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments. Technical report, NICTA, Australia, October 2010.

[108] Alberto Sanfeliu and King-Sun Fu. A Distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems, man, and cybernetics*, 13(3):353–362, 1983.

[109] Gerard Sanromà, René Alquézar, and Francesc Serratosa. A new graph matching method for point-set correspondence using the EM algorithm and Softassign. *Computer Vision and Image Understanding*, 116(2):292–304, 2012.

[110] G. L. Scott and H. C. Longuet Higgins. An Algorithm for Associating the Features of Two Images. *Proceedings of Royal Society of London*, B-244:21–26, 1991.

[111] Haichuan Shang, Ying Zhang, Xuemin Lin, and Jeffrey Xu Yu. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *Proceedings of the VLDB Endowment, PVLDB*, 1(1):364–375, 2008.

[112] L. S. Shapiro and J. M. Brady. Feature-Based Correspondence: An Eigenvector Approach. *Image and Vision Computing*, 10(5):283–288, June 1992.

[113] Dennis Shasha, Jason Tsong-Li Wang, and Rosalba Giugno. Algorithmics and Applications of Tree and Graph Searching. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, PODS '02*, pages 39–52, 2002.

[114] D. Shin and T. Tjahjadi. Similarity Invariant Delaunay Graph Matching. *Structural, Syntactic, and Statistical Pattern Recognition, SSSPR'08*, pages 25–34, 2008.

[115] Jin Tang, Bo Jiang, Aihua Zheng, and Bin Luo. Graph matching based on spectral embedding with missing value. *Pattern Recognition*, 45(10):3768–3779, 2012.

[116] NA Thacker, PA Riocreux, and RB Yates. Assessing the completeness properties of pairwise geometric histograms. *Image and Vision Computing*, 13(5):423–429, 1995.

[117] Julian R. Ullmann. An Algorithm for Subgraph Isomorphism. *Journal of the ACM*, 23(1):31–42, 1976.

[118] S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):695–703, 1988.

[119] Jules Vleugels and Remco C. Veltkamp. Efficient image retrieval through vantage objects. *Pattern Recognition*, 35(1):69–80, 2002.

[120] R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.

[121] Nikil Wale, Xia Ning, and George Karypis. Trends in Chemical Graph Data Mining. In *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*, pages 581–606. 2010.

[122] Jan Oliver Wallgrün, Diedrich Wolter, and Kai-Florian Richter. Qualitative matching of spatial information. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 300–309, 2010.

[123] Guoren Wang, Bin Wang, Xiaochun Yang, and Ge Yu. Efficiently Indexing Large Sparse Graphs for Similarity Search. *IEEE Transactions on Knowledge and Data Engineering*, 24(3):440–451, 2012.

[124] Xiaoli Wang, Xiaofeng Ding, Anthony K. H. Tung, Shanshan Ying, and Hai Jin. An Efficient Graph Indexing Method. In *Proceedings of the 28th International Conference on Data Engineering, ICDE'12*, 2012.

[125] Xiong Wang, D Shasha, BA Shapiro, I Rigoutsos, and Kaizhong Zhang. Finding Patterns in Three-Dimensional Graphs: Algorithms and Applications to Scientific Data Mining. *IEEE Trans. on Knowl. and Data Eng.*, 14(4):731–749, July 2002.

[126] R. C. Wilson and P. Zhu. A study of graph spectra for comparing graphs and trees. *Pattern Recognition*, 41(9):2833–2841, September 2008.

[127] R.C. Wilson and E.R. Hancock. Structural matching by discrete relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):634–648, 1997.

[128] Laurenz Wiskott, Jean-Marc Fellous, Norbert Krüger, and Christoph von der Malsburg. Face Recognition by Elastic Bunch Graph Matching. In *Proceedings of the International Conference on Image Processing, ICIP'97*, pages 129–132, 1997.

[129] C. F. J. Wu. On the convergence properties of the EM algorithm. *The Annals of Statistics*, 11(1):95–103, 1983.

[130] Bai Xiao, Edwin R. Hancock, and Richard C. Wilson. A generative model for graph matching and embedding. *Computer Vision and Image Understanding*, 113(7):777–789, 2009.

[131] Chuan Xiao, Xuemin Lin, Xiang Zhao, and Wei Wang. Efficient Graph Similarity Joins with Edit Distance Constraints. In *Proceedings of the 28th International Conference on Data Engineering, ICDE'12*, 2012.

[132] Xifeng Yan, Philip S. Yu, and Jiawei Han. Graph indexing based on discriminative frequent structure analysis. *ACM Transactions on Database Systems*, 30(4):960–993, 2005.

[133] Chang Hun You, Lawrence B. Holder, and Diane J. Cook. Application of Graph-based Data Mining to Metabolic Pathways. In *Proceedings of the Sixth IEEE International Conference on Data Mining Workshops, ICDM Workshops 2006.*, pages 169–173, 2006.

[134] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing Stars: On Approximating Graph Edit Distance. *Proceedings of the VLDB Endowment, PVLDB*, 2(1):25–36, 2009.

[135] Shijie Zhang, Meng Hu, and Jiong Yang. TreePi: A Novel Graph Indexing Method. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE'07*, pages 966–975, 2007.

[136] Peixiang Zhao, Jiawei Han, and Yizhou Sun. P-Rank: a comprehensive structural similarity measure over information networks. In *Proceeding of the 18th ACM conference on Information and knowledge management, CIKM'09*, pages 553–562, 2009.

[137] Peixiang Zhao, Jeffrey Xu Yu, and Philip S. Yu. Graph Indexing: Tree + Delta >= Graph. In *Proceedings of the 33rd international conference on Very large data bases, VLDB'07*, pages 938–949, 2007.

[138] Xiang Zhao, Chuan Xiao, Xuemin Lin, Wang Wei, and Yoshiharu Ishikawa. Efficient processing of graph similarity queries with edit distance constraints. *VLDB Journal*, 22(6):727–752, 2013.

[139] Weiguo Zheng, Lei Zou, Xiang Lian, Dong Wang, and Dongyan Zhao. Graph similarity search with edit distance constraint in large graph databases. In *Proceedings of the 22nd ACM international conference on Conference on information and knowledge management, CIKM'13*, pages 1595–1600. ACM, 2013.

[140] Feng Zhou and Fernando De la Torre. Factorized graph matching. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR'12*, pages 127–134, 2012.

[141] Yuanyuan Zhu, Lu Qin, Jeffrey Xu Yu, Yiping Ke, and Xuemin Lin. High efficiency and quality: large graphs matching. In *Proceedings of the 20th ACM international conference on Conference on information and knowledge management, CIKM'11*, pages 1755–1764, 2011.

[142] Lei Zou, Lei Chen, Jeffrey Xu Yu, and Yansheng Lu. A novel spectral coding in a large graph database. In *Proceedings of the 11th International Conference on Extending Database Technology, EDBT'08*, pages 181–192, 2008.