

The Matching-Graph

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Mathias Fuchs

von Luzern LU

Leiter der Arbeit:

PD Dr. K. Riesen
Institut für Informatik, Universität Bern

This work has different copyright licenses and is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International license where not differently stated.

To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-nd/4.0/>



The Matching-Graph

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Mathias Fuchs

von Luzern LU

Leiter der Arbeit:

PD Dr. K. Riesen
Institut für Informatik, Universität Bern

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Bern, den 14.09.2023

Der Dekan:
Prof. Dr. Marco Herwegh

Abstract

The increasing amount of data available and the rate at which it is being collected is driving the rapid development of intelligent information processing and pattern recognition systems. Often the underlying data is inherently complex, making it difficult to represent it using linear, vectorial data structures. Graphs offer a versatile alternative for formal data representation. Actually, quite a number of graph-based pattern recognition methods have been proposed, and a considerable part of these methods rely on graph matching.

This thesis introduces a novel method for encoding specific graph matching information into a meta-graph, termed *matching-graph*. The basic idea is to formalize the stable cores of individual classes of graphs – discovered during intra-class matching. This meta-graph is useful in several applications ranging from the analysis of inherent patterns, over graph classification, to graph augmentation. The benefits of the matching-graphs are evaluated in three parts.

First, their usefulness in classification scenarios is evaluated in two approaches. The first approach is a distance-based classifier that focuses on the matching-graphs during dissimilarity computation. The second approach uses sets of matching-graphs to embed input graphs into a vector space. The basic idea is to first generate hundreds of matching-graphs, and then represent each graph g as a vector that shows the occurrence of, or the distance to, each matching-graph. In a thorough experimental evaluation on real-world data sets it is empirically confirmed that these novel approaches are able to improve the classification accuracy of systems that rely on comparable information as well as state-of-the-art methods.

The second part of the research targets a prevalent challenge in graph-based pattern recognition, viz. computing the maximum common subgraph

(MCS). Current exact algorithms compute the MCS with exponential time complexity. In this second part, it is investigated whether matching-graphs, computable in polynomial time, provide a suitable approximation for the MCS. Results show that, for specific graphs, a matching-graph equals the maximum common edge subgraph, thereby establishing an upper limit to the size of the maximum common induced subgraph. The experimental evaluation further confirms that matching-graphs outperform existing algorithms in terms of computation time and classification accuracy.

The third part of this thesis addresses the problem of graph augmentation. Regardless of the actual representation formalism used, it is inevitable that supervised pattern recognition algorithms need access to large sets of labeled training samples. However, in some cases, this requirement cannot be met because the set of labeled samples is inherently limited. The last part shows that matching-graphs can be used to augment graph training sets in order to make the training of a classifier more robust. The benefit of this approach is empirically validated in two different experiments. First, the augmentation approach is studied on very small graph data sets in conjunction with a graph kernel classifier, and second, the augmentation approach is studied on data sets with reasonable size in conjunction with a graph neural network classifier.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my exceptional supervisor, PD Dr. Kaspar Riesen. Your always motivated way, together with your profound technical knowledge, exceptional guidance, insightful suggestions and open approach to research have been the fundamental pillars of this thesis. I feel truly fortunate to have had the opportunity to work under and learn from such a great person and brilliant researcher. I would also like to thank Prof. Dr. Andreas Fischer for acting as co-referee of this thesis and Prof. Dr. Timo Kehrer for supervising the examination. I would like to acknowledge the funding by the Swiss National Science Foundation (Project 200021_188496).

I am deeply grateful to Dr. Peppo Brambilla and Bettina Choffat for their always positive attitude, moral support and assistance with countless technical and administrative problems. Your presence was always appreciated and without you the creation of this thesis would have been infinitely more difficult. I would also like to thank Prof. Dr. Thomas Studer and his fantastic Logic and Theory Group for providing an amazing working environment and support in all areas. My gratitude also extends to the entire Pattern Recognition Group, and especially to Anthony, who has become a close friend over the course of this thesis. Thank you for your moral support as a friend and your ingenious technical skills as a researcher.

My heartfelt thanks go to my close friends, Tatjana Meier and Stefan Remund, who were also my flatmates. Without your support as friends and as roommates in daily matters, I would have been living in pure chaos. A special thank you goes out to all my close friends for the invaluable moral support and many good memories, without you this journey would not have been possible. I am so grateful to have such a large and amazing circle of close friends. I would also like to thank all my students at Gymnasium

Lerbermatt, for always providing a welcome break from my research life. It has been a privilege to teach all of you.

A very special thank you goes out to Gabriela Waldvogel for her tireless support during the last and most stressful months of this journey. You have made this difficult time one of the best periods of my life and I am forever grateful.

Finally, I would like to express my deepest appreciation to my parents, Elisabeth and Christian Fuchs, for always believing in me as a person and in my work, even when I did not. I cannot express in words, how valuable your support has been to me. This thesis is dedicated to you.

As I reflect on this journey, I realize that these acknowledgments barely scratch the surface of expressing my gratitude to everyone who has played a part in this process. My journey would not have been the same without your support and encouragement. Thank you.

Contents

<i>Abstract</i>	iii
<i>Acknowledgments</i>	v
1. Introduction	1
2. Graph Based Pattern Recognition	7
2.1 Basic Definitions on Graphs	7
2.2 Graph Matching	12
2.2.1 Exact Graph Matching	13
2.2.2 Inexact (Error-tolerant) Graph Matching	16
2.3 Graph Edit Distance	19
2.3.1 Cost Functions	22
2.3.2 Graph Edit Distance Computation	23
3. Graph Data Sets	27
3.1 Chemical Compound Graph Data Sets	27
3.1.1 AIDS	28
3.1.2 Mutagenicity	30
3.1.3 NCI1	31
3.1.4 COX-2	32
3.1.5 PTC(MR)	34
3.2 Protein Graph Data Sets	36
3.2.1 PROTEINS	36
3.2.2 ENZYMES	38

3.3	Data Sets From Various Domains	41
3.3.1	Letter	41
3.3.2	IMDB	43
3.3.3	Cuneiform	44
3.3.4	Synthie	46
3.4	Applications and Cost Functions	48
4.	Matching-Graphs for Graph Classification	51
4.1	Introduction	51
4.2	Related Work and Broader Perspective	54
4.3	Creating Matching-Graphs	55
4.3.1	Selecting a Small Set of Matching-Graphs	57
4.3.2	Creating a Large Set of Distinct Matching-Graphs	59
4.4	Classification with Matching-Graphs	61
4.4.1	Distance-Based Classification Using Matching-Graphs	61
4.4.2	Graph Embedding Using Matching-Graphs	62
4.5	Experimental Evaluation	64
4.5.1	Experimental Setup	64
4.5.2	Validation of Metaparameters	65
4.5.3	Validation Results k -NN(d_M) and SVM($-d_M$)	67
4.5.4	Validation Results k -NN($s_{\varphi(g)}$) and SVM($\kappa_{\varphi(g)}$)	68
4.5.5	Test Results and Discussion	69
4.5.6	Ablation Study, Comparison with State of the Art and Run Time Analysis	70
4.6	Conclusion and Future Work	74
5.	Matching-Graphs and the Maximum Common Subgraph	77
5.1	Introduction	77
5.2	Matching-Graphs	79
5.2.1	Iterative Building of Matching-Graphs	80
5.3	Experimental Evaluation	82
5.3.1	Test Results and Discussion	83
5.4	Maximum Common Subgraph	85
5.5	Relation Between MCS and Matching-Graphs	87
5.6	Experimental Evaluation	90
5.6.1	Comparison with Exact MCS Computation	90

Contents

ix

5.6.2	Classification with MCS Based Dissimilarities . . .	92
5.7	Conclusion and Future Work	95
6.	Matching-Graphs for Graph Augmentation	97
6.1	Introduction	97
6.2	Basic Definitions	100
6.3	Augment Training Sets by Means of Matching-Graphs . .	101
6.4	Experimental Evaluation: Augmenting Training Sets . . .	105
6.4.1	Augment Small Training Sets	105
6.4.2	Graph Augmentation for Neural Networks	109
6.5	Building an Ensemble with Matching-Graphs	111
6.5.1	Matching-Graphs for Ensemble Learning	112
6.5.2	Building a Bagging Ensemble Using Matching- Graphs	114
6.5.3	Experimental evaluation	115
6.6	Conclusion	118
7.	Conclusion and Future Work	121
Appendix A	TSNE Visualizations	127
<i>Bibliography</i>		129

Introduction

1

A novel graph they did define,
Where matching pairs would
intertwine. The underlying
core, a stable part, the heart of
each graph's heart.

ChatGPT

Pattern recognition refers to the cognitive process that involves identifying and categorizing data to make sense of the environment. In essence, pattern recognition is a fundamental human ability, which is part of our intelligence and survival. We are constantly processing vast amounts of sensory data and identifying patterns that allow us to make predictions or informed decisions about the future.

Every individual faces an enormous variety of pattern recognition challenges on a daily basis. Face recognition is one of the most prominent examples of human pattern recognition. Every day we distinguish between countless faces and remember faces we may not have seen for years. Understanding language is another example of pattern recognition. We effortlessly decode the symbols and sounds into meaningful sentences, by recognizing patterns in the arrangement of these symbols. The way we understand people's handwriting, despite variations in style, also demonstrates our ability to recognize patterns in lines and strokes. Navigation is another manifestation of the human ability to recognize patterns. Whether we are maneuvering through a city or our own home, we rely on recognizable patterns in the environment to guide us. Our social interactions also depend on pattern recognition, as we interpret speech patterns, body language, and facial expressions to understand the emotions and intentions of others. Even our health awareness benefits from pattern recognition, as

we monitor changes in normal patterns of our bodies, such as appetite and sleep, to detect potential health problems.

It is obvious that pattern recognition plays a crucial role in our lives, enabling us to make predictions, solve problems and make informed decisions. It is so seamlessly integrated into our daily experiences that we often take it for granted.

However, as the amount of available data increases, it becomes increasingly difficult and tedious for humans to analyze and recognize the patterns that exist. While humans are excellent at recognizing patterns in most everyday activities, there are cases where the complexity, scale, or subtlety of the patterns exceed human capabilities. In such cases, automated pattern recognition is crucial. For example, in medical imaging, radiologists and pathologists have to go through complicated medical images to detect abnormalities that indicate disease. This is a challenging task due to the large volume of images and the minute details involved. Similarly, in the financial sector, millions of transactions are processed every day. It is almost impossible to manually detect fraudulent activity in this vast amount of data. A third example involves the human genome of around three billion base pairs. It is impossible to manually detect patterns or anomalies in this vast sequence of base pairs.

In these and many other scenarios, automated pattern recognition surpasses the limitations of manual analysis, allowing us to process large amounts of data by machines. Actually, pattern recognition as a computer science discipline has become a fundamental part of machine learning and artificial intelligence, and its importance cannot be overstated. To name just a few examples, pattern recognition systems are able to solve various problems such as the recognition of facial expressions [1], the temporal sorting of images [2], the enhancing of weakly lighted images [3], situation recognition [4], or breast cancer detection [5].

Pattern recognition can be roughly divided into two main approaches with respect to the formal data or pattern representation. *Statistical pattern recognition* relies on *feature vectors* for data representation, while *structural pattern recognition* employs *strings*, *trees*, or *graphs* for the same task. At their core, graphs are a collection of nodes and edges, representing entities and their connections, respectively. Because graphs can encode more information than just an ordered and fixed-size list of real numbers, they offer a compelling alternative to vectorial approaches. Graph structures have numerous applications in the real world. For example, in modeling public transport networks, where each station is modeled with a node and the

routes between them are modeled by edges [6]. Graphs can also be used to model electrical circuits, where each node represents a component (such as a resistor, capacitor, etc.) and the connections between them can be modeled by edges [7]. Another common example of graph-based representation is the modeling of social media networks, where the nodes represent individual users, and the edges model the connections between them [8]. A further example is handwritten signatures, where the nodes and edges of a graph can be used to represent individual letters and words [9]. Graphs can also be used to intuitively model biological structures, such as relationships between genes, where nodes represent genes and the edges represent the relationship between them [10]. In pattern recognition applications graphs are also used for a diverse range of tasks [11, 12]. From protein function/structure prediction [13], over signature verification [14], to the detection of Alzheimer's Disease [15].

A key component of structural pattern recognition is *graph matching*, which involves finding similarities between two or more graphs. It can be used to identify and classify patterns within complex data. The field of graph matching can be divided into three different areas. The first area focuses on the direct comparison of graphs, e.g. by using *graph isomorphism* or related concepts. Second, the area of *graph kernels* [16], where the goal is to compute a kernel value for graph pairs to make kernel machines applicable to graphs. After the great success of the second area, a third area has emerged, namely *graph neural networks* [17], which aims to transfer the power of neural networks to the graph domain.

Over the last four decades, a large number of graph matching, graph kernel, and graph neural network procedures have been proposed in the literature [11, 12]. Some commonly used graph matching methods include approaches based on *spectral methods* [18–21], *relaxation labeling* [22–24] or *genetic algorithms* [25–27].

Prominent examples of graph kernels include, for instance, the *subgraph kernel* [28], the *random walk kernel* [29], or the *shortest path kernel* [30]. In the area of graph neural networks, one should mention the *graph convolutional neural network* [31], the *graph attention network* [32] or the *graph isomorphism network* [33].

For the present thesis, one specific graph matching model plays a pivotal role, viz. *graph edit distance*. Graph edit distance [34], introduced about 40 years ago, is still recognized as one of the most flexible and robust graph matching models available. The graph edit distance is an error-tolerant dissimilarity measure that is applicable to any kind of graph, without the

limiting factors of several other graph matching algorithms. Roughly speaking, the graph edit distance calculates the minimum cost that is needed to transform one graph into another graph using a given set of edit operations. In contrast to many other distance measures (e.g. graph kernels or graph neural networks), the graph edit distance offers more information than merely a dissimilarity score, viz. the information which subparts of the underlying graphs actually match with each other (known as *edit path*).

Graph edit distance is a robust method that has applications in all of the three areas of structural pattern recognition mentioned above. To date, however, we see no substantial research that exploits the knowledge contained in the edit path as meta-information for reasoning about graphs, classifying graphs, and/or generating new graphs. The present thesis aims at bridging this gap. That is, we propose a specific encoding of the matching information derived from the graph edit distance in a novel meta-graph, called *matching-graph*. In essence, a matching-graph represents the core of a pair of graphs. An illustrative example of this concept is shown in Figure 1.1¹. The parts highlighted in green represent the core that is eventually represented in the matching-graph.

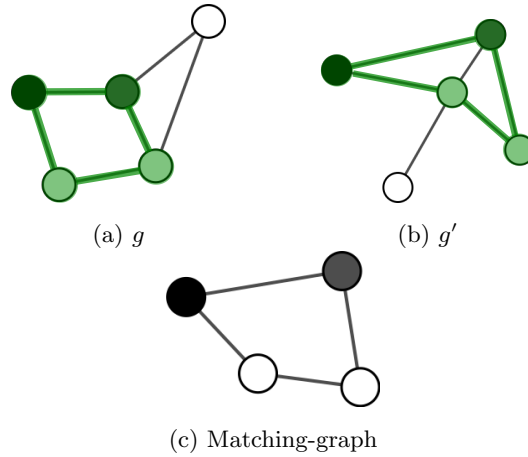


Fig. 1.1: Two graphs g and g' and their resulting core, termed matching-graph.

¹Readers who are familiar with the maximum common subgraph might notice a certain similarity between the matching-graph and the maximum common subgraph. This is indeed the case and will be discussed in detail in Chapter 5.

In principle, the proposed procedure first computes the graph edit distance for several pairs of graphs. Based on the matching found, the matching-graph is eventually created, which formalizes the corresponding parts of the two graphs. The information contained by the matching-graph is able to accurately model the core of a given class, and thus helps to understand underlying patterns. The overall goal of the present thesis is to introduce matching-graphs, investigate their benefits by exploring several possible applications, and verify their usefulness in diverse experiments.

The remainder of this thesis is organized as follows (see Figure 1.2 for a

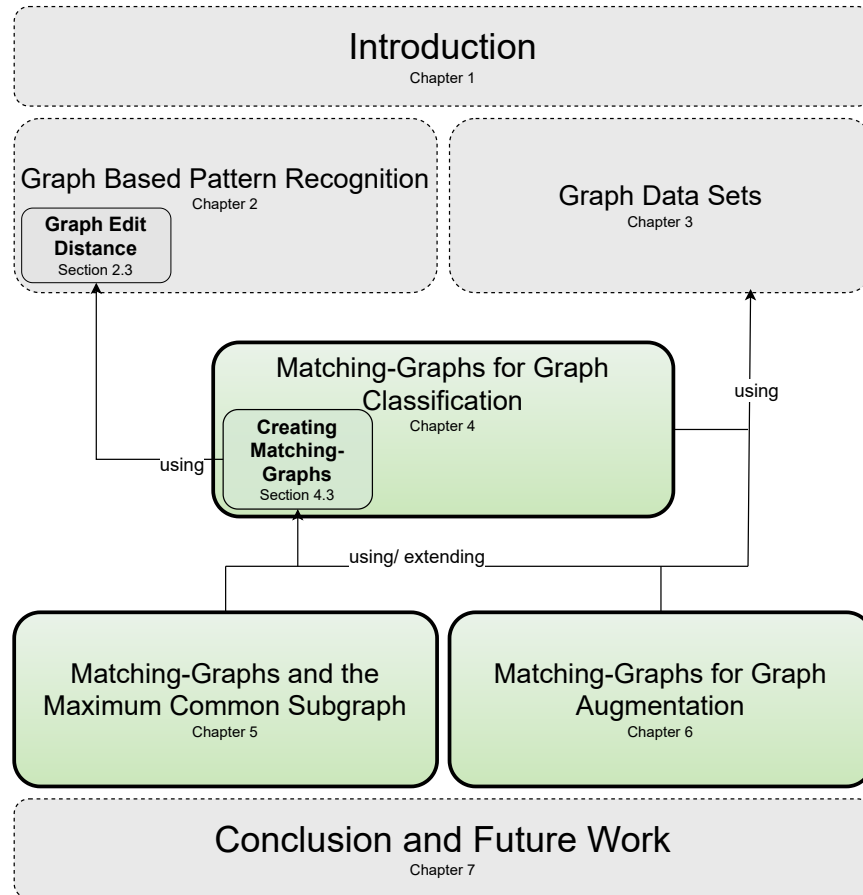


Fig. 1.2: Overview of the structure of the present thesis. The chapters highlighted in green mark the main contribution of this thesis.

graphical overview). First, Chapter 2 provides an overview of the theoretical background as well as existing concepts which are relevant for understanding the details of the present contribution. Next, Chapter 3 describes the data sets used throughout this thesis. In total, 11 different graph data sets with very different characteristics from several different domains are presented and discussed. These data sets are used in various experimental evaluations throughout the thesis. The next three chapters (Chapters 4, 5, and 6) represent the main contribution of this thesis. Note that the thesis is structured so that Chapters 4, 5, and 6 are self-contained and understandable without having to read the rest of the thesis. In Chapter 4 the concept of matching-graphs is explained in detail, and it is shown how matching-graphs can be used for classification. Namely, we introduce two conceptually different strategies for using the matching-graph to solve classification problems. First, the matching-graphs are used in a distance-based classifier, and second, a classifier is built based on a graph embedding that crucially relies on the matching-graphs. This chapter marks the first part of our contribution and is based on three preliminary conference papers [35–37] and one journal paper [38]². Then, in Chapter 5, the matching-graph is qualitatively evaluated using an iterative generation procedure and compared to the maximum common subgraph. This is done on a theoretical level, as well as underlined by an exhaustive experimental evaluation. This chapter is based on a journal paper [39]. The last part of the contribution of this thesis is described in Chapter 6, where the matching-graphs are used for graph augmentation purposes in three different scenarios. First, the matching-graphs are used to augment very small graph data sets. Second, the matching-graphs are used to augment graph data sets to be used in conjunction with graph neural networks. Third, the matching-graphs are used to build a robust and diverse ensemble classifier. Chapter 6 summarizes and combines three preliminary papers [40–42]. Finally, Chapter 7 draws general conclusions and suggests some worthwhile avenues for future research activities.

²This work was published under a Creative Commons license <https://creativecommons.org/licenses/by/4.0/>, <https://s100.copyright.com/AppDispatchServlet?publisherName=ELS&contentID=S0031320322003272&orderBeanReset=true>

Graph Based Pattern Recognition

2

In the realm of theory, where the basics take root, Concepts dance freely, in harmony, not mute. They twirl and they leap, ideas forming a band, Unveiling truths that practice alone can't command.

ChatGPT

This chapter closely follows [43, 44] and provides the necessary theoretical background on which the main contribution of this thesis is based (described in Chapters 4, 5 and 6)¹.

The present chapter is structured as follows. First, Section 2.1 provides a comprehensive overview of the necessary theoretical foundations of graphs. Second, in Section 2.2 the concept of graph matching is explained in detail. Finally, in Section 2.3, one graph matching framework, namedly *graph edit distance*, and a specific algorithm called *BP* are explained in detail, as the matching-graphs, which are the main contribution of this thesis, are based on this concept.

2.1 Basic Definitions on Graphs

The present thesis is based on graph-based pattern representations. The following definition makes it possible to handle arbitrarily structured graphs with unconstrained labeling functions.

¹It is important to note, however, that this chapter does not describe a contribution of the author.

Table 2.1: Graph related notations used throughout this thesis.

Expression	Explanation
g, g'	Two graphs, when exactly two graphs are relevant.
g_i, g_j, \dots, g_n where $i, j, \dots, n \in \mathbb{Z}$	Two or more graphs, when the indices are relevant,

Definition 2.1 (Graph). Let L_V and L_E be finite or infinite label sets for nodes and edges, respectively. A graph g is a four-tuple $g = (V, E, \mu, \nu)$, where

- V is the finite set of nodes,
- $E \subseteq V \times V$ is the set of edges,
- $\mu : V \rightarrow L_V$ is the node labeling function, and
- $\nu : E \rightarrow L_E$ is the edge labeling function.

The size of a graph g is defined as the number of nodes, i.e. $|V|$.

In Table 2.1, two notation systems for graphs used throughout the thesis are defined to give the reader a better understanding. In some algorithms and applications it is necessary to include *empty “nodes”* and/or *empty “edges”*. Both empty nodes and empty edges are denoted by ε . Edges are given by pairs of nodes $(u, v) \in V \times V$. Commonly, one distinguishes between directed and undirected edges:

- *Directed edges:* The direction of the edges $(u, v) \in E$ is indicative and $u \in V$ is denoted as *source node* and $v \in V$ as *target node*.
- *Undirected edges:* The direction of the edges is not indicative, which means that the edge $(u, v) \in E$ is the same as the reverse edge $(v, u) \in E$ (with identical labels, i.e. $\nu(u, v) = \nu(v, u)$). Naturally, because edges in both directions exist in this case, the direction of an edge can be ignored, and both directed edges can be accumulated to one undirected edge. A graph with undirected edges is called an *undirected graph*.

Two nodes, u and v , connected by an undirected edge (u, v) , are defined as *adjacent*. If $(u, v) \in E$, v is considered a *neighbor* of u (and vice versa). The collection of neighbors for a specific node u is called the *neighborhood* of u ,

represented by $\mathcal{N}(v)$. The *degree* of a node $u \in V$, denoted as $\deg(u)$, refers to the number of nodes adjacent to u (or the number of *incident* edges of u). More specifically, the *in-degree* and *out-degree* of a node $u \in V$, denoted as $\text{in}(u)$ and $\text{out}(u)$, correspond to the number of incoming and outgoing edges of node u , respectively. In undirected graphs, $\text{in}(u) = \text{out}(u) = \deg(u)$ for all $u \in V$.

A large variety of specific graph types are included in Definition. 2.1. The following list outlines some of these special types.

- *Labeled graphs*: Given that the nodes and/or the edges are labeled, the graphs are referred to as *labeled graphs*². The labels for nodes as well as edges are given by a set of continuous labels $L = \mathbb{R}^n$ or by a set of categorical labels, which can either be a set of integers $L = \{1, 2, 3, \dots\}$ or a set of symbolic labels $L = \{\alpha, \beta, \gamma, \dots\}$, or a combination of different alphabets from different domains.
- *Unlabeled graphs*: These graphs are particular types of graphs in which every node and edge is assigned the same (empty) label \emptyset , symbolized as $L_V = L_E = \{\emptyset\}$. In this context, a graph is usually represented as $g = (V, E)$, without the labeling functions μ and ν .
- *Directed graphs* and *undirected graphs*: These graphs consist of directed or undirected edges, respectively.
- *Connected and disconnected graphs*: A graph is *connected* if all nodes are connected to each other (directly or indirectly). A graph that is not connected is called *disconnected*. A *connected component* refers to the largest connected subgraph within a graph. A graph is only connected if it consists of a single connected component.
- *Weighted graphs*: For this type of graph, the nodes are unlabeled, $L_V = \{\emptyset\}$ and the edges contain continuous labels, $L_E = \mathbb{R}$.
- *Graphs with unique node labels*: In these graphs, the labeling is restricted such that each node contains a unique label. Formally, for any two nodes $u, v \in V$, if $u \neq v$, then $\mu(u) \neq \mu(v)$.
- *Trees*: Trees are connected acyclic graphs, where nodes can be categorized into *children nodes* and *parent nodes*. Every node has zero or more children nodes and at most one parent node.
- *Ordered Graphs*: An ordered graph is a directed graph with *topologically ordered* node sets V . This means that for each directed edge $(u, v) \in E$, u is sorted before v in V .

² *Attributes* and *attributed graphs* are sometimes synonymously used for *labels* and *labeled graphs*, respectively.

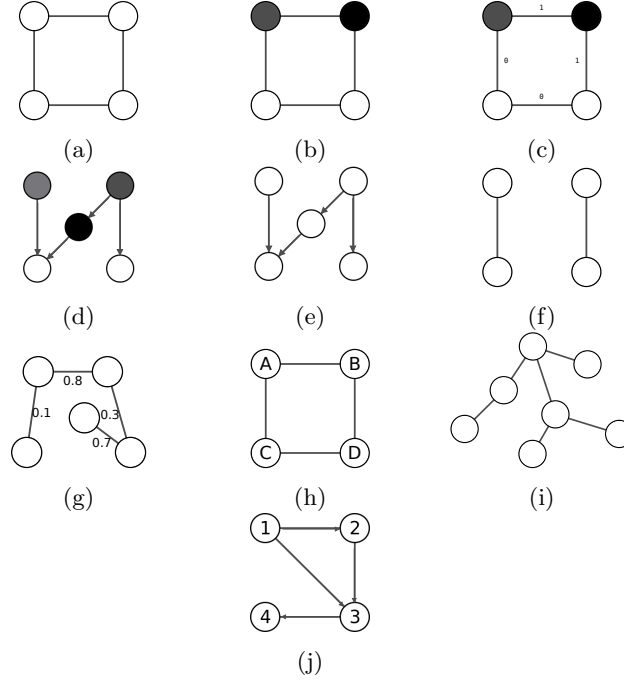


Fig. 2.1: Different kinds of graphs: (a) unlabeled, (b) labeled nodes, (c) labeled nodes and labeled edges (edges are labeled with binary numbers), (d) directed with labeled nodes, (e) directed unlabeled, (f) unconnected (a graph with two connected components), (g) weighted graph, (h) graph with unique node labels, (i) tree, (j) ordered graph.

Example 2.1. In Figure 2.1 different kinds of graphs are shown. Different shades of grey refer to different labels.

The present thesis exclusively uses undirected graphs that are potentially labeled with continuous and/or categorical node and/or edge labels. For more information on the data sets used, see Chapter 3.

The next relevant concept is the concept of a *subgraph*.

Definition 2.2 (Subgraph). Let $g = (V, E, \mu, \nu)$ and $g' = (V', E', \mu', \nu')$ be graphs. Graph g' is a subgraph of g , denoted by $g' \subseteq g$, if

- (1) $V' \subseteq V$,
- (2) $E' \subseteq E$,
- (3) $\mu'(u) = \mu(u)$ and $\nu'(e) = \nu(e)$ for all $u \in V'$ and for all $e \in E'$.

By replacing the second condition (2) by the following condition

$$(2') \quad E' = E \cap V' \times V',$$

g' becomes an *induced subgraph* of g . That is, g' is an induced subgraph of g if the edge set E' of g' includes all the edges $(u, v) \in E$ of graph g that connect two nodes u and v actually present in V' .

Obviously, a subgraph g' is obtained from a graph g by removing some nodes and their incident (as well as possibly some additional) edges from g . For g' to be an induced subgraph of g , some nodes, including their incident edges, are removed from g only, i.e. no additional edge removal is allowed.

Example 2.2. Figure 2.2 illustrates an example of a graph g along with two potential induced subgraphs g_i and g_j , and two non-induced subgraphs g_k and g_o . Node labels are indicated with two different colors (grey and white), while the numbers indicate an arbitrary identifier (this also applies to all other illustrations in the present thesis). Graphs g_i and g_j are induced, because all edges that exist in the original graph g , also exist in E_i and E_j , respectively. Vice versa, g_k as well as g_o are non-induced, because one edge, actually present in g , does not exist in E_k and E_o , respectively. As can be seen from this example, subgraphs (induced and non-induced) might contain isolated nodes (i.e., nodes without any edges).

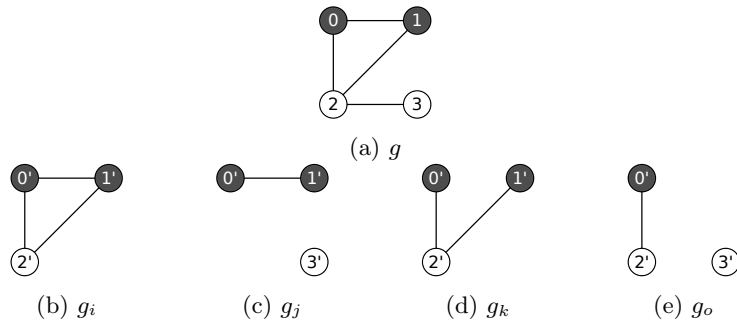


Fig. 2.2: A graph g and two induced subgraphs g_i and g_j as well as two non-induced subgraphs g_k and g_o .

The edge structure of a graph $g = (V, E)$ with $V = \{v_1, \dots, v_n\}$ and $E = \{e_1, \dots, e_m\}$ is commonly described using the *adjacency matrix* $\mathbf{A}(g)$.

Definition 2.3 (Adjacency Matrix). The $n \times n$ adjacency matrix $\mathbf{A}(g)$ of a graph g with entries $\mathbf{A}(i, j) = (a_{ij})$ ($i, j = 1, \dots, n$) is defined by

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

The entry a_{ij} of the adjacency matrix $\mathbf{A}(g)$ is equal to 1 if there is an edge (v_i, v_j) connecting the i -th node with the j -th node in g , and 0 otherwise.

In case of labeled edges, instead of only encoding the presence or absence of an edge, the edge label can be encoded by means of the labeling function $\nu : E \rightarrow L_E$. Formally, if $(v_i, v_j) \in E$, then $a_{ij} = \nu((v_i, v_j))$

Example 2.3. The adjacency matrix of the graph in Figure 2.2 (a).

$$\mathbf{A}(g) = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

2.2 Graph Matching

The concept of dissimilarity is an important issue in many domains of pattern recognition. The process of evaluating the dissimilarity of two graphs is commonly referred to as *graph matching*. The overall aim of graph matching is to find a correspondence between the nodes and edges of two graphs that satisfies some, more or less, stringent constraints. By means of a graph matching process similar substructures in one graph are mapped to similar substructures in the other graph. Through the graph matching procedure, an associated similarity or dissimilarity score is usually calculated, which quantifies the distance between two graphs g and g' . This distance is usually denoted as $d(g, g')$.

Roughly speaking, there are two categories of graph matching: *exact graph matching* and *inexact graph matching* (also termed *error-tolerant graph matching*).

- **Exact graph matching** is the search for a mapping between the nodes of two graphs which is edge-preserving, in the sense that if two nodes in the first graph are linked by an edge, the corresponding nodes in the second graph must have an edge, too. Several variants of exact graph

matching exist depending on, for instance, whether this constraint must hold in both directions of the mapping or not.

- **Inexact or error-tolerant graph matching** is the process of finding correspondences between the nodes of two graphs, while taking into account inconsistencies such as missing, extra, or distorted nodes and edges, in order to identify similar structures or patterns despite the presence of noise or variation.

Several methods from both categories are relevant for this thesis and are discussed in the following sections. In Section 2.2.1, two exact methods are discussed, namely the concepts of *isomorphism* and *subgraph isomorphism*, which are relevant for the embedding-based classifier used in Chapter 4. Next, in Section 2.2.2 inexact graph matching and several concrete methods for this category are explained. Finally, in Section 2.3 the concept of *graph edit distance* for inexact matching is explained in detail, upon which the major contribution of this thesis –the matching-graph– is built.

2.2.1 Exact Graph Matching

The aim of exact graph matching is to determine whether two graphs, or at least part of them, are identical in terms of structure and labels. Generally, for the nodes (and also the edges) of a graph there is no unique order. Thus, for a single graph $g = (V, E)$ with n nodes there are $n!$ possibilities to arrange the nodes of g . Consequently, for checking two graphs g and g' for structural identity, it is not possible to simply compare their corresponding adjacency matrices $\mathbf{A}(g)$ and $\mathbf{A}(g')$. The identity of two graphs g and g' is commonly established by defining a function, termed *graph isomorphism* [44, 45], mapping g to g' .

Definition 2.4 (Graph Isomorphism). Assume that two graphs $g = (V, E, \mu, \nu)$ and $g' = (V', E', \mu', \nu')$ are given. A graph isomorphism is a bijective function $f : V \rightarrow V'$ satisfying

- (1) $\mu(u) = \mu'(f(u))$ for all nodes $u \in V$
- (2) for each edge $e = (u, v) \in E$, there exists an edge $e' = (f(u), f(v)) \in E'$ such that $\nu(e) = \nu'(e')$
- (3) for each edge $e' = (u, v) \in E'$, there exists an edge $e = (f^{-1}(u), f^{-1}(v)) \in E$ such that $\nu(e) = \nu'(e')$

Two graphs g and g' are called *isomorphic* if there exists an isomorphism between them.

Example 2.4. In Figure 2.3 (a) and (b) two isomorphic graphs are shown. In this example, the edges are unlabeled and the labels of the nodes are represented by the colors black and white (the numbers 0, 1, 2... or 0', 1', 2'... are random indices to identify the nodes). The corresponding isomorphism mapping is $f : \{0 \mapsto 0', 1 \mapsto 1', 2 \mapsto 2', 3 \mapsto 3'\}$.

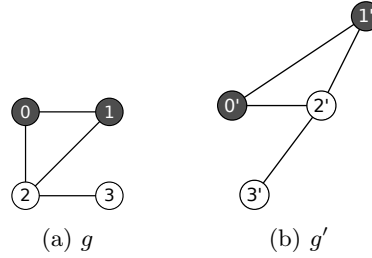


Fig. 2.3: Two graphs g and g' that are isomorphic.

Clearly, isomorphic graphs share the same structure and labels. It is necessary to establish a bijective mapping between each node of the first graph and each node of the second graph to determine the existence of graph isomorphism. This mapping must maintain the edge structure and ensure consistency in node and edge labels.

Formally, if and only if the corresponding labels are the same, this is $\mu(u) = \mu'(f(u))$, a specific node u in graph g can be associated with node $f(u)$ in g' . This also applies to the edges, which means that their labels should also be the same after mapping, viz. $\nu((u, v)) = \nu'(f(u, v))$. Furthermore, if two nodes (u, v) are connected in g , their mapped counterparts $f(u)$ and $f(v)$ in g' must also be connected [43].

Unfortunately, no polynomial runtime algorithm is known for the problem of graph isomorphism [46]. Note that, for the graph isomorphism problem, there are no indications that this particular problem belongs to P. Furthermore, graph isomorphism is currently the most prominent decision problem for which it is not yet proved whether it is P or NP-complete [47]. Moreover, there are strong assumptions that indicate that graph isomorphism is not even NP-complete [48], and thus the computational complexity of any of the available algorithms for graph isomorphism would be exponential in the number of nodes of the two graphs.

However, since the pattern recognition scenarios encountered in practice are usually different from the worst cases, and the labels of both

nodes and edges often help to substantially reduce the search time, the actual computation time can still be manageable. Polynomial algorithms for graph isomorphism have been developed for special kinds of graphs, such as trees [49, 50], bounded-valence graphs [51], ordered graphs [52], planar graphs [53], permutation graphs [54] and graphs with unique node labels [55, 56].

Subgraph isomorphism, a term closely linked to graph isomorphism, refers to the equality of subgraphs. Intuitively, subgraph isomorphism aims at identifying if a given smaller graph can be found exactly within a larger one. More formally, rather than demanding a complete match as in graph isomorphism, subgraph isomorphism requires only the existence of an isomorphism between a graph g and a subgraph of another graph g' .

Definition 2.5 (Subgraph Isomorphism). *Let $g = (V, E, \mu, \nu)$ and $g' = (V', E', \mu', \nu')$ be graphs. An injective function $f : V \rightarrow V'$ from g to g' is a subgraph isomorphism if there exists a subgraph $g'' \subseteq g'$ such that f is a graph isomorphism between g and g'' . The subgraph isomorphism is denoted as $g \subseteq g'$.*

Example 2.5. In Figures 2.4 (a) and (b), an example of a graph g' that is subgraph isomorphic to a graph g is shown.

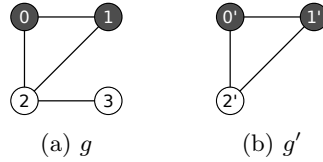


Fig. 2.4: Graph g' (b) is subgraph-isomorphic to g (a), meaning that g' is isomorphic to a subgraph of g .

Actually, subgraph isomorphism is a harder problem than graph isomorphism as one has to check not only whether a permutation of g is identical to g' , but also has to decide whether g is isomorphic to any of the subgraphs of g' with size $|V|$. In contrast with the problem of graph isomorphism, subgraph isomorphism is known to be NP-complete.

Common methods for testing graph or subgraph isomorphism are based on tree search approaches using backtracking, where both breadth-first and depth-first search procedures can be employed [57, 58]. The core concept involves iteratively extending a partial node matching that maps nodes

from two graphs onto each other. This extension continues until either the edge structure constraint is violated or the node or edge labels become inconsistent. In either of these situations, a backtracking process is initiated, i.e. the most recent node mappings are reversed until a partial node matching with a viable alternative expansion is found. Of course, if no further expansion of the partial node mapping can occur without breaking the constraints, the algorithm will terminate, indicating that there is no isomorphism between the two graphs under consideration.

Another important graph matching concept relevant to this thesis is the *maximum common subgraph* problem. Intuitively, the maximum common subgraph $MCS(g, g')$ between two graphs g and g' is the largest possible subgraph of both g and g' . This particular concept is explained in more detail in Chapter 5, as this chapter is devoted exclusively to the analysis of the similarity between the maximum common subgraph and the novel matching-graph.

2.2.2 Inexact (Error-tolerant) Graph Matching

In exact graph matching for two graphs g and g' to be similar, it is required that a significant part of the topology together with the corresponding node and edge labels in g and g' is identical. The usage of exact graph matching algorithms can thus be problematic in many real-world scenarios, where noisy and incomplete data is often present. In these cases, exact graph matching would be too stringent to handle these imperfections, not allowing for meaningful matches in the presence of noise.

Example 2.6. Consider the two graphs g and g' in Figure 2.5 [59] that both represent the word *October* – all exact graph matching procedures discussed so far would fail in terms of finding a high dissimilarity value on these graphs.

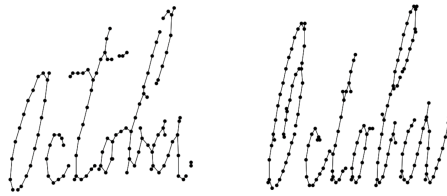


Fig. 2.5: The word *October* (handwritten), two different times, represented by two graphs g and g' (figure stems from [59]).

Inexact graph matching, also known as *approximate or error-tolerant graph matching*, is needed when exact graph matching is too restrictive or impractical for a given application. Inexact methods offer a certain tolerance to errors and allow for some degree of deviation between the graphs that are compared. Furthermore, inexact graph matching provides us with the flexibility to define various similarity or distance measures between graphs, making it adaptable to different requirements, allowing us to incorporate domain-specific knowledge or customizing the matching process. For this reason, a large number of inexact graph matching methods have been proposed [11, 12].

Error-tolerant graph matching extends the idea of graph isomorphism in three important ways. First, given two graphs $g = (V, E, \mu, \nu)$ and $g' = (V', E', \mu', \nu')$, it allows the mapping of nodes $u \in V$ to nodes $f(u) \in V'$ with different labels ($\mu(u) \neq \mu'(f(u))$), which also applies to edges. Second, it allows for node mappings that may not preserve the edge structure. Finally, error-tolerant graph matching explicitly allows the deletion of some nodes (and edges) from the first graph and/or the insertion of nodes (and edges) into the second graph, as opposed to matching all nodes (and edges) with a bijective function between the graphs involved.

Error-tolerant graph matching algorithms aim to find a mapping f between two graphs that minimizes the overall cost $c(f)$. Cost $c(f)$ accounts for node mappings, node deletions, node insertions, edge mappings, edge deletions, and edge insertions. The challenge of optimizing the cost $c(f)$ for an error-tolerant graph matching f is known to be NP-hard [43], implying that the run time to minimize $c(f)$ can be very large even for relatively small graphs.

Various error-tolerant graph matching algorithms have been proposed in the literature. In the following paragraphs some commonly used categories are briefly described.

Spectral methods constitute a first important category for error-tolerant graph matching [18–21]. These methods are based on the fact that the eigenvectors and eigenvalues are invariant to node permutations, meaning that if two graphs are structurally isomorphic they will have the same eigendecomposition [43]. In [19], a hierarchical clustering method is used for graph matching. The method constructs a mixture model over possible correspondences and uses the modal eigenvalues and coefficients to compute cluster correspondence probabilities. In [20], a subspace projection method is used, where nodes of different graphs are projected into a common metric space. In this space, correspondences between nodes are

defined by their edge distributions. In [21], the affinity matrix of a graph is approximated using the linear combination of Kronecker products between bases and index matrices, which requires less memory than computing the whole affinity matrix. Based on this matrix, the eigenvector is computed for the matching process.

Other approaches are based on relaxation labeling techniques [22–24]. These follow the idea that the graph matching problem is formulated as a labeling problem, where each node of a graph is assigned a label from a given set of labels, such that it matches a node of another graph [22]. Each node of each graph is then represented as a vector of probabilities that states how suitable a potential label is [43]. In [23], the relaxation framework is extended using a Bayesian consistency measure. In [24] it is proposed to use the expectation maximization algorithm [60] to iteratively get the state of a match between two graphs by estimating a set of assignment variables.

Several inexact graph matching methods based on genetic algorithms [61] have been proposed as well [25–27]. The general idea of these approaches is to define the matchings as the chromosomes and to use a corresponding performance criterion, e.g. fitness function to improve the quality of the chromosomes. The pool of chromosomes changes iteratively into so-called generations of chromosomes. To generate a diverse pool of chromosomes, so called *mutations*, *crossovers* and *reproductions* are applied at each iteration. The algorithm eventually favours promising chromosomes, i.e. chromosomes with a high fitness value. In [25] the crossover is performed by combining consistent subgraphs. In [26], local search strategies and an inhibitive selection operator are used to maintain the diversity of the population. In [27] a roulette wheel selection method is used to select parents for crossover.

Kernel methods are also widely used in error-tolerant graph matching [62, 63]. These methods aim at implicitly embedding graphs into the vector space, which in turn enables the use of certain statistical pattern recognition methods. There are several types of graph kernels available. The idea of random walk kernels, for instance, is to measure the similarity between two graphs by the number of random walks that are common in both graphs [29]. The shortest-path kernel [30] is a type of graph kernel based on the shortest distance between nodes in a graph. In particular, shortest-path kernels consider paths between two pairs of nodes in a graph and use this similarity to compute the kernel value. Another type of kernel is based on convolution operations, which are able to infer the similarity of complex objects by looking at the similarity of their simpler parts [64, 65].

The graphlet kernel [66, 67], for instance, is based on convolution kernels and works by sampling subgraphs (called graphlets) of fixed size in a graph. More details on more types of kernels are discussed in Chapter 4, where a novel graph embedding using matching-graphs is introduced and discussed.

With the surging popularity of neural networks, another type of error-tolerant graph matching has been proposed, namely graph neural networks (GNNs). In [68], for instance, a Hopfield neural network is applied in order to minimize a given energy criterion used for graph matching. A similar concept is used in [69] for graph classification. In [70] the idea of a GNN is introduced that encodes the nodes of a graph into a vector by aggregating information about the neighborhood of a node using a form of a neural message passing algorithm. In [31] the Graph Convolutional Network (GCN) is proposed, which adds a convolution operation for neural networks. The GCN operation uses a weight matrix to aggregate the representations of neighboring nodes based on their connectivity in the graph. This matrix is learned and shared between all nodes of the graph. In [32] the Graph Attention Network (GAT) is proposed. GATs leverage self-attentional layers to attend to nodes and their neighborhood features. The key idea is that when updating the feature vector of a node, a GAT does not treat all neighbors equally, but instead assigns weighted attention to them. This means that the new feature vector for a node is a weighted sum of the feature vectors of its neighbours, where the weights are determined by an attention mechanism. In [33] another extension to the GNN model is introduced, called the graph isomorphism network (GIN). The GIN model has the same discriminative power as the Weisfeiler-Lehmann graph isomorphism test (which is able to check if two graphs are non-isomorphic). In the GIN model the node aggregation function is learned by using a 1-layer multi-layer perceptron.

For this thesis, the graph edit distance is the most important paradigm of inexact graph matching. Hence, this particular concept is explained in detail in the next Section.

2.3 Graph Edit Distance

This thesis relies on the paradigm of *graph edit distance* [34, 71] for inexact graph matching. Graph edit distance aims to offer a metric for quantifying the dissimilarity between two graphs by determining the minimum number of edit operations needed to transform one graph into the other.

Graph edit distance is capable of handling directed and undirected graphs, as well as labeled and unlabeled ones. When labels are present on nodes, edges, or both, there is no need to consider any constraints on the respective label alphabets. Additionally, the graph edit distance can be adapted and customized for various problem specifications by employing application-specific cost functions. Consequently, the fundamental concept of graph edit distance can be regarded as one of the most flexible and adaptable graph matching models available [71–73].

Formally, given two graphs g and g' , the basic idea of graph edit distance is to transform the *source graph* g into the *target graph* g' using some *edit operations* for both nodes and edges (such as *insertions*, *deletions*, and *substitutions*). Note that other operations, such as *splitting*, *merging* or *edge contracting* have been proposed [74–78] (not necessarily in conjunction with graph edit distance). Splitting nodes means dividing a single node into two or more nodes, redistributing the edges and labels as required. Merging nodes, is an operation which combines two nodes into a single node, updating the corresponding edges and labels as needed. Edge contraction merges two adjacent nodes into a single node and thus removes the edge connecting them. These operations are very specialized and can be useful in certain situations, however they are not considered in the present thesis.

We denote the substitution of two nodes $v \in V$ and $v' \in V'$ by $(v \rightarrow v')$, the deletion of node $v \in V$ by $(v \rightarrow \varepsilon)$, and the insertion of node $v' \in V'$ by $(\varepsilon \rightarrow v')$. For edge edit operations we use a similar notation.

Definition 2.6 (Edit Path). A set $\{e_1, \dots, e_k\}$ of k edit operations e_i that transform a source graph g completely into a target graph g' is called an *edit path* $\lambda(g, g')$ between g and g' .

Let $\Upsilon(g, g')$ denote the set of all edit paths transforming g into g' while c denotes the cost function that measures the strength $c(e_i)$ of edit operation e_i . Clearly, between two similar graphs, there should exist an inexpensive edit path, representing low cost operations, while for dissimilar graphs an edit path with high cost is needed. The graph edit distance can now be defined as follows.

Definition 2.7 (Graph Edit Distance). Let $g = (V, E, \mu, \nu)$ be the source and $g' = (V', E', \mu', \nu')$ the target graph. The graph edit distance between g and g' is defined by

$$d_{\lambda_{\min}}(g, g') = \min_{\lambda \in \Upsilon(g, g')} \sum_{e_i \in \lambda} c(e_i) \quad , \quad (2.1)$$

Note that the edit operation of the edge structure is distinctly determined through operations performed specifically on the nodes. That is, it is sufficient that an edit path $\lambda(g, g')$ covers the nodes from V and V' only. Thus, we will assume that an edit path $\lambda(g, g')$ clearly outlines the correspondences identified between the nodes of graphs V and V' , while the edge edit operations are implicitly provided by these node correspondences.

It is evident that in the set $\Upsilon(g, g')$, there may be two or more edit paths with the same minimal cost $d_{\lambda_{\min}}(g, g')$. In other words, the minimal-cost edit path $\lambda_{\min}(g, g') \in \Upsilon(g, g')$ is not necessarily unique.

Note that it is also possible that only parts of the full edit path $\lambda(g, g')$ are needed. This *partial edit path* is defined as $\tau(g, g') = \{e_{(1)}, \dots, e_{(t)}\} \subseteq \lambda(g, g')$ with $t < k$ being the required number of edit operations.

Example 2.7. In Figure 2.6 a transformation is illustrated using an edit path $\lambda(g, g')$ between two undirected and labeled graphs g and g' (the labels are symbolized by node colors). The edit path is defined by

$$\lambda(g, g') = \{(0 \rightarrow a), (1 \rightarrow b), (2 \rightarrow c), (3 \rightarrow \varepsilon), (4 \rightarrow \varepsilon), (\varepsilon \rightarrow d)\} \quad .$$

This particular edit path implies the following edge edit operations

$$\begin{aligned} &\{((0, 1) \rightarrow (a, b)), ((0, 2) \rightarrow (a, c)), ((0, 3) \rightarrow \varepsilon), (1, 2) \rightarrow (b, c)), \\ &((2, 3) \rightarrow \varepsilon), ((3, 4) \rightarrow \varepsilon), (\varepsilon \rightarrow (a, d)), (\varepsilon \rightarrow (c, d))\} \quad . \end{aligned}$$

An example of a partial edit path $\tau(g, g')$ is given as well and is defined by

$$\tau(g, g') = \{(0 \rightarrow a), (1 \rightarrow b), (2 \rightarrow c)\} \quad .$$

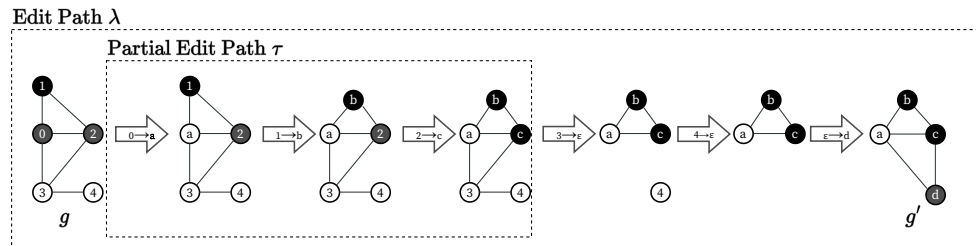


Fig. 2.6: An example of a transformation of graph g into g' using a complete edit path λ , as well as a partial edit path τ .

2.3.1 Cost Functions

The cost of a given edit operation $c(e_i)$ is of crucial importance for the performance of graph edit distance [79]. In some applications, for example, the label alphabet could be given by the set of all strings of arbitrary size over a finite set of symbols. In this case, a distance model for strings, as for instance, the *string edit distance* [80, 81], could be used to measure the cost of a substitution. Furthermore, if in a particular case prior knowledge about the labels is not available, automatic procedures can be used to learn the cost model from a set of sample graphs [82–84]. In this section, we define and explain several cost functions that are actually used in this thesis.

In the present thesis we employ graphs with either continuous or categorical labels or a mix thereof. This distinction is specifically important when it comes to cost functions, since a suitable cost function depends on the type of graph that is given. Initially, an abstract cost function can be defined.

Definition 2.8 (Abstract Cost Function). *Given two graphs $g = (V, E, \mu, \nu)$ and $g' = (V', E', \mu', \nu')$, an abstract cost function can be defined by*

$$\begin{aligned}
 c(u \rightarrow \varepsilon) &= \alpha \times \tau_{node} \\
 c((u, v) \rightarrow \varepsilon) &= (1 - \alpha) \times \tau_{edge} \\
 c(\varepsilon \rightarrow v') &= \alpha \times \tau_{node} \\
 c(\varepsilon \rightarrow (u', v')) &= (1 - \alpha) \times \tau_{edge} \\
 c(u \rightarrow u') &= \alpha \times S_1(u, u') \\
 c((u, v) \rightarrow (u', v')) &= (1 - \alpha) \times S_2((u, v), (u', v'))
 \end{aligned} \tag{2.2}$$

where $u, v \in V$, $u', v' \in V'$, $(u, v) \in E$, $(u', v') \in E'$ and $\tau_{node}, \tau_{edge} \in \mathbb{R}^+$ are positive constants representing the cost of a deletion/insertion of a node and edge, respectively³. The weighting parameter $\alpha \in [0, 1]$ controls the relative weight between the node and edge edit operations. The functions S_1 and S_2 are responsible for calculating the cost of a substitution between two nodes and two edges, respectively.

In case of unlabeled graphs, S_1 and S_2 are generally defined to be free of cost, and the insertion/deletion cost is set to $\tau_{node} = \tau_{edge} = 1$. In gen-

³Note that for the sake of symmetry, an identical cost for deletions and insertions is defined here, which could indeed be represented by two different values.

eral, however, the cost $c(e)$ of a particular edit operation e is defined with respect to the underlying label alphabets L_V and L_E . In this case, we usually differ between continuous and categorical node and edge labels. For continuous labels, i.e. $L_V, L_E = \mathbb{R}^n$, a *Minkowski* distance (e.g. the Euclidean distance) can be employed [43, 80, 85]. The present thesis employs the Euclidean version of the Minkowski distance for continuously labeled graphs. That is,

$$S_1(u, u') = \|\mu(u) - \mu'(u')\|_2$$

and

$$S_2((u, v), (u', v')) = \|\nu((u, v)) - \nu'((u', v'))\|_2$$

Observe that if the cost of substituting a node is greater than $2\tau_{node}$, it can be effectively replaced by a combination of deleting and inserting the involved nodes (this principle also applies to edges). This notion aligns with the fundamental idea that substitutions should be preferred over deletions and insertions to some extent [43].

In the case of categorical node labels, the label alphabet is defined by a finite set of n symbolic labels $L_{V/E} = \{\chi_1, \chi_2, \dots, \chi_n\}$. The present thesis employs a *Dirac function* for the substitution cost model, which returns zero when the involved labels are identical and a non-negative constant otherwise. Formally, the function $S_1(u, u')$ for a node substitution is defined via

$$S_1(u, u') = \begin{cases} 2\tau_{node} & \text{if } \mu(u) \neq \mu'(u') \\ 0 & \text{if } \mu(u) = \mu'(u') \end{cases}$$

and S_2 similarly for edges,

$$S_2((u, v), (u', v')) = \begin{cases} 2\tau_{edge} & \text{if } \nu((u, v)) \neq \nu'((u', v')) \\ 0 & \text{if } \nu((u, v)) = \nu'((u', v')) \end{cases}$$

2.3.2 Graph Edit Distance Computation

In order to compute the graph edit distance $d_{\lambda_{\min}}(g, g')$ usually a tree search algorithm is employed [86, 87]. A widely used A* based search technique using some heuristics is employed in [86]. The basic idea of A* based search methods is to organize the underlying search space as an ordered tree. The root node of the search tree represents the starting point of the search

procedure, inner nodes of the search tree correspond to partial solutions, and leaf nodes represent complete – not necessarily optimal – solutions. In case of graph edit distance computation inner nodes and leaf nodes correspond to partial and complete edit paths, respectively. Such a search tree is dynamically constructed at runtime by iteratively creating successor nodes linked by edges to the currently considered node in the search tree. A* is a best-first search algorithm which is *complete* and *admissible*, i.e. it always finds a solution if there is one and never overestimates the cost of reaching the goal [86].

Unfortunately, computing the exact graph edit distance using the above mentioned method exhibits exponential growth in relation to the number of nodes in the involved graphs. Formally, for graphs with m and n nodes we observe a time complexity of $O(m^n)$. This means that calculating the exact edit distance for large graphs becomes infeasible. In real-world applications, we can typically calculate the edit distance for graphs containing up to 12 nodes only. In order to counteract this problem and speed up the exact algorithm, the use of different heuristics is often employed [88].

In recent years several *approximate*, or *suboptimal*, algorithms for graph edit distance have been proposed [11, 12]. These algorithms offer polynomial, rather than exponential, run times. However, in contrast to optimal error-tolerant graph matching, approximate algorithms do not guarantee to find the global minimum of the matching cost, but only a local one [11]. Usually, this approximation is not very far from the global one, but there are no guarantees.

Many of the suboptimal algorithms offer cubic or quadratic time complexity with respect to the number of nodes of the involved graphs. Some approximations [89, 90] use a variant of the basic A* procedure that is based on *beam search* [91]. Instead of expanding all successor nodes in the search tree, only a fixed number s of nodes to be processed are kept at all times. Whenever a new partial edit path is added, only the s partial edit paths τ with the lowest costs $g(\tau) + h(\tau)$ are kept and the remaining partial edit paths are removed. This means that not the full search space is explored, but only those nodes that belong to the most promising partial matches are expanded. For similar graphs, it is clear that edit operations of an optimal path have low costs. Therefore, if only the partial edit paths with lowest costs are considered, we will obtain an edit path that is nearly optimal, which will result in a suboptimal distance close to the exact distance. For dissimilar graphs, the suboptimal distance will remain large. In summary, this method generally does not return the optimal edit path, but

only a suboptimal one.

In [92], a linear programming method for computing upper and lower bounds of the edit distance of graphs with unlabeled edges is reported. In [93, 94], a Bayesian perspective on graph edit distance is adopted, which builds upon the idea of probabilistic relaxation labeling [95], and iteratively applies edit operations to improve a maximum a posteriori criterion. The methods proposed in [96, 97] perform a randomized construction of initial mappings which is followed by a local search procedure. K-REFINE [98] is a local search based algorithm. Given one or more initial node maps, local search based algorithms explore fitly defined neighborhoods to find improved node maps that induce cheaper edit paths. K-REFINE improves REFINE [99] by not only considering binary swaps and computing the swap costs more efficiently.

In this thesis the algorithm *BP* [85] is employed for graph edit distance computation. BP is initially introduced as a unique heuristic for the calculation of the optimal graph edit distance based on a rapid assignment of the nodes [100].

Algorithm BP, introduced in [85], approximates the graph edit distance computation by reducing the quadratic assignment problem of the graph edit distance computation to an instance of a linear sum assignment problem (LSAP), which allows a much faster computation, without excluding graph types. The most prominent algorithms for LSAPs are *Munkres/Hungarian* [101, 102] or the algorithm of *Volgenant-Jonker* [103] (in the original implementation [85] the Munkres/Hungarian algorithm is used, however quite a few other algorithms exist [104–106]). The time complexity of the best-performing exact algorithms for LSAPs is cubic in the size of the problem.

Note that several other approaches exist that also reduce the graph edit distance problem to an LSAP. In [107] a faster version of the previously mentioned BP algorithm is proposed, called FBP. To make BP faster, a restriction on the edit costs is imposed so that the cost of an insertion plus a deletion of a node has to be lower or equal than the cost of a substitution of a node. By doing this, it can be ensured that the Munkres/Hungarian algorithm only needs to explore the first quadrant of the underlying cost matrix, which is composed of the first n rows and m columns, if we have two graphs $g = (V, E)$ and $g' = (V', E')$ with $|V| = n$ and $|V'| = m$.

In [108] the Hausdorff distance [109] is adapted for the graph edit distance. The concept combines the idea of an assignment edit distance, which aims to identify a correspondence between nodes and their local configura-

tion, with a more effective pairwise node matching technique. Each node of one graph is compared to each node of the other graph, similar to comparing subsets of a metric space by using the Hausdorff distance. Using the costs for deletions, insertions and substitutions for both nodes and edges, an optimal match is determined for each node individually. The resulting sum of all these costs yields a distance measure that is less than or equal to the graph edit distance. In [110] a ring based approximate graph edit distance is proposed, by using rings as a local structure. The rings are defined as a collection of nodes and edges that have a fixed distance from the root node.

Most of the currently available solvers for the LSAP require the pairwise costs to respect the triangle inequality [43]. In [111] an algorithm is proposed that does not need to respect the triangle inequality constraints and relies on the fact that node substitutions whose costs do not respect the triangle inequality can be factorized into removals and insertions. In [112] an approach to redefine the cost matrix, such that it always converges by using the Volgenant-Jonker LSAP algorithm is proposed. The Volgenant-Jonker algorithm is usually faster than the Munkres/Hungarian algorithm, but does not always converge using the cost matrices proposed in the BP or FBP algorithms.

Other approaches try to leverage the power of deep learning to find an approximation to the graph edit distance. In [113] the Hausdorff edit distance [108] is learned by means of a triplet loss neural network [114], while in [115] a deep learning based method is used, which also tries to recover the edit path.

Actually, any computation method for graph edit distance could be employed as basis in our framework, as long as it provides us with a valid edit path between two graphs. That is, the actual algorithm for graph matching does not crucially impact the rest of the proposed method of this thesis. We decide to use the graph edit distance algorithm BP throughout this thesis, since it is a widely used (and somewhat standard) algorithm for the graph edit distance approximation [100, 116, 117].

Graph Data Sets

3

Oh! Graph data sets, how
valuable you are, In the world of
big data, you truly are the star.

ChatGPT

This chapter presents the data sets for subsequent empirical investigations, offering readers a clear understanding of the data-driven foundation upon which the contribution of this thesis is built. It provides a comprehensive overview of the characteristics of the data sets. The data sets contain a diverse range of graph structures and properties that enable the evaluation of the proposed approaches.

First, in Section 3.1, the data sets consisting of graphs that model chemical compounds are presented. Then, in Section 3.2 the data sets that are based on protein structure graphs are discussed. Section 3.3 provides an overview of data sets from various other domains. Section 3.4 gives an overview of which data sets are employed in which experiment and which cost functions are used. Furthermore, a visualization of some data sets is given, to get an idea of how difficult it is to separate the individual classes.

3.1 Chemical Compound Graph Data Sets

Chemical compounds are a good example of data that can be easily represented as a graph. That is, a molecular composition comprised of atoms and covalent bonds can be described using a graph in an intuitive and direct manner by representing the atoms as nodes and the covalent bonds as edges. Nodes can be labeled with the relevant chemical symbols, whereas edges may be labeled by the valence of the linkage. Given the established

connection between a specific molecular compound's structure and its activity, employing a graph-based representation appears to be an appropriate data description methodology. This section describes the graph data sets used in this thesis that represent graphs based on chemical compounds.

3.1.1 AIDS

The *AIDS* data originates from data gathered by the National Cancer Institute (NCI), namely from the AIDS Antiviral Screen Database of Active Compounds [118]. The graph based data set stems from the IAM graph repository [119].

Starting 1999, the NCI conveys AIDS antiviral screen tests to discover chemical compounds that might be capable of inhibiting the HIV virus. The aim of these screen tests is to measure how strongly the compounds under consideration are able to protect human cells from infection by the HIV virus. The experiment is carried out in two stages. In a first experiment, those chemical compounds that are able to provide a protection from an HIV infection in at least half of the cases are selected for a second experiment. Chemical compounds that reproducibly provide a full protection from HIV in this second experiment are labeled *confirmed active* (CA), while compounds that are only able to protect cells from infection in at least 50% of the cases are labeled *moderately active* (MA). All of the remaining molecules are labeled *confirmed inactive* (CI). In the present thesis, the categories CA and CI are used. In Figure 3.1 [43] we can see an example of two molecules of the classes CA (a) and CI (b).

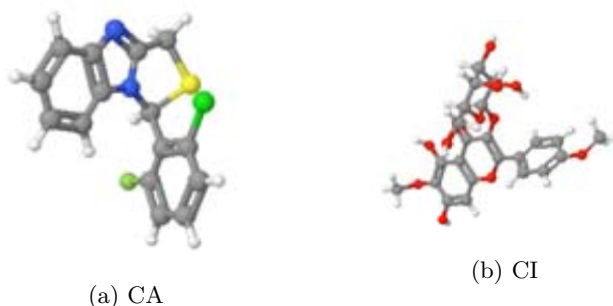


Fig. 3.1: Examples of molecules of both classes of the AIDS data set (figures stem from [43]).

Table 3.1: Summary of some metrics of the AIDS data set.

AIDS	
Graphs	2,000
Classes	2 (confirmed active, confirmed inactive)
Avg. V	15.7
Avg. E	16.2
Max. V	95
Max. E	103
Node labels	1 (Atom type (categorical)).
Edge labels	-

The nodes of the graphs from this data set represent the atoms, and the edges represent covalent bonds between the nodes. The nodes are labeled by *atom type*¹. In total the data set contains 2,000 graphs. In Table 3.1, some key metrics about the AIDS data set and in Figure 3.2 two example graphs are shown.

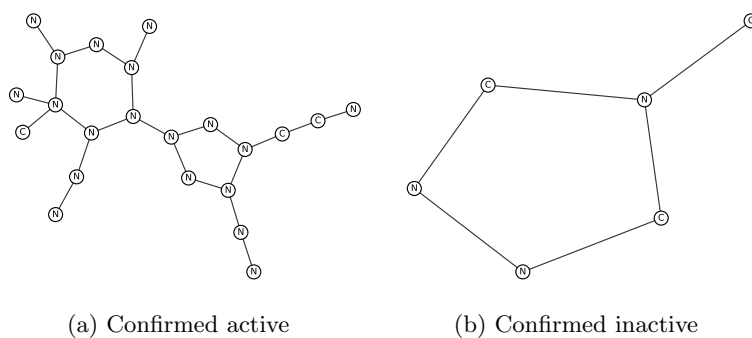


Fig. 3.2: Example graph for both classes of the AIDS data set.

¹Actually, on the original data set there are other labels, namely, the *charge* and the *x*- and *y*-coordinates. The edges contain the *valence* of the linkage. In the present thesis, we only consider the atom type on the nodes.

3.1.2 Mutagenicity

The *Mutagenicity* data stems from the *Chemical Carcinogenicity Research Information System* (CCRIS) database, which holds scientifically assessed test data for approximately 7,000 compounds. The data set was initially collected by Kazius et al. [120] and the graph based data set stems from the IAM graph repository [119].

Mutagenicity refers to the capacity of a chemical substance to induce mutations in DNA [120]. Mutations are changes in the genetic material (DNA or RNA) that can occur spontaneously or be induced by external factors. A mutagenic compound's reactivity with DNA can lead to the formation of DNA adducts or base deletions, causing significant distortion in the DNA structure [121]. As a result, mutagenic compounds pose a toxic threat to humans, and screening drug candidates for mutagenicity is a regulatory prerequisite for drug approval [120]. This data set contains molecules that are either mutagenic or not. In Figure 3.3 we can see an example of a *non-mutagenic* and a *mutagenic* molecule [120].

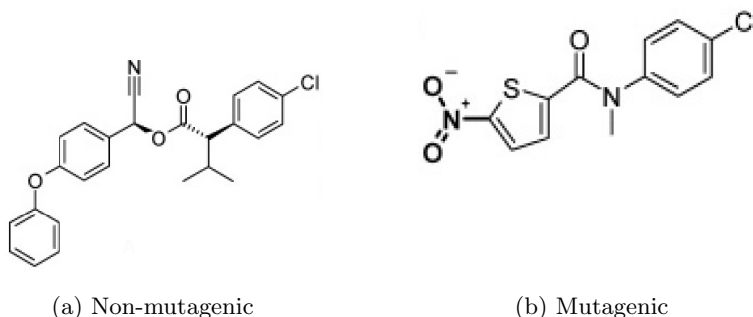


Fig. 3.3: Examples of molecules of both classes of the Mutagenicity data set (figures stem from [120]).

The nodes of the graphs from this data set represent the atoms, and the edges represent covalent bonds between the nodes. The nodes are labeled by *atom type*. The edges are unlabeled. In total the data set contains 4,337 graphs. In Table 3.2, we present some key metrics about the Mutagenicity data set and in Figure 3.4 we show two example graphs.

Table 3.2: Summary of some metrics of the Mutagenicity data set.

Mutagenicity	
Graphs	4,337
Classes	2 (mutagenic, non-mutagenic)
Avg. V	30.3
Avg. E	30.8
Max. V	417
Max. E	112
Node labels	1 (Atom type (categorical)).
Edge labels	-

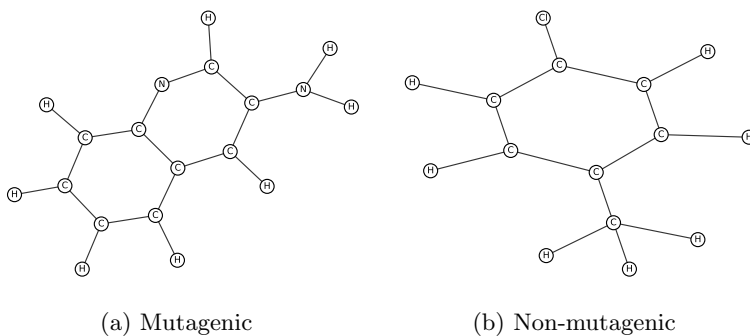


Fig. 3.4: Example graph for both classes of the Mutagenicity data set.

3.1.3 NCI

The *NCI* data originates from anti-cancer screens performed by the NCI and is derived from the PubChem website [122]. The graph based data set was initially prepared by Wale and Karypis [123, 124].

The primary goal of the screens is to identify potential anti-cancer compounds, determine their effectiveness against various types of cancer, and understand the underlying molecular mechanisms. The *NCI* data set measures molecules that contain activity for growth inhibition of non-small cell lung cancer and those that do not. As there is more than one screen available for a specific type of cancer, the screen with the highest number of compounds tested has been used [123].

The nodes of the graphs from this data set represent the atoms, and the

Table 3.3: Summary of some metrics of the NCI1 data set.

NCI1	
Graphs	4,110
Classes	2 (inhibiting, non-inhibiting)
Avg. V	29.87
Avg. E	32.3
Max. V	111
Max. E	238
Node labels	1 (Atom type (categorical)).
Edge labels	-

edges represent covalent bonds between the nodes. The nodes are labeled by *atom type*. The edges are unlabeled. In total the data set contains 4,110 graphs. In Table 3.3, we present some key metrics about the NCI1 data set and in Figure 3.5 we show two example graphs.

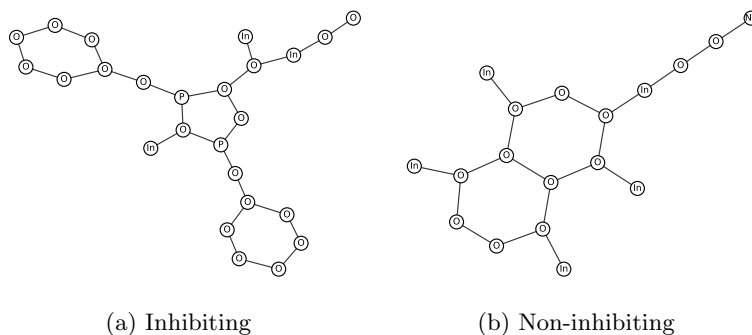


Fig. 3.5: Example graph for both classes of the NCI1 data set.

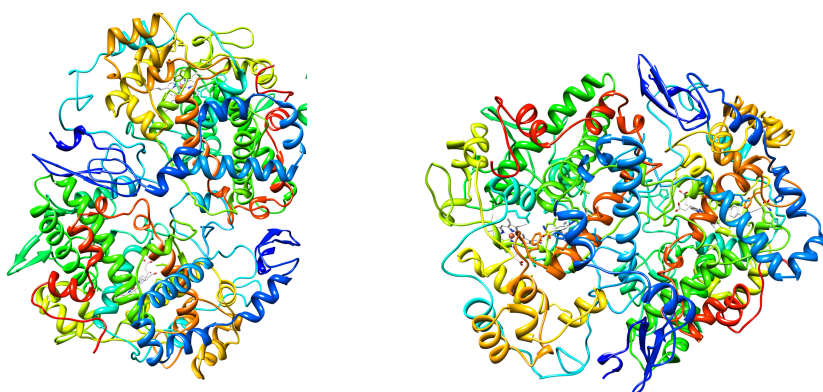
3.1.4 COX-2

The *COX-2* data originates from the work of a single research group and the graph based data set was initially prepared by Sutherland et al. [125].

The COX-2 data set contains cyclooxygenase-2 (COX-2) inhibitors with or without in-vitro activities against human recombinant enzymes [125]. COX-2 is an enzyme involved in the synthesis of prostanoids, which plays

a role in various physiological processes, including inflammation, pain, and fever [126–128]. COX-2 inhibitors are a class of drugs that target the COX-2 enzyme, reducing the production of pro-inflammatory prostaglandins and are commonly used as anti-inflammatory, analgesic, and antipyretic medications [129].

In Figure 3.6 we can see an example of uninhibited COX-2 [130] and an example of COX-2 in complex with a COX-2 selective inhibitor [126, 131]².



(a) Uninhibited COX-2 [130]

(b) Inhibited COX-2 [126, 131]

Fig. 3.6: Examples of uninhibited COX-2 (a) and COX-2 in complex with a COX-2 selective inhibitor (b).

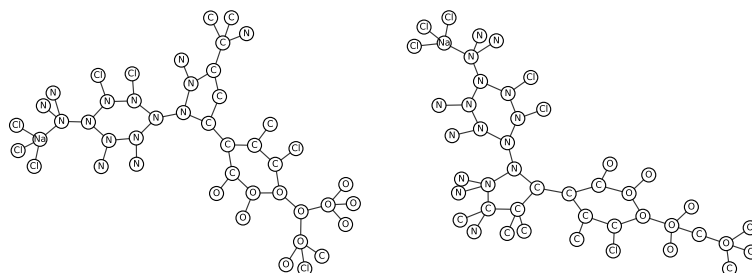
The activity is expressed as IC_{50} values (the concentration required to inhibit 50% of the enzymes activity) ranging from 1 nanometer to > 100 micrometers [125]. A 314 compound subset of these inhibitors has been studied and they used $pIC_{50} = 6.5$ as the threshold for classifying compounds as active or inactive [132]. The same threshold is used to generate this data set.

The nodes of the graphs from this data set represent the atoms, and the edges represent covalent bonds between the nodes. The nodes are labeled by *atom type*. The edges are unlabeled. In total the data set contains 467 graphs. In Table 3.4, we present some key metrics about the COX-2 data set and in Figure 3.7 we show two example graphs.

²The molecular graphics images were produced using the UCSF Chimera package from the Resource for Biocomputing, Visualization, and Informatics at the University of California, San Francisco (<https://www.cgl.ucsf.edu/chimera/>).

Table 3.4: Summary of some metrics of the COX-2 data set.

COX-2	
Graphs	467
Classes	2 (COX-2 inhibitory activity, No COX-2 inhibitory activity)
Avg. V	41.22
Avg. E	43.45
Max. V	56
Max. E	59
Node labels	1 (Atom type (categorical)).
Edge labels	-



(a) COX-2 inhibitory activity

(b) No COX-2 inhibitory activity

Fig. 3.7: Example graph for both classes of the COX-2 data set.

3.1.5 $PTC(MR)$

The $PTC(MR)$ data stems from the Predictive Toxicology Challenge [133], a competition organized in the early 2000s to stimulate research in the area of predictive toxicology. The graph data set $PTC(MR)$ was built by Kriege and Mutzel [28].

The goal of the challenge is to predict the carcinogenic potential of compounds based on their molecular structure. The specific $PTC(MR)$ challenge is about predicting whether compounds are carcinogenic to male rats (MR). The original data is in the form of Simplified molecular-input line-entry system (SMILES) strings [134] and during the conversion to graphs explicit hydrogen atoms are removed [28].

The nodes of the graphs from this data set represent the atoms, and the

Table 3.5: Summary of some metrics of the PTC(MR) data set.

PTC(MR)	
Graphs	344
Classes	2 (carcinogenic, non-carcinogenic)
Avg. V	14.29
Avg. E	14.69
Max. V	64
Max. E	71
Node labels	1 (Atom Type (categorical)).
Edge labels	-

edges represent covalent bonds between the nodes. The nodes are labeled by *atom type*³. In total the data set contains 344 graphs. In Table 3.5, we present some key metrics about the PTC(MR) data set and in Figure 3.8 we show two example graphs.

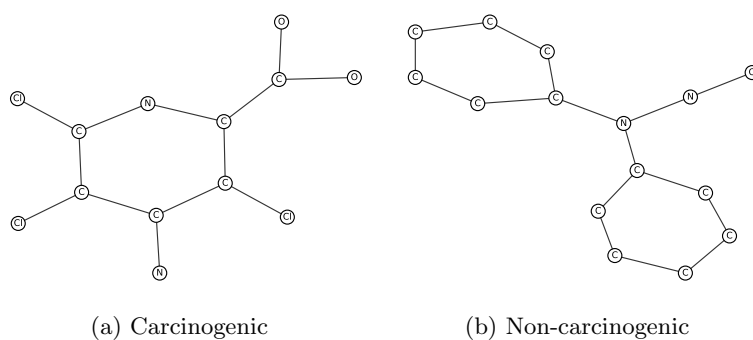


Fig. 3.8: Example graph for both classes of the PTC(MR) data set.

³Actually, on the original data set there are other labels. The edges are labeled by the *bond type*. In the present thesis, however, we only consider the atom type of the nodes, the labels on the edges are removed.

3.2 Protein Graph Data Sets

A protein is a large, complex molecule made up of long chains of smaller building blocks called amino acids. There are 20 different types of amino acids that can be combined in various sequences to create a vast array of proteins. Proteins play an important role in the structure, function, and regulation of the cells, tissues, and organs within all living organisms [135].

The sequence of amino acids in a protein is determined by the information encoded in an organism's DNA. Once the amino acids are assembled into a chain, the protein folds into a specific three-dimensional structure, which is crucial for its proper function. This folding process is primarily governed by the interactions between the amino acids, including hydrogen bonding, hydrophobic interactions, and electrostatic forces [136].

Protein secondary structure refers to the local three-dimensional arrangement of amino acids in a protein chain, which is primarily driven by hydrogen bonding between the peptide backbone atoms. The secondary structure elements act as the building blocks for the protein's overall three-dimensional conformation, known as the tertiary structure [135]. There are several Secondary Structure Elements (SSEs), namely *alpha helices*, *beta sheets* and *beta turns/omega loops* [136].

In this section several graph data sets of graphs that represent protein structures stemming from different applications are presented.

3.2.1 PROTEINS

The *PROTEINS* data set stems from protein files of the Protein Data Bank (PDB) [137] and was initially created by Dobson and Doig [138] and transformed to graphs by Borgwardt et al. [138, 139].

The data set contains graphs that hold information about the chemical properties, structure and sequence of proteins and is concerned with distinguishing *enzymes* from *non-enzymes*. An enzyme is a specific type of protein that acts as a biological catalyst, speeding up chemical reactions in living organisms [140]. While all enzymes are proteins, not all proteins are enzymes. Enzymes are a subset of proteins that facilitate and regulate biochemical reactions without being consumed or permanently altered in the process [141]. They work by lowering the activation energy required for a reaction to occur, thereby increasing the reaction rate. In contrast, proteins serve a wide range of functions, including structural support, transportation of molecules, cell signaling, immune response, and others. Enzymes

are just one of the many functional categories of proteins. In Figure 3.9 we can see one example of a protein that is an enzyme and one example of a protein that is not an enzyme (all taken from the PDB) [137, 142–145].

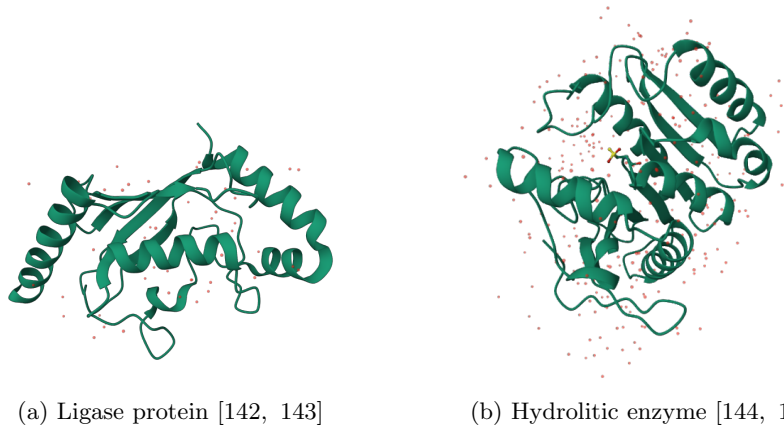


Fig. 3.9: Example of a protein that is a non-enzyme (a) and one that is an enzyme (b).

The proteins are transformed into undirected graphs (one graph representing one protein) with nodes and edges representing their structure, sequence, and chemical properties [139]. Nodes are labeled with their SSEs (helix, sheet or turn), according to the Amino Acid Index Database [146]. Furthermore one continuous label, which represents the length of the amino acid chain of the corresponding node, is also added to the node⁴. Edges link nodes that are either neighbors in the amino acid sequence or spatially adjacent within the protein structure, with each node being connected to its three closest spatial neighbors [139].

To summarize, the nodes represent the SSEs within the protein, and the edges represent neighboring SSEs. The nodes are labeled by their SSE, as well as the length of the corresponding amino acid chain. The edges contain the *type* of the edge. In total the data set contains 1,113 graphs. In Table 3.6, we present some key metrics about the PROTEINS data set and in Figure 3.10 we show two example graphs.

⁴Note that there is a version of the data set available, that adds multiple continuous labels to each node. However this version is not used in the present thesis.

Table 3.6: Summary of some metrics of the PROTEINS data set.

PROTEINS	
Graphs	1,113
Classes	2 (enzyme, non-enzyme)
Avg. $ V $	39.06
Avg. $ E $	72.82
Max. $ V $	620
Max. $ E $	1,049
Node labels	2 (SSE (helix, sheet, loop) (categorical) and amino acid length (continuous)).
Edge labels	-

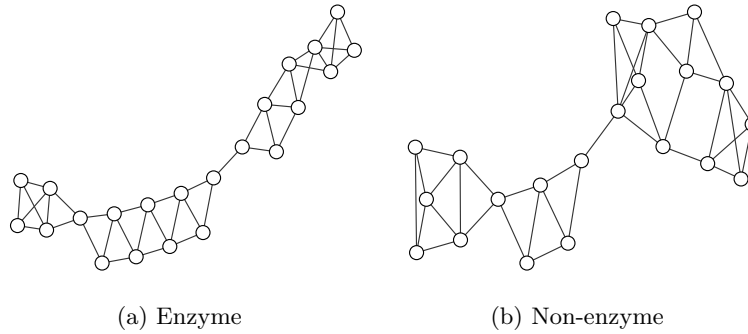


Fig. 3.10: Example graph for both classes of the PROTEINS data set.

3.2.2 ENZYMES

The *ENZYMES* data set also stems from proteins of the PDB [137] and is labeled with the corresponding enzyme class by using the BRENDA enzyme database [140] and transformed to graphs by Borgwardt et al. [139].

The *ENZYMES* data set contains graphs that are labeled with EC (Enzyme Commission) classes. These are a classification system for enzymes based on the chemical reactions they catalyze. The system was established by the *Enzyme Commission* of the *International Union of Biochemistry and Molecular Biology* (IUBMB)⁵. Enzymes are assigned a unique EC number,

⁵<https://iubmb.org/>

which consists of four components separated by periods (e.g., EC 1.1.1.1). In total there are six classes, namely Oxidoreductases (EC1), Transferases (EC2), Hydrolases (EC3), Lyases (EC4), Isomerases (EC5), Ligases (EC6). Two proteins having the same EC number catalyze the same reaction. For example, proteins from the EC1 class are Oxidoreductases. These enzymes catalyze oxidation-reduction reactions, where one molecule is oxidized while another is reduced. They are for example involved in electron transfer processes. In Figure 3.11 we can see one example of each EC class (all taken from the PDB) [137, 147–157].

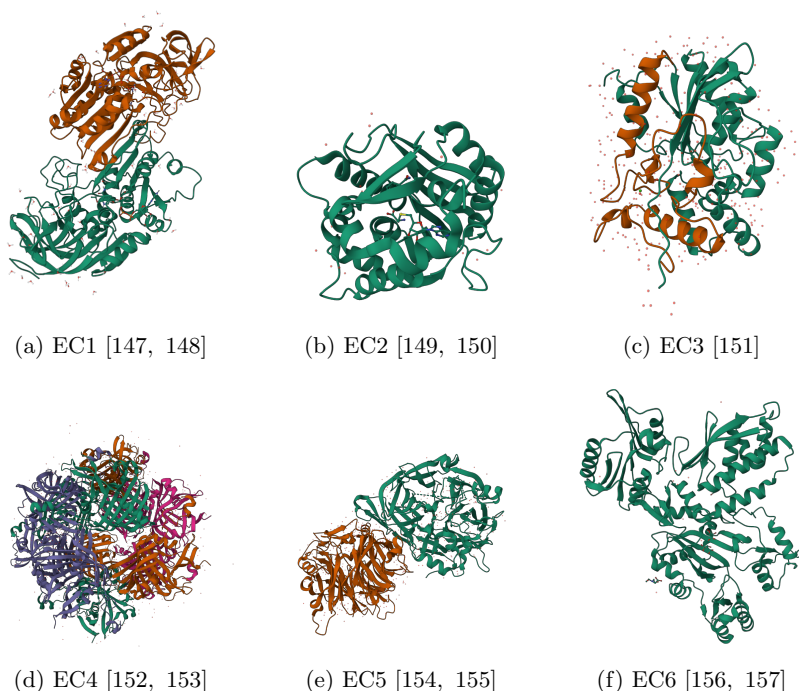


Fig. 3.11: Examples of enzymes of the top level classes EC1, EC2, EC3, EC4, EC5 and EC6.

The proteins are converted into graphs the same way, as described in Section 3.2.1, with one difference, viz. more continuous labels, than just the length of the amino acid chain are added to the node. The first node label is categorical which represents the SSEs (helix, sheet or turn). In addition to this, the distance between the central carbon atoms of the first and last

Table 3.7: Summary of some metrics of the ENZYMES data set.

ENZYMES	
Graphs	600
Classes	2 (EC1, EC2, EC3, EC4, EC5, EC6)
Avg. V	32.63
Avg. E	62.14
Max. V	126
Max. E	149
Node labels	18 (SSE (helix, sheet, loop) (categorical), amino acid length, 3d-length, van der Waals volume, hydrophobicity, polarity, polarizability and the corresponding 3-bin distributions (continuous)).
Edge labels	-

residue is measured in angstroms and added as a label (3d-length). Next, chemical information like the van der Waals volume [158], hydrophobicity [159], polarity [160] and polarizability [161] of the corresponding SSE are added. Furthermore, each node is labeled with the total number of residues with low, medium or high normalized van der Waals volume values (denoted *3-bin* distribution). The same 3-bin distribution is calculated for hydrophobicity, polarity and polarizability [139]. The edges are labeled in a similar way as described in 3.2.1.

To summarize, the nodes represent the SSEs within the protein, and the edges represent neighboring SSEs. The nodes are labeled by their SSE, as well as amino acid length, 3d-length, van der Waals volume, hydrophobicity, polarity, polarizability and the corresponding 3-bin distributions. The edges contain the *type* of the edge. In total the data set contains 600 graphs. In Table 3.7, we present some key metrics about the ENZYMES data set and in Figure 3.12 we show six example graphs.

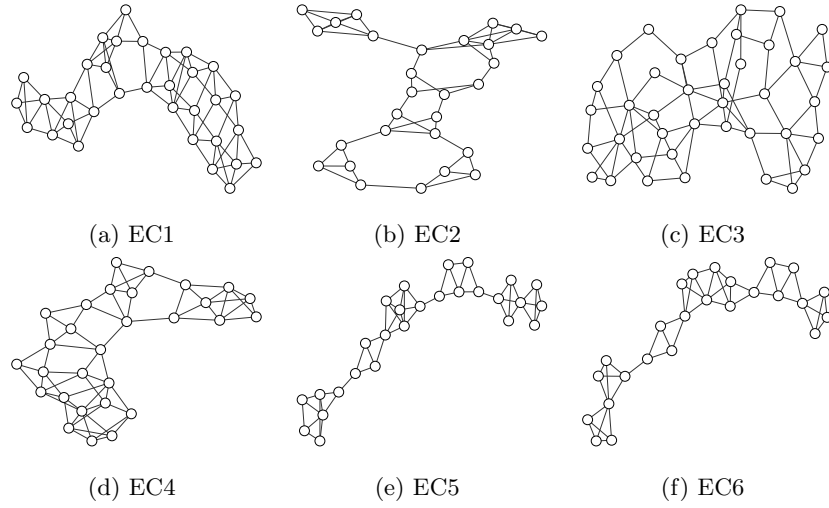


Fig. 3.12: Example graph for each of the six classes of the ENZYMES data set.

3.3 Data Sets From Various Domains

3.3.1 *Letter*

The *Letter* data set stems from the IAM graph repository [119]. It contains graphs that represent artificially distorted letter line drawings of 15 different letters that consist of straight lines (A, E, F, H, I, K, L, M, N, T, V, W, X, Y, Z) [62, 119]. In Figure 3.13 we can see examples of manually constructed letters.



Fig. 3.13: Examples letters (A, E, F, H, I, K, L, M, N, T, V, W, X, Y, Z) [119, 62].

Table 3.8: Summary of some metrics of the Letter data set.

Letter	
Graphs	2,250
Classes	15 (A,E,F,H,I,K,L,M,N,T,V,W,X,Y,Z)
Avg. V	4.67
Avg. E	4.50
Max. V	9
Max. E	9
Node labels	2 (x , y -coordinates (continuous)).
Edge labels	-

The letters are transformed into graphs, by representing the end points of the individual lines as nodes and the lines themselves as edges. The distortions consist of random removals, insertions, and displacements of the nodes and their corresponding edges [119]. The distortions are added at low, medium, and high levels to the letters, yielding three different data sets. In the present thesis, we use only the data set with the highest distortion level.

The nodes are labeled with their corresponding x - and y -coordinates. The edges are unlabeled. In total the data set contains 2,250 graphs. In Table 3.8, we present some key metrics about the Letter data set and in Figure 3.14 we show 15 example graphs.

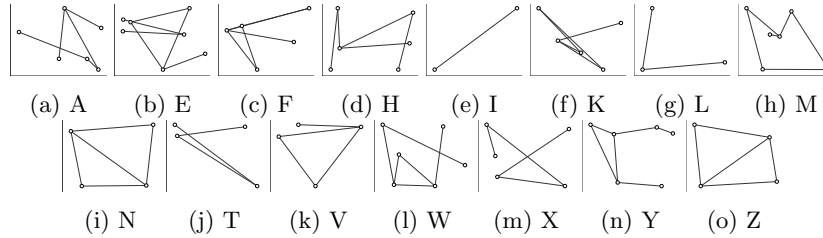


Fig. 3.14: Example graph for each of the 15 classes of the Letter data set.

Table 3.9: Summary of some metrics of the IMDB data set.

IMDB	
Graphs	1,000
Classes	2 (Action, Romance)
Avg. V	19.77
Avg. E	96.53
Max. V	136
Max. E	1,249
Node labels	-
Edge labels	-

3.3.2 IMDB

The *IMDB Binary* data (denoted as *IMDB* in the following) stems from the Internet Movie Database (IMDB) [162] and the graph data set was created by Yanardag and Vishwanathan [163].

The IMDB data set is based on collaboration graphs [164] of different actors or actresses for the *action* and *romance* movie genres. Based on these collaboration graphs, individual ego-networks are derived for different actors. Ego-networks are graphs centered around a specific actor or actress, including their direct collaborators [165]. So each graph represents the ego-network of a specific actor or actress, which is based on the specific movie genre (action or romance). This results in graphs where the nodes represent actors or actresses as nodes, and edges are formed if they appear in at least one movie of the corresponding genre together. It is important to note that a movie can belong to both genres at the same time, and if that is the case, the authors of the data set discarded the romance genre, if the movie already belonged to the action genre [163]. The goal of the data set is to label each ego-network with the genre it belongs to.

To summarize, nodes represent the actors or actresses, and the edges represent co-occurrences between them. The nodes and the edges are unlabeled. In total the data set contains 1,000 graphs. In Table 3.9, we present some key metrics about the IMDB data set and in Figure 3.15 we show two example graphs.

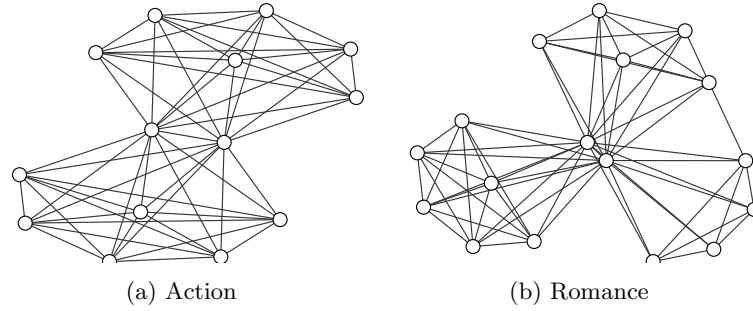


Fig. 3.15: Example graph for both classes of the IMDB data set.

3.3.3 Cuneiform

The *Cuneiform* data set stems from Kriege et al. [166] and has been generated using an approach of Fisseler et al. [167].

Cuneiform script was developed by the ancient Sumerians around 3000BC in Mesopotamia (modern-day Iraq) and was used for several thousand years by various cultures in the region, including the Akkadians, Babylonians, and Assyrians and is one of the oldest writing systems in the world [168]. The script consists of wedge-shaped marks usually written on clay tablets using a stylus, hence the name “cuneiform,” which comes from the Latin word “cuneus”, meaning “wedge” [168]. In Figure 3.16 we see an example of a Cuneiform tablet from the British Museum [169].



Fig. 3.16: Cuneiform Tablet that dates back to around 3000BC, on display in the British Museum [169].

The data set contains graphs that represent 30 different Hittite Cuneiform signs (tu, ta, ti, nu, na, ni, bu, ba, bi, zu, za, zi, su, sa, si, hu, ha, hi, du, da, di, ru, ra, ri, ku, ka, ki, lu, la, li), obtained from nine cuneiform tablets, provided by the Hethitologie-Portal [170]. Each cuneiform sign consists of tetrahedron shaped markings (wedges). In Figure 3.17 we can see a visualization of the 30 different signs used in the data set. The signs consist of visually very distinctive wedge constellations as well as visually very similar signs.

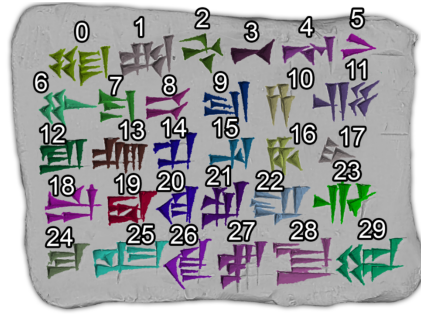


Fig. 3.17: Model of a Cuneiform tablet, with 30 different Hittite Cuneiform signs [166].

The resulting graphs represent each wedge of the sign by four nodes, that are categorically labeled by their point (depth, tail, right, left) as well as the type of the glyph (vertical, horizontal, “Winkelhaken”) and continuous labels on the nodes that represent the spatial coordinates of the wedge [166]. There are two types of edges in the data set. The first edge, connects the four parts of a wedge together, and the second type of edge are additional edges, between all pairs of depth nodes. These edges are further used to compare relative position of wedges, hence they are referred to as arrangement edges. In the end, the label of an edge indicates whether it is a wedge connecting edge, or an arrangement edge [166].

In summary, the nodes represent parts of the wedges (four nodes represent one wedge), and the first type of edge, connects the wedge parts. The second type of edge is the arrangement edge, which is used to compare positions of the wedges. The nodes are labeled with their *point* as well as the *glyph type*. The edges state whether it is a wedge *connecting* edge or an *arrangement* edge. In total the data set contains 267 graphs. In Table 3.10, we present some key metrics about the Cuneiform data set and in

Table 3.10: Summary of some metrics of the Cuneiform data set.

Cuneiform	
Graphs	467
Classes	30 (tu, ta, ti, nu, na, ni, bu, ba, bi, zu, za, zi, su, sa, si, hu, ha, hi, du, da, di, ru, ra, ri, ku, ka, ki, lu, la, li)
Avg. V	21.27
Avg. E	44.80
Max. V	36
Max. E	90
Node labels	2 (Point (depth, tail, right, left) and glyph type (vertical, horizontal, “Winkelhaken”) (both categorical)).
Edge labels	1 (Type (wedge, arrangement)).

Figure 3.18 we show 30 example graphs.

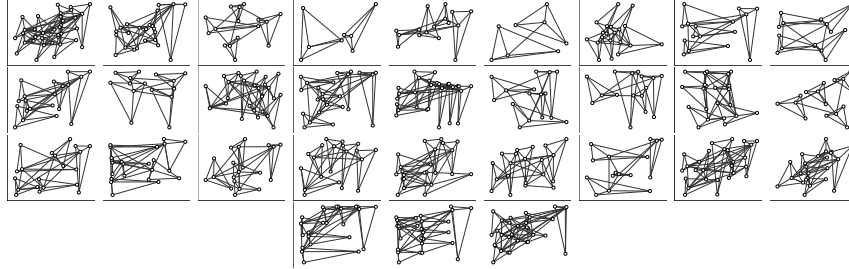


Fig. 3.18: Example graph for each of the 30 classes of the Cuneiform data set. Starting from the top left tu, ta, ti, nu, na, ni, bu, ba, bi, zu, za, zi, su, sa, si, hu, ha, hi, du, da, di, ru, ra, ri, ku, ka, ki, lu, la, li.

3.3.4 *Synthie*

The *Synthie* data set was built by Morris et al. [171] and contains synthetically created graphs.

The algorithm for the creation of the data set can be roughly described as follows:

Table 3.11: Summary of some metrics of the Synthie data set.

Synthie	
Graphs	400
Classes	4 ($C_1(A), C_1(B), C_2(A), C_2(B)$)
Avg. $ V $	95.00
Avg. $ E $	172.93
Max. $ V $	100
Max. $ E $	212
Node labels	15 (Algorithmically generated vector of real-valued numbers (continuous)).
Edge labels	-

- Generate two Erdős-Rényi graphs [172, 173], g_1 and g_2 . This model generates graphs with randomly connected nodes.
- Generate two seed sets S_1 and S_2 for g_1 and g_2 respectively of 200 graphs by randomly adding or deleting 25% of the edges of g_1 and g_2 . Graphs that are connected were obtained by randomly sampling 10 of the seeds and randomly adding edges in the resulting graphs.
- The two graph classes C_1 and C_2 are created by randomly selecting seeds from S_1 with $p = 0.8$ and from S_2 with $p = 0.2$ for C_1 and with the probabilities flipped for C_2 .
- Two sets A and B of real-valued vectors with 15 labels are created by using the approach described by Guyon [174].
- Both classes C_1 and C_2 are subdivided into two classes each, by randomly adding labels from A and B onto the nodes (using seeds from S_1 and S_2 , respectively).

In summary, the nodes and edges are part of algorithmically generated graphs. The nodes contain 15 real-valued labels, and edges are unlabeled. In total the data set contains 400 graphs. In Table 3.11, we present some key metrics about the Synthie data set and in Figure 3.19 we show 4 example graphs.

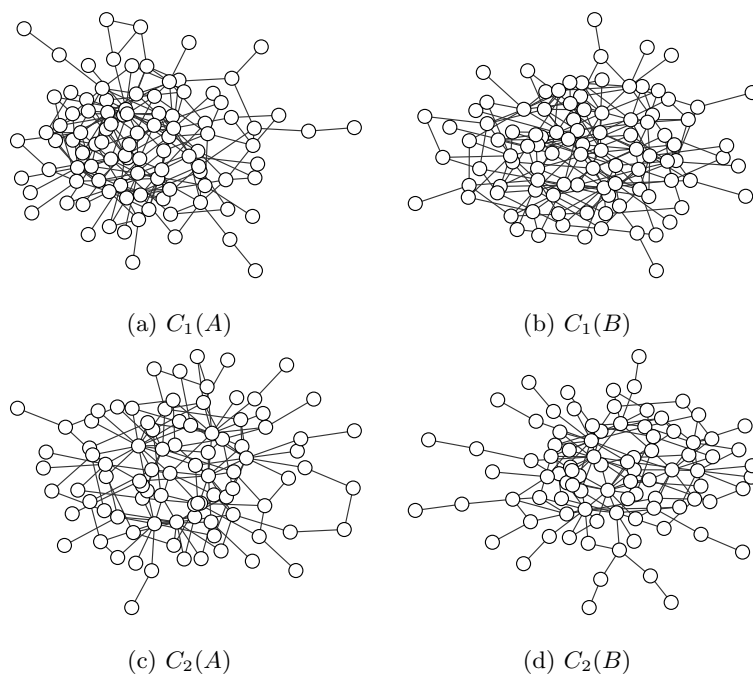


Fig. 3.19: Example graph for each of the four classes of the Synthie data set.

3.4 Applications and Cost Functions

The described data sets are used in several experiments throughout this thesis (especially in Chapters 4, 5 and 6). The version of the data sets that was finally used is taken from the TUdataset database [175]. Table 3.12 provides an overview over which data set is used in which chapter and for which method.

The NCI1, COX-2 and PTC(MR) data sets are used for all quantitative experiments. The usage of these data sets has developed over time with the several preliminary publications, as they have proven to pose difficult to classify. In addition, some data sets are used for specific purposes. For example, the PROTEINS and ENZYMES have been added for the MCS approximation experiments, because in these experiments specifically data sets with categorical labels are needed. Furthermore, for the graph augmentation experiments, Cuneiform and Synthie have been added, to specifically test the augmentation process on data sets with several continuous node

Table 3.12: Summary of which data sets are used in which chapters, and by which methods.

Chapter	Method	Data Sets
4	Distance based classifier	AIDS, Mutagenicity, NCI1, IMDB, COX-2, PTC(MR), Letter
4	Embedding based classifier	AIDS, Mutagenicity, NCI1, IMDB, COX-2, PTC(MR), Letter
5	Qualitative evaluation	AIDS, Mutagenicity
5	MCS approximation	AIDS, Mutagenicity, NCI1, COX-2, PTC(MR), PROTEINS, ENZYMES
6	Graph augmentation	AIDS, Mutagenicity, NCI1, COX-2, PTC(MR), Cuneiform, Synthie
6	Ensemble Classifier	NCI1, COX-2, PTC(MR), Cuneiform, Synthie

labels.

As described in Section 2.3.1, there are several cost functions employed in the present thesis. For the data sets in this thesis, we differ between four graph types and cost functions.

- (1) Graphs with **categorical** node labels use the Dirac distance for the substitution cost function S_1 .
- (2) Graphs with **continuous** node labels use the Euclidean distance for S_1 .
- (3) Graphs with **categorical and continuous** node labels use the Euclidean distance for S_1 . To obtain one vector of continuous labels, the categorical label is one-hot encoded.
- (4) **Unlabeled** graphs use the abstract cost function with substitutions S_1 and S_2 being free of cost.

Each cost function requires the parameters τ_{node} , τ_{edge} , α as well as a proper definition the substitution functions S_1, S_2 for nodes and edges, respectively. Parameter α is optimized in all experiments described in Chapters 4, 5 and 6. Parameters τ_{node} and τ_{edge} are always set to 1. In Table 3.13 an overview over the different data sets and the used cost functions is presented.

Figure 3.20 shows a visualization of three example data sets to give an idea of how difficult it is to separate the classes of the individual data

Table 3.13: Summary of the cost functions used for the individual data sets.

Graph Category	Cost Function S_1	Data Sets
(1) Categorical	Dirac	AIDS, Mutagenicity, NCI1, COX-2, PTC(MR)
(2) Continuous	Euclidean	Letter, Synthie
(3) Categorical and continuous	Euclidean (One-hot)	PROTEINS, ENZYMES, Cuneiform
(4) Unlabeled	Free of cost	IMDB

sets. To this end, the *t-distributed stochastic neighbor embedding* (T-SNE) [176, 177] as described in [178] is used. The different classes are represented by different colors. In Figure 3.20 (a) we can see a visualization of the AIDS data set, which is the only data set where the classes are clearly separable. For the data sets Mutagenicity, NCI1, PROTEINS, Letter and IMDB the classes are more difficult to separate. Figure 3.20 (b) shows as an example the PROTEINS data set. The classes of the COX-2, PTC(MR), ENZYMES, Cuneiform and Synthie data sets are not easily separable, as shown in Figure 3.20 (c) using the PTC(MR) data set as an example. In Appendix A a T-SNE visualization can be found for all data sets.

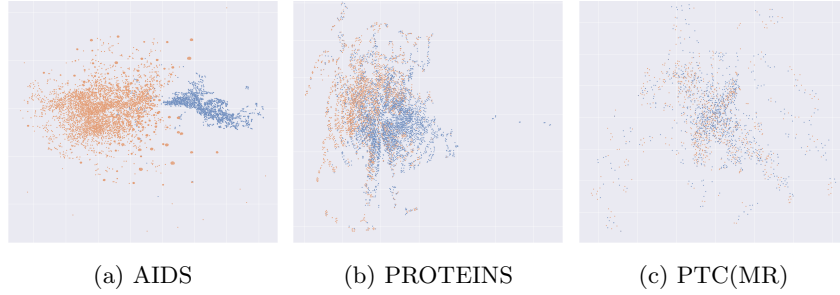


Fig. 3.20: T-SNE visualization of three data sets of different separability. The AIDS data set is easily separable, the PROTEINS data set is of medium separability and the PTC(MR) classes are difficult to separate. The individual colors represent the individual classes of the data set.

Matching-Graphs for Graph Classification

4

Matching-graphs unfold with a tale of lore. Born of edit distance, in a dance, unwrapping insights, through the embedding's lens, the journey of the matching-graph commences hence.

ChatGPT

4.1 Introduction

In the last four decades a huge number of procedures for *graph matching* have been proposed in the literature [11, 12]. Graph matching is typically used for quantifying graph proximity. As formally already described in Chapter 2, the *Graph edit distance* [34], introduced about 40 years ago, is still recognized as one of the most flexible and robust graph matching models available. In contrast with many other distance measures (e.g. *graph kernels* [16] or *graph neural networks* [17]), graph edit distance offers more information than merely a dissimilarity score, viz. the information which subparts of the underlying graphs actually match with each other (known as *edit path*). To date, we see no substantial research that exploits this particular knowledge as meta-information for reasoning about graphs and/or classifying graphs.

In this chapter, we will introduce and describe the main novelty of this thesis, namely the *matching-graph*. To this end, we propose a specific encoding of matching information derived from graph edit distance in the data structure of the matching-graph. In principle, the proposed procedure

first computes the graph edit distance for several pairs of labeled training graphs (stemming from the same class). Based on the matching found, a meta-graph is eventually created, which formalizes the corresponding parts of the two graphs. The rationale of these matching-graphs is to formalize the stable cores of specific classes of graphs. It is our hypothesis that the information captured in the resulting matching-graphs offers the potential to achieve both a better understanding of the regularities and stable parts of a given class and ultimately improve the matching quality of unknown patterns (by focusing on these stable cores of the graphs during the matching process, for instance).

Note that the present chapter is based on three conference papers [35–37] and one journal paper [179]. In [35] we describe the novel concept of matching-graphs for the first time and confirm the potential usefulness of these graphs in some initial experiments. In [36] we extend our initial idea by iteratively creating sets of matching-graphs on the basis of already existing matching-graphs. Finally, in [37] we propose to embed graphs into a vector space with the aid of matching-graphs. In [179] we combine the proposed methods in an overarching framework, give a more thorough and detailed description of the individual methods, and substantially extend both the methods and the empirical evaluation.

The remainder of the chapter is organized as follows. Section 4.2 discusses the related work. In Section 4.3 we describe in detail the first of two major building blocks of our complete framework (illustrated in Figure 4.1). In particular, we define how a set \mathcal{M} of initial matching-graphs is built from arbitrary sets of graphs \mathcal{G} and introduce two approaches to post-process the initial matching-graphs. The first approach – described in Section 4.3.1 – condenses the set of matching-graphs by selecting a useful set of matching-graphs from the raw set. The second approach – described in Section 4.3.2 – is somehow complementary to the first approach, since this method iteratively increases the initial set of matching-graphs. In Section 4.4, we describe the second building block of our novel framework, viz. the classification with matching-graphs. In particular, we introduce two conceptually different strategies to use the matching-graphs in order to solve classification problems. The first approach makes use of the matching-graphs in a distance-based classifier (detailed in Section 4.4.1). The second classifier is built on a graph embedding that heavily relies on the matching-graphs (detailed in Section 4.4.2).

As shown in Figure 4.1 we primarily use the matching-graphs selected with the method described in Section 4.3.1 for the distance-based classi-

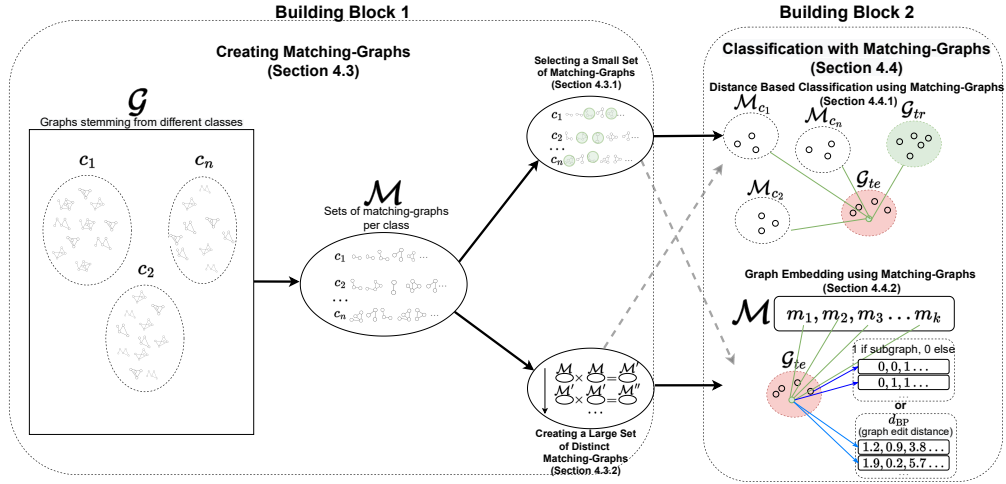


Fig. 4.1: The proposed framework consists of two major building blocks. The first is about creating matching-graphs (detailed in Section 4.3) and the second is about using the matching-graphs for classification (detailed in Section 4.4).

fier. Likewise, the matching-graphs which are iteratively created by means of the method described in Section 4.3.2 are used for the graph embedding classifier. One could, however, also combine the method described in Section 4.3.1 with the classification method from Section 4.4.2 and, conversely, the method from Section 4.3.2 with the classification method from Section 4.4.1 (shown with dashed lines in Figure 4.1). Although possible, we actually follow only those combinations that are connected with drawn lines (mainly for the sake of conciseness). The background and rationale for this decision follow in the corresponding subsections.

Eventually, in Section 4.5, we conduct a thorough experimental evaluation of our framework. That is, we empirically confirm that the matching-graphs are actually significant substructures of their classes and that our approach is able to improve the classification accuracy of various related systems. Finally, in Section 4.6, we conclude this chapter and discuss potential ideas for future work.

4.2 Related Work and Broader Perspective

The concept of matching-graphs actually requires a graph matching procedure – the task of identifying similar substructures in two graphs. We make use of graph edit distance [34] for this basic task (outlined above). Over the years, however, several other dissimilarity measures for graphs have been proposed [11, 12]. They range from *Spectral Methods* [21], over *Graduated Assignment Algorithms* [180], to *Expectation Maximization Algorithms and Continuous Optimization Algorithms* [24]. Some of the most prominent graph matching algorithms are *graph kernels*. A seminal contribution in the field are kernels that are based on the analysis of walks or paths in graphs. These kernels measure the similarity of two graphs by the number of equal (or at least similar) walks or paths in the underlying graphs [16]. In [13] a second class of kernels for graphs with discrete labels is introduced. This class of kernels is based on the 1-dimensional Weisfeiler-Lehman, or color refinement, algorithm. Since this contribution, several extensions and adaptations of this idea have been proposed [63].

A further prominent class of graph kernels is based on the work on convolution kernels, which provide a general framework for dealing with complex objects that consist of simpler parts. In particular, convolution kernels infer the similarity of two objects from the similarity of their parts (e.g., nodes, subgraphs, or trees [16]).

Graph embedding approaches can actually also be interpreted as graph kernels. In [181], for instance, a graph g is represented by a vector that counts the number of times certain subgraphs occur in g , while the *Subgraph Matching Kernel* [28] and *Graphlet Kernel* [67] both count the number of matchings between subgraphs of fixed sizes in two graphs. In [182], a graph is represented based on its dissimilarities to certain prototypes.

Besides the strong dependency of our novel matching-graphs on a specific graph matching procedure, we also observe a certain similarity of our novel concept with the idea of *Frequent Subgraph Mining (FSM)* [183]. FSM focuses on the identification of frequent subgraphs within a set of graphs. In particular, in FSM, one aims at extracting all subgraphs from a given set of graphs that occur more often than a specified threshold. We observe two main categories in FSM, viz. *Apriori-based approaches* [184] and *Pattern-growth approaches* [185]. The apriori-based methods start with frequent nodes and proceed to grow subgraphs by using a *Breadth First Search* strategy. That is, before they continue to find graphs of size $k + 1$ these approaches first search for all frequent graphs of size k . Pattern-growth ap-

proaches, on the other hand, work by using a *Depth First Search* strategy, where one graph is extended until all frequent supergraphs of this graph are found.

4.3 Creating Matching-Graphs

The general idea of *matching-graphs* – as proposed in [35, 38] – is to extract information on the matching of pairs of graphs in a new data structure that in turn encodes the corresponding parts of the two graphs.

Formally, we assume k sets of training graphs $G_{\omega_1}, \dots, G_{\omega_k}$ stemming from k different classes $\omega_1, \dots, \omega_k$. We formalize the information on the matching of two graphs $g = (V, E, \mu, \nu)$, $g' = (V', E', \mu', \nu')$ – stemming from the same class ω_l – in a graph denoted by $m_{g \times g'}$.

Basically, a matching-graph $m_{g \times g'}$ should represent both nodes and edges of g and g' that have been matched under the usage of some particular graph matching algorithm. In our scenario, matching-graphs $m_{g \times g'}$ are created according to the following procedure.

For all pairs of graphs stemming from the same class ω_l , the graph edit distance is computed by means of algorithm BP [85] (keep in mind, that any other approximation that yields a valid edit path could be used. For details see Chapter 2). Hence, we obtain a (sub-optimal) edit path $\lambda(g, g')$ for each pair of graphs g, g' . For each edit path $\lambda(g, g')$, two matching-graphs $m_{g \times g'}$ and $m_{g' \times g}$ are built (for the source and the target graph g and g' , respectively). To this end, all nodes of g and g' that are actually substituted in edit path $\lambda(g, g')$ are added to $m_{g \times g'}$ and $m_{g' \times g}$, respectively. Vice versa, all nodes that are deleted in g or inserted in g' are neither considered in the two matching-graphs.

In preliminary experiments we observe that isolated nodes might occur in the resulting matching-graphs. Although many graph matching algorithms can actually handle isolated nodes, we still remove them from our matching-graphs. The rationale for this heuristic is that we aim at building small and robust cores of the graphs with nodes that are actually connected to at least one other node in the formal substructure.

The question remains how to handle the edges of the involved graphs g, g' in the resulting matching-graphs $m_{g \times g'}$ and $m_{g' \times g}$. Clearly, if a node is not included in the matching-graph (since it was either deleted or inserted in the underlying edit path), the incident edges of this node will not be included in the resulting matching-graph as well. Edges that connect two

substituted nodes, however, can be included in the matching-graphs. We propose two different strategies for edge handling.

- (1) *No Pruning*: If two nodes $u, v \in V$ of a source graph g are substituted with nodes $u', v' \in V'$ in a target graph g' and there is an edge $(u, v) \in E$ available, (u, v) is actually included in the matching-graph $m_{g \times g'}$ regardless whether or not edge (u', v') is available in E' . Hence, in this case *no pruning* is applied to the edges.
- (2) *Pruning*: We assume the same scenario as above. However, edge (u, v) is included in the matching-graph $m_{g \times g'}$ if, and only if, there is an edge (u', v') available in E' . Hence, in cases where no corresponding edge can be found in the other graph, the edge is actually *pruned*.

Formally, a matching-graph $m_{g \times g'} = (V_{g \times g'}, E_{g \times g'})$ is defined as

- $V_{g \times g'} = \{v \in V : (v \rightarrow v') \in \lambda(g, g') \text{ and } v' \in V'\}$
- Unpruned: $E_{g \times g'} = \{E \cap (V_{g \times g'} \times V_{g \times g'})\}$
- Pruned: $E_{g \times g'} = \{E \cap E' \cap (V_{g \times g'} \times V_{g \times g'})\}$

For the matching-graph $m_{g' \times g}$ the definition is similar to $m_{g \times g'}$, but the roles of g and g' have to be exchanged. From a broader perspective, the novel matching-graphs can be interpreted as a generalization of the concept of a *common subgraph* [186]. In its original definition, a common subgraph of two graphs consists of nodes which occur identically in the both graphs. In our novel data structure, a node is incorporated whenever the corresponding node is actually substituted with another node w.r.t. the found edit path.

Example 4.1. In Figure 4.2 an illustration of the procedure is given for two graphs of the Letter graph data set (graphs from this data set represent artificially distorted letter line drawings, and are often used for illustration purposes [119]). For this example the matching between the source and target graph results in the edit path $\lambda(g, g') = \{0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 2, 3 \rightarrow 3, 4 \rightarrow 4, 5 \rightarrow \varepsilon, \varepsilon \rightarrow 5, \varepsilon \rightarrow 6, \varepsilon \rightarrow 7\}$. According to this edit path, the two matching-graphs that are generated without pruning are shown in Figure 4.2 (b) and (e). By applying edge pruning, we observe that edges that have no counterpart in the other graph are not included in the resulting matching-graph (like, for instance, the edges $(0, 2)$ or $(0, 4)$ in the source and target graphs, respectively). Regardless the strategy actually applied, we observe a strong denoising effect on the input graphs in this illustrative example.

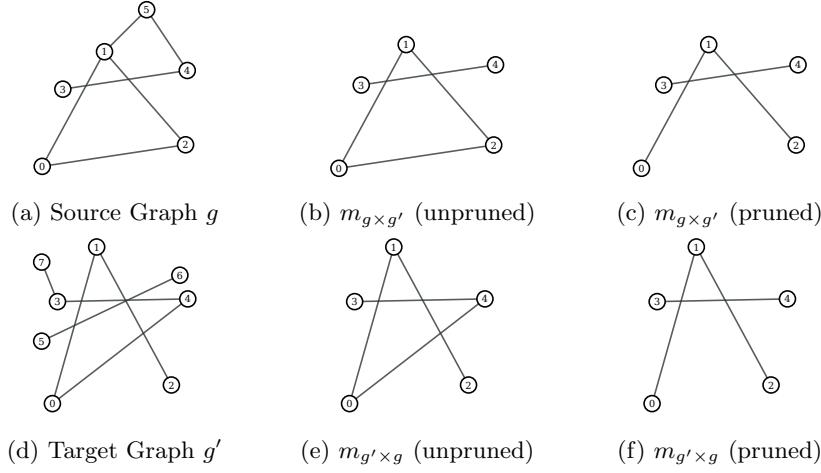


Fig. 4.2: Matching-Graphs with unpruned and pruned edges derived from source graph g and target graph g' .

The actual definition of the cost function, and in particular the cost for insertions and deletions of nodes, has a crucial impact on the resulting matching-graphs. The higher the cost for node deletions/insertions is defined, the more nodes of both graphs are substituted with each other, which in turn leads to larger matching-graphs in general. This effect is illustrated in Figure 4.3. We show different matching-graphs derived from two source graphs (representing the letters A and E). We use different cost values for node deletions/insertions. By decrementing the deletion/insertion cost we gradually obtain smaller matching-graphs (with fewer and fewer nodes in general).

4.3.1 Selecting a Small Set of Matching-Graphs

Let us assume we have a set of training graphs G_{ω_l} available. Furthermore, we assume that G_{ω_l} contains n graphs representing class ω_l . If we create all possible matching-graphs for all possible combinations of graph pairs (g, g') stemming from $G_{\omega_l} \times G_{\omega_l}$, we end up with a set of matching-graphs \mathcal{M}_{ω_l} of size $n(n-1)$. Depending on both the actual size of G_{ω_l} and the specific requirements for \mathcal{M}_{ω_l} this quantity might be too large¹. In order

¹Note that the proposed framework can instantly produce matching-graphs for any pair of graphs, and is thus not reliant on a specific set or subset of graphs. This, in turn, makes our system quite fast and flexible (since we only need to consider pairs of graphs

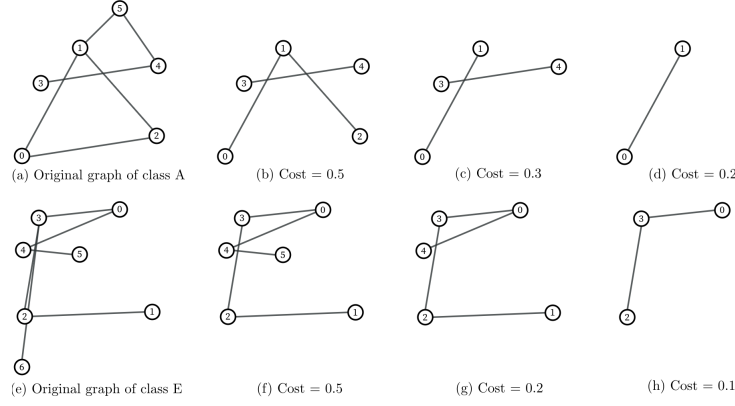


Fig. 4.3: The smaller the cost for both node deletion and insertion is defined, the smaller is the resulting matching-graph in general.

to reduce \mathcal{M}_{ω_l} to a reasonable size, various graph selection methods can be used [182]. We propose to reduce \mathcal{M}_{ω_l} with the aid of the *set median graph* [187], which is defined as

$$\text{median}(\mathcal{S}) = \underset{g \in \mathcal{S}}{\operatorname{argmin}} \sum_{g' \in \mathcal{S}} d(g, g') \quad ,$$

where \mathcal{S} is an arbitrary set of graphs. The set median graph is the graph of \mathcal{S} whose sum of distances to all other graphs in \mathcal{S} is minimal.

Based on the set median graph we propose two ways to select matching-graphs. Both approaches take as input an initial set of matching-graphs \mathcal{M}_{ω_l} and a user defined parameter t which corresponds to the number of matching-graphs desired. The first algorithm, iteratively selects (and eventually removes) the set median graph from the set of all available matching-graphs \mathcal{M}_{ω_l} until the required number t of matching-graphs is selected (see Algorithm 1). That is, we select in total t matching-graphs that are situated in, or near, the center of the complete set of matching-graphs.

Second, we propose a spanning based approach. We also start by selecting the set median graph. Each additional matching-graph selected is the graph the furthest away from the already selected matching-graphs. We repeat this procedure until the required number t of matching-graphs is selected (see Algorithm 2).

to create a new graph). Moreover, since it is possible to specify in advance how many graphs of the training set are actually used to create the matching-graphs, scalability is not a major problem in practical applications.

Algorithm 1: Center-Selection ($\mathcal{M}_{\omega_l}, t$)

```

1: Initialize  $\bar{\mathcal{M}}_{\omega_l}$  to the empty set  $\{\}$ 
2: while  $|\bar{\mathcal{M}}_{\omega_l}| < t$  do
3:    $m = \text{median}(\mathcal{M}_{\omega_l})$ 
4:    $\bar{\mathcal{M}}_{\omega_l} = \bar{\mathcal{M}}_{\omega_l} \cup \{m\}$ 
5:    $\mathcal{M}_{\omega_l} = \mathcal{M}_{\omega_l} \setminus \{m\}$ 
6: end while
7: return  $\bar{\mathcal{M}}_{\omega_l}$ 

```

Algorithm 2: Spanning-Selection ($\mathcal{M}_{\omega_l}, t$)

```

1: Initialize  $\bar{\mathcal{M}}_{\omega_l}$  to the empty set  $\{\}$ 
2:  $m = \text{median}(\mathcal{M}_{\omega_l})$ 
3:  $\bar{\mathcal{M}}_{\omega_l} = \bar{\mathcal{M}}_{\omega_l} \cup \{m\}$ 
4:  $\mathcal{M}_{\omega_l} = \mathcal{M}_{\omega_l} \setminus \{m\}$ 
5: while  $|\bar{\mathcal{M}}_{\omega_l}| < t$  do
6:    $m = \underset{g \in \mathcal{M}_{\omega_l}}{\operatorname{argmax}} \min_{m \in \bar{\mathcal{M}}_{\omega_l}} d(g, m)$ 
7:    $\bar{\mathcal{M}}_{\omega_l} = \bar{\mathcal{M}}_{\omega_l} \cup \{m\}$ 
8:    $\mathcal{M}_{\omega_l} = \mathcal{M}_{\omega_l} \setminus \{m\}$ 
9: end while
10: return  $\bar{\mathcal{M}}_{\omega_l}$ 

```

4.3.2 Creating a Large Set of Distinct Matching-Graphs

The overall aim of the two methods described in the previous subsection is to reduce the set of matching-graphs to a reasonable size. Depending on the actual application and requirements it might be beneficial to have a large set of matching-graphs that are distinct from each other. For this purpose, we propose an iterative algorithm to produce matching-graphs out of existing matching-graphs². Algorithm 3 takes as input k sets of graphs $G_{\omega_1}, \dots, G_{\omega_k}$ with graphs from different classes $\omega_1, \dots, \omega_k$, as well as the number of matching-graphs n that will be kept from one iteration to another (n is a user defined parameter).

The algorithm iterates over all k sets (classes) of graphs from $G \in \mathcal{G}$ (main loop of Algorithm 3, from line 2 to line 14). For each set of graphs G and for all possible pairs of graphs g, g' stemming from the current set G , the initial set of matching-graphs M is produced (line 3 to 6)³. Note that a matching-graph is only added to M if it does not already exist in

²The method described in the present section is similar to the algorithm proposed in [36]. In contrast with [36], however, we use a simplification of the algorithm so that we get more and also distinct graphs.

³We take into account the first matching-graph $m_{g \times g'}$ only. Moreover, due to computational reasons we stick with the pruned version of the matching-graphs.

Algorithm 3: Iterative Matching-graph Creation.

input : sets of graphs from k different classes $\mathcal{G} = \{G_{\omega_1}, \dots, G_{\omega_k}\}$, the maximum number n of matching-graphs to keep in each iteration
output: sets of matching-graphs for each of the k different classes
 $\mathcal{M} = \{M_{\omega_1}, \dots, M_{\omega_k}\}$

```

1 Initialize  $\mathcal{M}$  as the empty set:  $\mathcal{M} = \{\}$ 
2 foreach set of graphs  $G \in \mathcal{G}$  do
3   Initialize  $M$  as the empty set:  $M = \{\}$ 
4   foreach pair of graphs  $g, g' \in G \times G$  with  $j > i$  do
5      $M = M \cup \{m_{g' \times g}, m_{g \times g'}\}$ 
6   end
7   do
8      $M'$  is a subset of  $n$  randomly selected graphs of  $M$ 
9     foreach pair of graphs  $m_i, m_j \in M' \times M'$  with  $j > i$  do
10       $M = M \cup \{m_{m_j \times m_i}, m_{m_i \times m_j}\}$ 
11    end
12    while  $M$  has changed in the last iteration
13     $\mathcal{M} = \mathcal{M} \cup M$ 
14 end

```

M , meaning that we do not allow duplicates in M . Eventually, we aim at iteratively building matching-graphs out of pairs of existing matching-graphs. The motivation for this procedure is to further reduce the size of the matching-graphs and to find small core-structures that are often available in the corresponding graphs. Due to computational limitations, we randomly select a subset of size n from the current matching-graphs M (line 8). Based on this selection, the next generation of matching-graphs is built. This is actually carried out in the second for-loop on lines 9 to 11 where for all pairs of graphs in M' two novel matching-graphs are created and added to M . This process is repeated until no more changes occur in set M . Finally, set \mathcal{M} is compiled as the union of all matching-graphs individually produced for all available classes.

Example 4.2. In Figure 4.4 we provide a small illustrative example of our iterative procedure on four graphs from the Letter data set [119] in order to give the reader an intuition. Subfigures (a) to (d) show the original graphs. Subfigure (e) shows the resulting matching-graph $m_{g_i \times g_j}$ of graph g_i and g_j , whereas Subfigure (f) shows the matching-graph $m_{g_o \times g_k}$ resulting from graphs g_o and g_k . Finally, in Subfigure (g) we show the matching-graph resulting from the two matching-graphs of the first iteration. This example illustrates that the size of the matching-graphs declines from one iteration to another, in general. The matching-graph in Subfigure 4.4 (g) appears very small and generic and not specific to the class. Note, however, that the specificity heavily depends on the node labels.

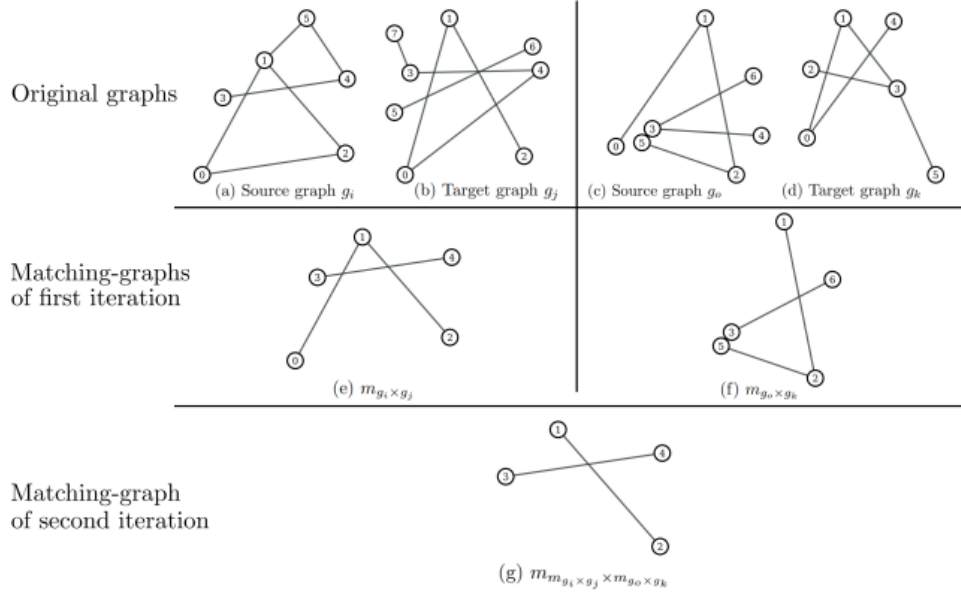


Fig. 4.4: Illustration of the procedure of creating matching-graphs over multiple iterations.

4.4 Classification with Matching-Graphs

We propose two approaches for using the matching-graphs in a classification scenario. The first idea is to enhance the accuracy of graph edit distance by explicitly focusing on matching-graphs (detailed in Section 4.4.1). The second idea is to use the resulting matching-graphs for graph embedding (detailed in Section 4.4.2).

4.4.1 Distance-Based Classification Using Matching-Graphs

In our first approach we use the matching-graphs in a distance-based classification scenario. Let us assume we aim at computing the distance between

a given test graph g and a training graph $g \in G_{\omega_l}$. We define a novel distance measure $d_M(\cdot, \cdot)$ that combines the following two distances with each other.

- (1) The approximated graph edit distance information $d_{BP}(g, g')$ between the test graph g and the original training graph $g' \in G_{\omega_l}$.
- (2) A statistical score S on the basis of *all* distances between the test graph g and all matching-graphs $m \in \mathcal{M}_{\omega_l}$ stemming from the set of matching-graphs of class ω_l (the actual class of the corresponding training graph g').

Formally, we define the distance d_M as a weighted sum of the original edit distance and the information obtained by means of the meta-matching. That is,

$$d_M(g, g') = \alpha \cdot d_{BP}(g, g') + (1 - \alpha) \cdot S(\{d_{BP}(g, m) : m \in \mathcal{M}_{\omega_l}\}) \quad ,$$

where $\alpha \in [0, 1]$ is a weighting parameter to trade off between the two dissimilarity scores and function S denotes a descriptive statistical value computed on the set of distances between the original graph g and the matching-graphs $m \in \mathcal{M}_{\omega_l}$ (we propose to use the minimum, the maximum, or the average function for S). Clearly, with $\alpha = 1$ we obtain the standard graph edit distance, while $\alpha = 0$ leads to a distance that relies on the matching-graphs only.

The set of matching-graphs \mathcal{M}_{ω_l} actually used for building d_M , is created and reduced according to the process described in Section 4.3.1. It would also be possible to use the large sets of matching-graphs iteratively created by means of the method described in Section 4.3.2 (it actually turns out that this produces similar results as the proposed combination). However, this specific setup is computationally much more demanding – due to the very large set of matching-graphs – and is therefore not a viable alternative which is not pursued.

We employ this novel distance model in two classifiers. First, we feed d_M into a k -NN classifier denoted by $k\text{-NN}(d_M)$. Second, we use the novel distance as basic similarity kernel $\kappa(g, g') = -d_M(g, g')$ in conjunction with a Support Vector Machine (denoted as $\text{SVM}(-d_M)$).

4.4.2 Graph Embedding Using Matching-Graphs

The general idea of the second classification approach is to embed a given graph into a vector space by means of the matching-graphs. Let g be

an arbitrary graph stemming from a given set of graphs. Using a large set $\mathcal{M} = \{m_1, \dots, m_N\}$ of N matching-graphs, created according to the method described in Section 4.3.2. One could also employ the matching-graph selection described in Section 4.3.1 for this purpose. The rationale of this selection is, however, to reduce an existing set. For embedding, on the other hand, we are more interested in generating large sets of distinct graphs. Therefore, this particular combination seems a bit counter-intuitive. Moreover, we observe that both combinations – given that the used sets are actually large enough – achieve quite similar results. Hence, we only follow one of the two possible combinations.

We embed g in two different ways. Once using subgraph isomorphism (as proposed in [37, 38]) and once using the graph edit distance.

The first embedding is defined by

$$\varphi_{sub}(g) = (sub(m_1, g), \dots, sub(m_N, g)),$$

where $sub(m_i, g) = 1$, if $m_i \subseteq g$, and 0 else.

That is, for this embedding we employ subgraph isomorphism that provides us with a binary similarity measure which is 1 or 0 for subgraph-isomorphic and non-subgraph-isomorphic graphs, respectively. When considering if a given matching-graph m_i is a subgraph of a graph g , it is necessary to decide whether or not two nodes are equal with respect to their labels. For nodes with categorical labels this task can be solved in a straightforward manner. When a node, however, contains continuous labels, this decision process is more subtle. In this particular case one could, for instance, employ a distance measure for the node labels and eventually define a threshold to decide whether or not two nodes are similar enough to be considered as equal.

There are various algorithms available that can be applied to solve the subgraph isomorphism problem[11, 12]. In the present chapter we employ the VF2 algorithm [188] which makes use of efficient heuristics to speed up the search process.

The second embedding is defined by

$$\varphi_{ged}(g) = (d_{BP}(g, m_1), \dots, d_{BP}(g, m_N))$$

In other words we compute the graph edit distance (in our case using the suboptimal algorithm BP [85]), between the graph g to be embedded and all matching-graphs in \mathcal{M} and then represent g as a vector of the resulting distances.

Obviously, both graph embeddings produce vectors with a dimension that is equal to the number of matching-graphs actually available. As the

iterative method described in Section 4.3.2 might generate thousands of matching-graphs, the dimension of the resulting feature vectors might be very large. In cases where the high dimensionality of the data is a problem, one can apply an arbitrary feature selection method to the resulting graph embeddings.

These specific graph embeddings are similar in spirit to the frequent substructure approaches [181], the subgraph matching kernel [28], the graphlet kernel [67], as well as dissimilarity based embeddings [182]. The main difference of our approach to those methods lies in the creation of the subgraphs (or prototypes in case of [182]). We employ graph edit distance to create our novel data structure of matching-graphs. These matching-graphs offer a natural way of defining significant and large sets of subgraphs that can readily be used for vector space embeddings.

Likewise to the novel distance d_M , also for the graph embedding we employ two different classifiers. First, we classify the resulting vectors using a k -NN in conjunction with a vector similarity measure s . For the subgraph based embedding $\varphi_{sub}(g)$ we use binary similarity measures *Dice*, *Yule*, *Tanimoto (Rogers)*, *Jaccard coefficient*, as well as *Kulczynski-1* and *2*. For the distance-based embedding $\varphi_{ged}(g)$ we use the *Euclidean*, *Cosine* and *Minkowski* dissimilarity. Second, we employ an SVM that operates on the embedding vectors (using standard kernel functions k for feature vectors such as the *Radial Basis Function (RBF)*, the *Sigmoid kernel*, and the *Linear kernel*). We denote these approaches as $k\text{-NN}(s_{\varphi(g)})$ and $\text{SVM}(\kappa_{\varphi(g)})$, respectively.

4.5 Experimental Evaluation

4.5.1 Experimental Setup

The main question to be answered in our empirical evaluation is whether the proposed matching-graphs can be used to improve the classification accuracies of existing graph matching procedures (that rely on the same graphs and graph edit distance information as our novel procedure). Hence, we compare our novel method with two reference classifiers that are often used in conjunction with graph edit distance.

The first reference system is a k -nearest-neighbor classifier (k -NN) that directly operates on the distances d_{BP} , denoted as $k\text{-NN}(d_{BP})$ from now on. The second reference system is a Support Vector Machine, denoted as $\text{SVM}(-d_{BP})$, that operates on the similarity kernel $\kappa(g, g') = -d_{BP}(g, g')$.

Table 4.1: The total number of graphs for each data set as well as the corresponding number of graphs in the training, validation, and test sets.

Data set	Total	Training	Validation	Test
AIDS	2,000	250	250	1,500
Mutagenicity	4,337	1,500	500	2,337
NCI1	4,110	2,465	822	823
COX-2	466	280	93	93
PTC(MR)	344	206	68	70
Letter	2250	750	750	750
IMDB	1,000	600	200	200

We evaluate the novel approaches for graph classification using matching-graphs on seven data sets (namely AIDS, Mutagenicity, NCI1, COX-2, PTC(MR), Letter and IMDB) as described in Chapter 3.

4.5.2 Validation of Metaparameters

For the experimental evaluation each data set is split into three predefined random disjoint sets for training, validation, and testing. Details about the size of the individual splits can be found in Table 4.1. The matching-graphs are created on the training set only, whereas the optimization of the metaparameters is performed with the help of the validation set. The optimal parameters obtained with the usage of the validation set are then applied on the test set (without any further modifications).

For algorithm BP, that approximates the graph edit distance, the cost for node and edge deletions, as well as a weighting parameter $\beta \in [0, 1]$ that is used to trade-off the relative importance of node and edge edit costs are often optimized [85, 182]. However, for the sake of simplicity we employ unit cost of 1.0 for deletions and insertions of both nodes and edges and optimize the weighting parameter β only (on all data sets). For data sets where the underlying graphs contain label alphabets L_v with categorical labels, the cost of non-identical substitutions is set to the cost of one insertion plus the cost of one deletion (which amounts to 2). For data sets with continuous node labels, we employ the Euclidean distance as a cost for substituting the two nodes. For the creation of the matching-graphs – actually also dependant on the cost model – the same cost parameters are employed.

For both classification algorithms k -NN(d_M) and SVM($-d_M$) we optimize the weighting parameter α (used in d_M), the type of matching-graphs

(pruned vs. unpruned), the matching-graph selection method (*center* vs. *spanning*), the number t of selected matching-graphs per class, and function S , that determines whether the minimum, the maximum, or the average is used to condense the set of distances $\{d_{BP}(g, m) : m \in \mathcal{M}_{\omega_l}\}$. In addition, for the k -NN classifier we optimize the number of neighbors k considered, while for the SVM classifier parameter C is optimized to trade off between the size of the margin and the number of misclassified training examples. The discussion of the validation results and the actual parameter values can be found in Subsection 4.5.3.

Both classification algorithms $k\text{-NN}(s_{\varphi(g)})$ and $\text{SVM}(\kappa_{\varphi(g)})$ rely on graph embeddings $\varphi(g)$ that are computed by means of large sets of matching-graphs. Remember that these sets are created in an iterative manner. We set the number of matching-graphs considered for the next iteration to $n = 200$ on all data sets. The stop criterion of the iterative process checks whether or not the last iteration resulted in a change of the currently considered set of matching-graphs and the dimension of the resulting feature vectors turns out to be very large. Thus, we apply a recursive feature elimination process [189] to the resulting graph embeddings. In Table 4.2 we show the total number of matching-graphs produced first and the number of matching-graphs selected. On all data sets substantial reductions can be observed. For instance, on AIDS, Mutagenicity, COX-2, PTC(MR), Letter and IMDB about 4 to 5% of the available matching-graphs are selected, while on NCI1 about 13% of the matching-graphs are selected. The conclusions we draw from this table are twofold. First, we note that the iterative procedure can be used to produce almost arbitrarily large sets of graphs. Second, it appears that only a small fraction of the matching-graphs are actually needed. Of course, it would be desirable to produce from the very start only those matching-graphs that will really be used – with our current solution this is not possible and we thus follow the well-known paradigm of *overproduce and select*.

In Figure 4.5 we can see the cross validation accuracy as a function of the number of features after each step of the recursive feature elimination process. We use the three data sets AIDS, Mutagenicity, and NCI1 as examples here, as all the data sets show a similar pattern. It is clearly visible that if the dimension of the vectors becomes too small, the validation accuracy drops by a large margin. However, before this significant drop the accuracy remains relatively stable.

For both approaches $k\text{-NN}(s_{\varphi(g)})$ and $\text{SVM}(\kappa_{\varphi(g)})$ we optimize the type of the embedding ($\varphi_{sub}(g)$ vs. $\varphi_{ged}(g)$) as well as the weighting parameter

Table 4.2: The number of matching-graphs created for each data set and the final number of matching-graphs after feature selection is applied.

Data Set	Total	Selected
AIDS	4,955	199
Mutagenicity	86,752	4,139
NCI1	4,544	618
COX-2	19,704	989
PTC(MR)	3,893	207
Letter	13,514	689
IMDB	43,015	2,141

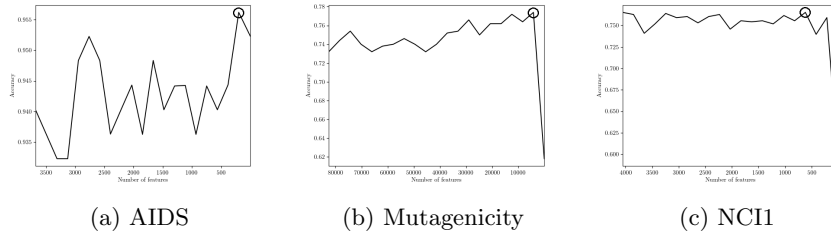


Fig. 4.5: Cross validation accuracy as a function of the number of features during the recursive feature elimination process. The global optimum is indicated with a small circle.

$\beta \in [0, 1]$ that is used to trade-off the relative importance of node and edge edit costs. For the k -NN we further optimize the similarity measure s as well as the number k of neighbors that are considered. For the SVM we optimize the kernel function and parameter $C \in [0, 1]$. In the case of an RBF or Sigmoid kernel, parameter $\gamma \in [0, 1]$ is optimized as well. All optimizations are conducted by means of a grid search. The detailed validation results can be found in Section 4.5.4.

4.5.3 Validation Results k -NN(d_M) and SVM($-d_M$)

In Tables 4.3 and 4.4 we show the best performing parameters found during validation for k -NN(d_M) and SVM($-d_M$), respectively. Major findings of the validation process for the k -NN are as follows (see also Table 4.3). On all data sets but NCI1 the parameter α lies between 0.35 and 0.65, which means that both distance informations are – more or less – equally important. Moreover, we observe that pruning seems to be beneficial in

Table 4.3: Optimal parameter values found on the validation sets for the k -NN(d_M) classifier (including the classification accuracy).

Data set	β	α	pruning	Selection	t	S	k	Accuracy
AIDS	0.90	0.35	pruned	center	30	max	7	99.6
Mutagenicity	0.60	0.60	pruned	center	10	max	1	76.2
NCI1	0.75	0.90	pruned	spanning	3	avg	1	78.5
COX-2	0.40	0.50	pruned	center	10	avg	5	84.9
PTC(MR)	0.60	0.30	pruned	center	7	max	7	69.1
Letter	0.65	0.60	pruned	center	60	max	3	93.2
IMDB	0.40	0.65	unpruned	center	65	max	3	75.5

Table 4.4: Optimal parameter values found on the validation sets for the SVM($-d_M$) classifier (including the classification accuracy).

Data set	β	α	pruning	Selection	t	S	C	Accuracy
AIDS	0.60	0.85	pruned	spanning	75	min	10^{-1}	100.0
Mutagenicity	0.75	0.65	pruned	center	15	max	10^{-2}	73.2
NCI1	0.75	0.95	pruned	center	75	max	10^{-2}	67.3
COX-2	0.95	0.90	pruned	center	25	max	10^{-1}	82.8
PTC(MR)	0.75	0.90	unpruned	spanning	25	min	10^1	77.9
Letter	0.70	0.90	pruned	spanning	75	min	10^{-2}	94.3
IMDB	0.35	0.90	pruned	center	65	avg	10^{-3}	70.0

general, and that the optimal selection method is center on all data sets but NCI1. Finally we can see that the max function performs the best for distance aggregation in almost all cases.

The optimal parameters for SVM (see Table 4.4) indicate that the aggregated distance is rather less important and that the optimal selection method is center in most of the cases. Furthermore, the min, together with the max function, is often used for condensing the set of distances.

4.5.4 Validation Results k -NN($s_{\varphi(g)}$) and SVM($\kappa_{\varphi(g)}$)

In Tables 4.5 and 4.6 we show the optimal parameters for k -NN($s_{\varphi(g)}$) and SVM($\kappa_{\varphi(g)}$). For the k -NN($s_{\varphi(g)}$) (see Table 4.5) the embedding applied is $\varphi_{sub}(g)$ for all data sets except for Letter. Regarding the similarity measure s we can not report a clear winner (although Kulczynski-1 or -2 might be a good choice when in doubt).

The optimal parameters for the SVM (see Table 4.6) indicate that the best embedding function is on four out of seven data sets the subgraph

Table 4.5: Optimal parameter values found on the validation sets for the k -NN($s_{\varphi(g)}$) classifier (including the classification accuracy).

Data set	Embedding	β	s	k	Accuracy
AIDS	φ_{sub}	N/A	Kulczynski-1	5	99.6
Mutagenicity	φ_{sub}	N/A	Dice	7	79.2
NCI1	φ_{sub}	N/A	Rogers	5	79.2
COX-2	φ_{sub}	N/A	Yule	5	82.8
PTC(MR)	φ_{sub}	N/A	Kulczynski-2	5	72.1
Letter	φ_{ged}	0.70	Cosine	5	91.2
IMDB	φ_{sub}	N/A	Kulczynski-2	7	76.0

Table 4.6: Optimal parameter values found on the validation sets for the SVM($\kappa_{\varphi(g)}$) classifier (including the classification accuracy).

Data set	Embedding	β	Kernel	C	γ	Accuracy
AIDS	φ_{sub}	N/A	RBF	10^{-1}	10^0	99.6
Mutagenicity	φ_{sub}	N/A	RBF	10^2	10^{-3}	82.4
NCI1	φ_{sub}	N/A	RBF	10^1	10^{-2}	77.4
COX-2	φ_{ged}	0.40	Linear	10^{-2}	N/A	86.0
PTC(MR)	φ_{ged}	0.05	Linear	5×10^{-3}	N/A	77.8
Letter	φ_{ged}	0.60	Linear	5×10^{-3}	N/A	93.1
IMDB	φ_{sub}	N/A	RBF	10^1	10^{-3}	74.5

based function φ_{sub} . The best performing kernel function κ is in four out of seven data sets the RBF kernel, which are notably all based on the φ_{sub} embedding. On the three other data sets that use φ_{ged} for embedding, the linear kernel seems to be optimal. The best value of C is either 10 or 100 for all data sets embedded with the φ_{sub} embedding, except for AIDS, where the optimal value is 0.1. For the φ_{ged} embedded graphs the optimal values for C are much smaller (between 0.005 and 0.01). The optimal parameter γ on the other hand is smaller than 0.01 for all data sets except for AIDS (where $\gamma = 1$ performs the best).

4.5.5 Test Results and Discussion

In Table 4.7 we show the classification accuracies of both reference systems, viz. k -NN(d_{BP}) and SVM($-d_{BP}$), as well as the results of our novel approaches k -NN(d_M), SVM($-d_M$), k -NN($s_{\varphi(g)}$), and SVM($\kappa_{\varphi(g)}$) that all rely on matching-graphs.

We observe that our novel approaches k -NN(d_M) and SVM($-d_M$) out-

Table 4.7: Classification accuracies of two reference systems compared to our novel approaches. Symbol \circ/\circ indicates a statistically significant improvement and \bullet/\bullet indicates a statistically significant deterioration over the first and second system, respectively (using a Z-test at significance level $\alpha = 0.05$). The best result per data set is shown in bold face.

Data Set	Reference Systems		Proposed System			
	$k\text{-NN}(d_{\text{BP}})$	$\text{SVM}(-d_{\text{BP}})$	$k\text{-NN}(d_{\text{M}})$	$\text{SVM}(-d_{\text{M}})$	$k\text{-NN}(s_{\varphi(g)})$	$\text{SVM}(\kappa_{\varphi(g)})$
AIDS	98.6	99.4	99.8 $\circ/-$	99.7 $\circ/-$	99.5 $\circ/-$	99.6 $\circ/-$
Mutagenicity	72.4	69.1	73.0 $-/\circ$	70.4 $-/\circ$	74.8 \circ/\circ	76.3 \circ/\circ
NCI1	74.4	68.6	77.6 \circ/\circ	68.8 $\bullet/-$	76.1 $-/\circ$	76.7 $-/\circ$
COX-2	76.3	71.3	81.7 \circ/\circ	81.0 $-/\circ$	80.6 $-/\circ$	78.5 $-/\circ$
PTC(MR)	55.7	54.3	65.7 \circ/\circ	67.1 \circ/\circ	58.6 $-/-$	61.4 $-/-$
Letter	89.9	92.7	91.7 $\circ/-$	92.5 $\circ/-$	90.8 \circ/\bullet	93.2 $\circ/-$
IMDB	60.5	63.5	68.0 \circ/\circ	66.0 $\circ/-$	59.0 $-/-$	68.5 $\circ/-$

perform their respective reference systems on all data sets (except for Letter where the $\text{SVM}(-d_{\text{M}})$ approach achieves approximately the same accuracy as $\text{SVM}(-d_{\text{BP}})$). For $k\text{-NN}(d_{\text{M}})$ we observe that six out of seven improvements are statistically significant, while three out of six improvements achieved with $\text{SVM}(-d_{\text{M}})$ are statistically significant⁴.

The classifier $k\text{-NN}(s_{\varphi(g)})$ achieves higher accuracies than both reference systems on five out of seven data sets (i.e., 10 improvements in total). Five of these improvements are statistically significant. The classifier $\text{SVM}(\kappa_{\varphi(g)})$ achieves even better accuracies in general. We outperform both reference systems on all data sets. Seven of the 14 improvements are statistically significant.

Comparing our novel classifiers with each other, we observe that $k\text{-NN}(d_{\text{M}})$ performs the best in general. That is, it outperforms both reference systems on all seven data sets, with 11 out of 14 improvements being statistically significant. Moreover, on three out of seven data sets, this classifier achieves the overall best classification results (followed by $\text{SVM}(\kappa_{\varphi(g)})$ that achieves the best result in three out of seven cases, and $\text{SVM}(-d_{\text{M}})$ that performs best for one data set).

4.5.6 Ablation Study, Comparison with State of the Art and Run Time Analysis

We can state the following as an interim conclusion. Our novel approach using matching-graphs is clearly beneficial when compared with similar

⁴The statistical significance is computed via Z-test using a significance level of $\alpha = 0.05$.

systems that have no access to the matching-graphs. The aim of the next evaluations presented in this subsection is threefold. First, we conduct an ablation study in order to better get to the root of the strength of our novel framework. Second, we conduct a comparison with three state-of-the-art methods from the field, and third we perform a run time analysis.

4.5.6.1 Ablation Study

For the embedding approach, we aim to determine whether it is the matching-graphs themselves, the iterative construction of the sets of matching-graphs or the selection of certain features that helps the most to improve the results. To this end, we conduct the following ablation study using the results of $\text{SVM}(\kappa_{\varphi(g)})$ (with the subgraph based embedding).

- *Without-1*: This is a system which operates *without* matching-graphs. That is, this approach uses randomly generated subgraphs, rather than our matching-graphs, for graph embedding. The random generation of subgraphs works by randomly removing 30 to 50% of the nodes from the graphs (and their incident edges). The amount of random graphs created for each data set corresponds to the number of matching-graphs actually used for $\text{SVM}(\kappa_{\varphi(g)})$. This set of random subgraphs is then used for graph embedding. We repeat the random creation of subgraphs and classification five times and report the mean and standard deviation of the accuracy.
- *Without-2*: This is a system which refrains from producing the matching-graphs with an iterative procedure as suggested in Section 4.3.2. Instead we use the matching-graphs created after the first for loop of Algorithm 3 (at line 6).
- *Without-3*: This is a system that works *without* feature selection. It uses all matching-graphs created during the iterative process.

In Table 4.8 we see that *Without-1* performs worse compared to all other approaches on all data sets (except for the PTC(MR) data set, where the accuracy of *Without-3* is even worse). This is a first and quite strong indication of the usefulness of the matching-graphs. Next, we conclude that the iterative process on its own is not beneficial, as *Without-2* and *Without-3* perform almost equally well (except on Mutagenicity and Letter). However, feature selection applied to the resulting embedding is definitely useful as our complete system outperforms both *Without-2* and *Without-3* on all data sets (except on Letter).

Table 4.8: Classification accuracies of an approach that uses randomly created subgraphs for embedding instead of using matching-graphs (Without-1), an approach that uses matching-graphs without using the iterative process (Without-2), as well as an approach that uses the embedded graphs without feature selection (Without-3), compared to our novel approach $\text{SVM}(\kappa_{\varphi(g)})$.

Data Set	Without-1	Without-2	Without-3	$\text{SVM}(\kappa_{\varphi(g)})$
AIDS	95.5 ± 0.7	99.3	99.2	99.6
Mutagenicity	69.1 ± 1.2	74.3	73.1	76.3
NCI1	70.6 ± 0.6	73.4	73.4	76.7
COX-2	73.1 ± 1.3	77.4	77.7	78.7
PTC(MR)	48.3 ± 1.2	44.3	44.3	67.2
Letter	86.1 ± 1.0	90.5	89.9	90.1
IMDB	60.9 ± 6.1	66.5	65.5	68.5

In summary, the strength of our novel framework also lies in the combination of the iterative generation with a subsequent feature selection. The most valuable component of the proposed system is, however, the concept of matching-graphs themselves (as the comparisons with the reference system *Without-1* clearly show).

4.5.6.2 Comparison with State-of-the-Art

In Table 4.9 we put the best accuracies of our novel framework (denoted as *Ours*) in the context with several other kernel based classifiers that are evaluated with the same experimental setup and data sets as used in the present chapter. In particular, we compare our method with the *Graphlet kernel* [67], *Shortest-Path kernel (SP)* [30], as well as the *Wasserstein Weisfeiler-Lehman kernel (WWL)* [190]⁵. On the Mutagenicity and NCI1 data sets our approach is narrowly outperformed by the WWL kernel and on the IMDB data set the SP kernel beats our system quite clearly. However, we can also report that on four out of seven data sets our framework achieves the overall best accuracy when compared with the current state-of-the-art.

⁵We compute the reference accuracies using the GraphKernels library [191] for the Graphlet and Shortest-Path kernel. For the WWL kernel we use the implementation provided in [190].

Table 4.9: Comparison of the classification accuracies of our novel framework with three state-of-the-art kernel based approaches, viz. Graphlet, Shortest-Path (SP) and Wasserstein Weisfeiler-Lehman (WWL). The best accuracy per data set is shown in bold face. A dash (-) as entry indicates that the experiment timed out or that we do not get a reasonable result.

Data Set	Graphlet [67]	SP [30]	WWL [190]	Ours
AIDS	98.5	99.4	99.5	99.8
Mutagenicity	55.5	-	77.0	76.3
NCI1	64.1	73.0	79.3	77.6
COX-2	77.4	49.5	77.4	81.7
PTC(MR)	55.7	55.7	54.3	67.1
Letter	30.1	-	41.1	93.2
IMDB	58.0	73.0	71.0	68.5

Table 4.10: Run time comparison of the time needed to calculate the baseline matrix $-d_{BP}$, as well as the time needed to calculate our novel matrix $-d_M$ and the times needed to create the subgraph embedding $\varphi_{sub}(g)$ and graph edit distance-based embedding $\varphi_{ged}(g)$ for one graph. Time in Seconds.

Data Set	Distance-Based Classifiers		Embedding-Based Classifiers	
	$-d_{BP}$	$-d_M$	$\varphi_{sub}(g)$	$\varphi_{ged}(g)$
AIDS	9	15	54	~ 0
Mutagenicity	224	240	3020	6
NCI1	552	601	633	1
COX-2	10	15	820	3
PTC(MR)	2	3	41	~ 0
Letter	41	63	3	1
IMDB	19	22	996	3

4.5.6.3 Run Time Discussion

Of course the main downside of the proposed framework is the additional run time that comes from the increased computational demands.

For the following discussion on the run times, we distinguish between classification systems that rely on distances and systems that are based on embeddings. For the first category we identify the following two computations whose run times are of interest.

- Run time to compute the complete distance matrix using the original graph edit distance d_{BP} by means of algorithm BP. This can be taken

as a reference run time for relative comparison with the following operation.

- Run time to compute the complete distance matrix using the enhanced graph edit distance d_M using the matching-graphs.

For the second category, we are most interested in the run time for graph embedding as this is the bottleneck of our framework. In particular, we report the run times for one embedding using either the subgraph approach $\varphi_{sub}(g)$ or the graph edit distance-based approach $\varphi_{ged}(g)$.

In Table 4.10 we show the four discussed run times of both categories in seconds. When comparing the run times for the computation of the novel distance matrix d_M with the original distance computation d_{BP} , only marginal differences can be observed. If we take into account that, for example, the k -NN that uses d_M outperforms k -NN(d_{BP}) on all seven data sets (six times with statistical significance), then this small overhead in run time is more than justified.

For the run times of the second category, weighing is more important than in the first category discussed above. For instance, it is obvious that the subgraph based embedding is significantly slower than the distance-based embedding. On the other hand, we have seen that subgraph based embedding basically performs slightly better than the distance-based embedding. It is not necessarily clear whether this rather small difference in accuracy can justify the large discrepancy in the run times. Note, however, that graph embedding can be parallelized with special hardware infrastructure, which in turn can dramatically reduce the high run time, if necessary.

4.6 Conclusion and Future Work

In the present chapter, we introduce and research a novel data structure called matching-graph, which can be pre-computed on training graphs. Our general goal is to leverage the power of graph edit distance to build a novel graph representation that formalizes the matching parts found between two graphs. This formalization can be interpreted as stable part, or core, of two graphs. We propose to build matching-graphs on the basis of the edit path between two graphs. Formally, a matching-graph of two graphs consists of the nodes substituted under a given cost model in a graph edit distance computation. We compute matching-graphs between all pairs of graphs stemming from the same class. Eventually, we define two complementary approaches that first, condense the initial set of matching-graphs to the

most influential ones, and second, iteratively enlarges the set of matching-graphs by recursively building novel matching-graphs out of already created matching-graphs. The benefit of these matching-graphs is that they can be utilized to improve the classification accuracy of various classifiers. The drawback is – of course – the increased computation time.

To show the usefulness of matching-graphs we propose two classification approaches. The first system employs a weighted distance of the original graph edit distance and an aggregated distance to sets of matching-graphs. The second approach uses the matching-graphs to build vector representations of the underlying graphs. To this end, we embed our graphs in an N -dimensional vector space such that the i -th entry of the resulting vector represents either the distance to the corresponding matching-graph or whether the corresponding matching-graph occurs as a subgraph in the graph to be embedded.

By means of a thorough experimental evaluation on diverse graph data sets covering a wide spectrum of applications, we empirically confirm that classification systems that (in part) rely on the novel matching-graphs significantly outperform their counterparts that have no access to this specific information.

In a thorough ablation study, we are also able to clearly underline the value of the novel matching-graphs. Last but not least, with a comparison with three state-of-the-art methods, we empirically confirm that our framework is able to set new benchmarks on several data sets.

The proposed matching-graphs have – besides the ability of improving the classification accuracy in a graph-based classification scenario – another interesting benefit. They can automatically reveal significant patterns in large sets of graphs. In particular, it turns out that matching-graphs often represent crucial patterns that actually constitute a certain class of patterns. For instance, for the mutagen class of the Mutagenicity data set we autonomically identified both patterns NO_2 and NH_2 in many matching-graphs. Both compounds are well known to be mutagenic [192]. This is especially interesting as the matching-graphs are automatically created on the basis of the edit path between training graphs without any domain knowledge. This idea is further explored in Chapter 5. Besides this idea, we identify several potential future research activities. First, rather than the approximation algorithm BP, one could employ any other graph edit distance computation. We believe that using more expensive algorithms for the computation of the edit distance could lead to other (perhaps larger?) matching-graphs. Second, the novel matching-graphs might actually be

used as a sparse representation of any input data. Hence, our framework could potentially be used as a novel and quite fast way for dictionary learning in the graph domain. Last but not least, we feel that the concept of matching-graphs might also be beneficial for regression problems (e.g. one could employ the matching-graphs in conjunction with a nearest-neighbor regression, which depends on a large number of training data).

Matching-Graphs and the Maximum Common Subgraph

5

A tale of similarity takes the stage, where matching-graphs and maximum common subgraphs engage. Through theoretical lens and trials galore, unfolds a comparison never seen before.

ChatGPT

5.1 Introduction

In various pattern recognition frameworks, it is crucial to find similarities between pairs of entities. In the case of graph-based pattern recognition, this process is called graph matching. Over the last four decades, numerous graph matching procedures have been proposed in the literature [11], ranging from *graph edit distance* [193], *graph kernels* [194], to *graph neural networks* [195]. A fundamental aspect of graph matching is the ability to find common substructures in graphs, which can provide insight into shared patterns and relationships between different entities.

The *matching-graph* (as already described in Chapter 4), follows the idea of formalizing the stable core of pairs of graphs. In particular, the information of the so-called *edit path* between two graphs is exploited in order to derive matching-graphs. An edit path gives us the information which subparts of the corresponding graphs are matched with each other by means of graph edit distance [34]. In order to optimize the matching-graphs to discover relevant substructures we refine their quality by using an iterative process that selects the best matching-graphs of each itera-

tion, creating new matching-graphs from these parent graphs. Our first hypothesis is therefore, that matching-graphs are able to find relevant substructures in their correct class and thus provide insights into the question which graph substructures actually make up a class of patterns.

Our approach is similar in spirit to approaches from graph transaction based *Frequent Subgraph Mining (FSM)* [183]. This field also focuses on the identification of frequent subgraphs within a set of graphs (extract all subgraphs that occur more often than a specified threshold). We observe two main categories in FSM, viz. *Apriori-based approaches* and *Pattern-growth approaches* [183]. The apriori-based methods proceed to grow subgraphs by using a *Breadth First Search (BFS)* strategy. Before they continue to graphs of size $k+1$ it first searches for all frequent graphs of size k . Pattern-growth approaches, on the other hand, work by using a *Depth First Search (DFS)* strategy, where one graph is extended until all frequent supergraphs of this graphs are found.

As it is our hypothesis that the matching-graphs are able to find common substructures in graphs, it naturally raises the question of their connection to the *maximum common subgraph (MCS)*. The MCS of two graphs can be interpreted as the intersection of the considered graphs. Hence, the larger the MCS is, the more similar the two graphs are. The MCS problem is known to be NP-complete [196], which means that current exact algorithms for the computation of the MCS have exponential time complexity [197]. This fact often renders the time required to compute an exact MCS unacceptable, especially for graphs with a large number of nodes. Approximate solutions aim at finding a graph that is reasonably close to the exact MCS within a much shorter time frame. However, there is generally no guarantee that the solution found by these procedures is actually similar to the MCS in terms of both size and composition.

There are several algorithms available that approximate the solution to the MCS problem. In [198], for instance, an algorithm that selects common features between two graphs using bit strings, which in turn allow faster traversals of a graph, is used. In [199] an approximation based on a discrete time quantum walk is introduced. A third approach tries to approximate the problem of MCS computation by using a build-up algorithm [200]. Note that this approach – as well as some others – are designed for specific types of graphs only (e.g., graphs that represent chemical structures). It is well known that graph edit distance and the concept of MCS are closely related with each other [201]. Hence, our second hypothesis is that the novel matching-graphs also relate to the concept of MCS.

In summary, the main contribution of the present chapter is to demonstrate the effectiveness of matching-graphs in identifying relevant substructures within individual graph classes, as well as their relation to the MCS. Note that the present chapter is based on a preliminary conference paper [36] and a journal paper [39].

The remainder of the present chapter is organized as follows. In Section 5.2, we explain which version of the matching-graph is used in this chapter and how we use it to iteratively find relevant substructures. Further, in Section 5.3, we conduct both a quantitative and qualitative experimental evaluation. In order to do this, we measure the frequencies of the found matching-graphs in their correct class and we visualize and inspect the most frequent subgraphs which in turn enables novel insights into the question which graph substructures actually make up a class of patterns. Next, in Section 5.4, we provide important definitions used in the remainder of the chapter. In Section 5.5, we explore the relation of the matching-graphs to the MCS and how we can use matching-graphs to approximate the MCS. Eventually, in Section 5.6, we conduct an experimental evaluation in order to verify the potential benefits of our novel approach. In particular, we compare the run time of the proposed MCS approximation with both exact and existing approximation algorithms. Further, we feed the new approximation into MCS based distance measures and observe whether and how this changes the accuracy of a distance-based classifier. Finally, in Section 5.7, we conclude the chapter and discuss some future research directions.

5.2 Matching-Graphs

Major contribution of the present chapter is that we explore the feasibility of matching-graphs to produce relevant substructures of pairs of graphs. This section summarizes the version of the matching-graph that is relevant for this chapter, in order to make the chapter self contained. The concept of matching-graphs is thoroughly explained in Section 4.3.

Formally, we assume two graphs $g = (V, E, \mu, \nu)$ and $g' = (V', E', \mu', \nu')$. First, the graph edit distance $d_{BP}(g, g')$ between g and g' is computed by means of algorithm BP. Hence, a (potentially suboptimal) edit path $\lambda(g, g')$ is obtained. For this edit path $\lambda(g, g')$, two matching-graphs $m_{g \times g'}$ and $m_{g' \times g}$ can now be built. To this end, all nodes of g and g' that are actually substituted in edit path $\lambda(g, g')$ are added to $m_{g \times g'}$ and $m_{g' \times g}$, respectively.

Vice versa, all nodes that are deleted in g or inserted in g' are not considered in neither of the two matching-graphs.

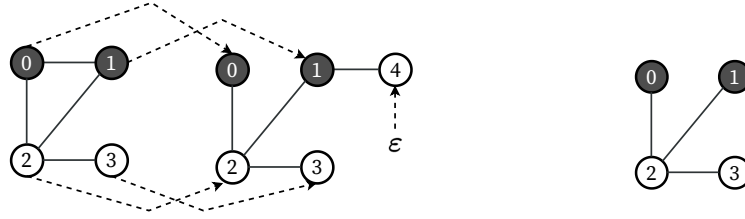
In this chapter we make use of the pruned version of the matching-graphs (as described in Section 4.3), as this version always produces sub-graphs of both original graphs. If a node is not included in the matching-graph (since it is either deleted or inserted), the incident edges of this node are not included in the resulting matching-graph. Edges that connect two substituted nodes are included in the matching-graphs under the following condition. Assume that two nodes $u, v \in V$ of g are substituted with nodes $u', v' \in V'$ in g' and there is an edge $(u, v) \in E$. In this case, (u, v) is included in the matching-graph $m_{g \times g'}$ if, and only if, edge (u', v') is also available in E' .

Formally, matching-graph $m_{g \times g'} = (V_{g \times g'}, E_{g \times g'}, \mu, \nu)$ is defined as

- $V_{g \times g'} = \{v \in V : (v \rightarrow v') \in \lambda(g, g') \text{ and } v' \in V'\}$
- $E_{g \times g'} = \{E \cap E' \cap (V_{g \times g'} \times V_{g \times g'})\}$
- μ, ν as defined for graph g .

For the matching-graph $m_{g' \times g}$ the definition is similar to $m_{g \times g'}$, but the roles of g and g' have to be exchanged.

In Figure 5.1 we show two graphs g and g' , a possible edit path $\lambda(g, g') = \{0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 2, 3 \rightarrow 3, \varepsilon \rightarrow 4\}$, as well as one of the resulting matching-graphs $m_{g \times g'}$.



(a) Two graphs, g and g' , and a possible edit path λ .

(b) $m_{g \times g'}$

Fig. 5.1: Two graphs g and g' , a possible edit path λ and the resulting matching-graph $m_{g \times g'}$.

5.2.1 Iterative Building of Matching-Graphs

Our first hypothesis is, that the matching-graphs are able to find relevant substructures in a given class of graphs. In this section we describe how we

use the matching-graph in conjunction with an iterative process, in order to produce a subset of the highest quality matching-graphs.

Using the described procedure for creating matching-graphs out of two input graphs, we now propose an algorithm that iteratively creates sets of matching-graphs out of existing sets of matching-graphs. The proposed procedure is formalized in Algorithm 4.

Algorithm 4: Algorithm for iterative matching-graph creation.

input : sets of graphs from k different classes $\mathcal{G} = \{G_{\omega_1}, \dots, G_{\omega_k}\}$, number of matching-graphs c
output: sets of matching-graphs for each of the k different classes $\mathcal{M} = \{M_{\omega_1}, \dots, M_{\omega_k}\}$

```

1 Initialize  $\mathcal{M}$  as the empty set:  $\mathcal{M} = \{\}$ 
2 foreach set of graphs  $G \in \mathcal{G}$  do
3   Initialize  $M$  as the empty set:  $M = \{\}$ 
4   foreach pair of graphs  $g_i, g_j \in G \times G$  with  $j > i$  do
5      $M = M \cup \{m_{g_j \times g_i}, m_{g_i \times g_j}\}$ 
6   end
7   reduce  $M$  to the  $c$  matching-graphs with highest quality  $q$ 
8   do
9     foreach pair of graphs  $m_i, m_j \in M \times M$  with  $j > i$  do
10       $M = M \cup \{m_{m_j \times m_i}, m_{m_i \times m_j}\}$ 
11    end
12    reduce  $M$  to the  $c$  matching-graphs with highest quality  $q$ 
13  while  $M$  has changed in the last iteration
14   $\mathcal{M} = \mathcal{M} \cup M$ 
15 end

```

The input of the algorithm is a set \mathcal{G} that contains several sets of graphs $\{G_{\omega_1}, \dots, G_{\omega_k}\}$ each representing members of a certain class ω_k . Additionally, the number of matching-graphs per class is fixed to a user-defined value c . The output is a set \mathcal{M} which consists of k different sets $M_{\omega_1}, \dots, M_{\omega_k}$ each containing c matching-graphs that represent one of the given classes.

First \mathcal{M} is initialized to the empty set. The algorithm then actually starts on line 2 by iterating over each set of graphs $G \in \mathcal{G}$. For each of these sets the corresponding result M is initialized as the empty set.

As seen on line 4 to 6, before beginning the main iterative process, we first loop through all possible combinations of graphs $g_i, g_j \in G \times G$, where $j > i$. From one edit path $\lambda(g_i, g_j)$ two matching-graphs are inferred, viz. $m_{g_i \times g_j}$ and $m_{g_j \times g_i}$, where g_i and g_j is the source and target graph, respectively (line 5). Hence, this process yields $n(n-1)$ matching-graphs, where n is the number of graphs in the current set G^1 .

¹Note that edit path $\lambda(g_i, g_j)$ is not necessarily the same as $\lambda(g_j, g_i)$ and thus, it could actually happen that the resulting matching-graphs stemming from these edit paths also differ. Yet, due to computational reasons we omit the computations of the edit paths

On line 7 we proceed to select the c graphs from M with the highest quality q by calculating the relative frequency of occurrence in their own class with respect to the occurrence in other classes. Formally, for a matching-graph $m \in M$ derived from graphs stemming from class ω_l , we verify for all graphs $g \in G_{\omega_l}$ whether or not m is a subgraph of g and store the number of positive matches in f_1 . Likewise, we count all graphs $g' \in G_{\omega_i}$, where $\omega_i \neq \omega_l$, that contain m as subgraph and store this number in f_2 .

Clearly, the higher f_1 and simultaneously the lower f_2 for a given matching-graph m , the better the quality of m . With $f_2 = \max(1, f_2)$ (in order to avoid divisions by zero), we formalize the quality q of a matching-graph m by means of

$$q(m) = \frac{f_1}{f_2} . \quad (5.1)$$

Given this initial set M of matching-graphs, the whole process is eventually repeated (lines 9 to 12). Yet, instead of creating the matching-graphs from the training set G , we produce matching-graphs from pairs of existing matching-graphs. This process is repeated as long as the c graphs in M have altered in the last iteration (line 13). Once the algorithm terminates, we obtain k sets of c matching-graphs for each class which are stored in \mathcal{M} .

5.3 Experimental Evaluation

For the first experimental evaluation, the main question is, whether or not our novel procedure is able to create more representative matching-graphs by using a quality measurement between the iterations. In order to answer this question, we count the occurrences of the matching-graphs found in a given test set (via subgraph isomorphism verification from graph-tool² that is based on the VF2 algorithm [58]). That is, we create the matching-graphs using the aforementioned algorithm, on the training sets and then count the actual occurrences of the created graphs as subgraphs in the corresponding test sets. The proposed approach is evaluated on two different data sets (namely AIDS and Mutagenicity (MUTA)) as described in Chapter 3.

The single parameter of our algorithm – namely, the number of matching-graphs being generated – is set to $c = 15$ in our experiments for the sake of convenience.

and matching-graphs in both directions and assume two matching-graphs per graph pair.

²<https://graph-tool.skewed.de>

Table 5.1: Development of the average number of nodes from the top matching-graphs between the first and last iteration.

		# iterations	Avg. # nodes first iteration	Avg. # nodes last iteration
MUTA	nonmutagen	2	17.9	18.1
	mutagen	2	14.4	14.3
AIDS	inactive	4	6.6	5.5
	active	3	14.5	12.1

5.3.1 Test Results and Discussion

First, we aim at researching whether or not the quality of the matching-graphs actually improves from iteration to iteration. To this end, we plot the qualities (according to Equation 5.1) of the top c matching-graphs from the first to the last iteration (see Figure 5.2). It is clearly observable that the quality of the matching-graphs increases by each iteration. For instance, for the AIDS data set and class active the initial matching-graphs offer quality values between 20 and 38, while the qualities of the final matching-graphs are between 39 and 45, which means that the final matching-graphs occur about 39 to 45 times more often in their own class than in the other class.

One could assume that this increase is mainly due to the fact that the matching-graphs become smaller from iteration to iteration (and are therefore found more often in the correct class). In fact, we observe only a marginal reduction of the average graph size (if any). This can be seen in Table 5.1 where we show the number of iterations per data set and class as well as the average number of nodes of the matching-graphs in the first and last iteration.

Next, we analyze the absolute frequencies of the resulting matching-graphs in the correct and false classes (see Figure 5.3). It can be clearly observed that the resulting matching-graphs occur significantly more often in their correct classes than in the wrong class for the AIDS active, Mutagenicity mutagen and nonmutagen classes.

In particular for AIDS active (Figure 5.3 (a)) we can report exciting results, where most of the matching-graphs occur in about 80% of the test graphs of the correct class, and only about 1% in the other class. However, for the AIDS inactive (Figure 5.3 (b)), the resulting matching-graphs do not seem to be representative.

Finally, we conduct a qualitative evaluation. In Figure 5.4 we visualize

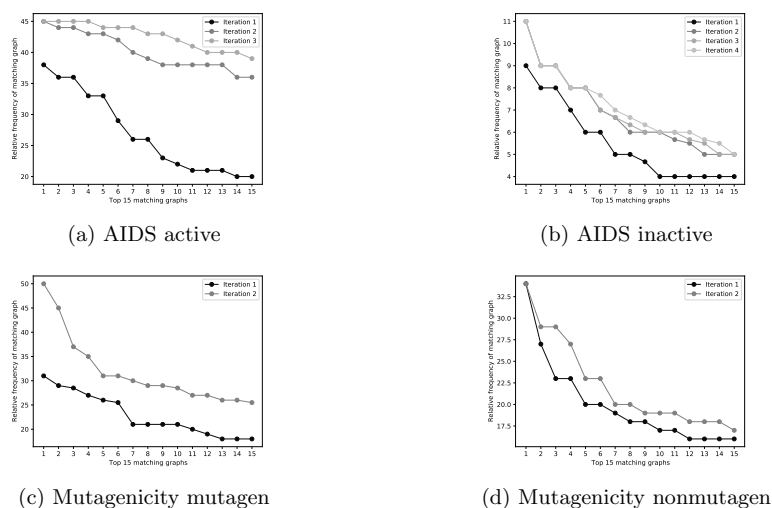


Fig. 5.2: Evolution of the relative frequencies of which a matching-graph occurs in the correct and incorrect class during the iterations.

the three matching-graphs with the best quality (according to Equation 5.1) of each class for both data sets. Interestingly, as seen in Figure 5.4 (a), the matching-graphs for the AIDS active class consist of carbon atoms only (in very specific combinations, that seems to be exclusive for this class). The matching-graphs of the inactive class on the other hand consist of chains of various atoms, that seem to be less common overall and not very specific to the inactive class (as seen in the quantitative analysis in Figure 5.3).

In Figure 5.4 (b) we show the top matching-graphs for the Mutagenicity data set. One of the major differences is, that for the mutagen class the matching-graphs found often contain carbon rings or partial carbon rings, whereas in the nonmutagen class we find much more hydrogen atoms. Also very interesting to see is that the second sample of the Mutagen class in Figure 5.4 (b) contains a NO_2 compound, which is well known to be mutagenic[202]. Overall the NO_2 compound occurs in 5 out of the 15 matching-graphs. This is especially interesting as the matching-graphs are automatically created on the basis of the edit path between training and matching-graphs without any further knowledge.

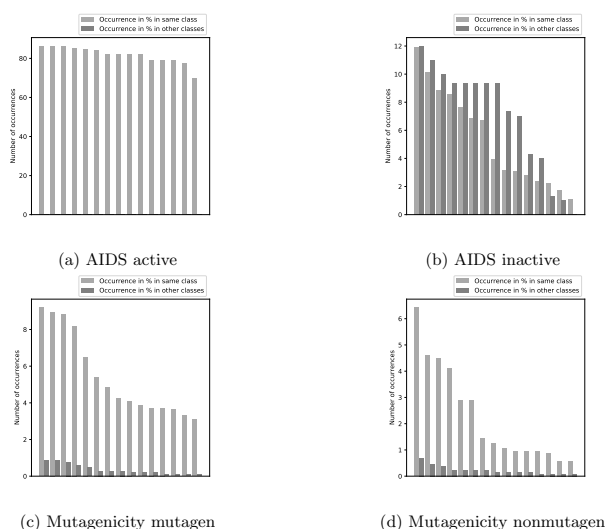


Fig. 5.3: Percentage of frequency of the final c matching-graphs in the test set. Bars in light gray show the frequency in the correct class while the darker bars show the frequency in the incorrect class.

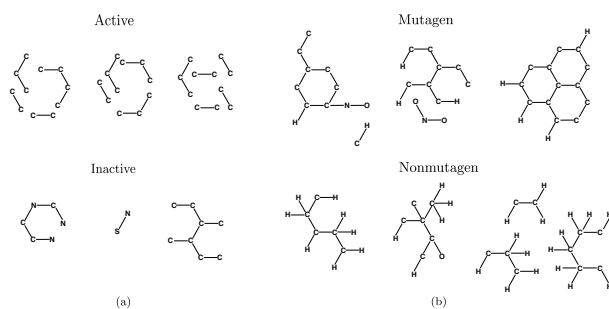


Fig. 5.4: The three matching-graphs with the best quality for the AIDS data set (a) and the Mutagenicity data set (b).

5.4 Maximum Common Subgraph

In this section, we provide some basic definitions and explanations of concepts used in the remainder of this chapter (as an addition to the concepts already introduced and discussed in Chapter 2), namely the *maximum common subgraph* (MCS).

Let $g = (V, E, \mu, \nu)$ and $g' = (V', E', \mu', \nu')$ be graphs, a graph $CS(g, g') = (V_{cs}, E_{cs}, \mu_{cs}, \nu_{cs})$ is called a *common subgraph* of g and g' if $CS(g, g')$ is actually a subgraph of both g and g' . The largest common subgraph $CS(g, g')$ with respect to the cardinality of the node set $|V|$ is referred to as a maximum common subgraph of g and g' (denoted by $MCS(g, g')$ from now on). In general, the $MCS(g, g')$ needs not to be unique, i.e., there might be more than one MCS of identical size for two given graphs g and g' .

We now revisit the distinction between induced and non-induced subgraphs. Depending on the definition actually employed, we either have a *maximum common induced subgraph* (MCIS) or a *maximum common edge subgraph* (MCES). In the former case (MCIS) we require that the resulting MCS is an induced subgraph to both graphs g and g' and in the latter case (MCES) the resulting MCS is not necessarily induced to both graphs g and g' . By definition, the size of the MCES is always greater than, or equal to, the size of the MCIS.

In Figure 5.5 we show an example of two graphs g and g' as well as their resulting MCIS(g, g') and MCES(g, g'). It is clearly visible that MCIS(g, g') is an induced subgraph to both graphs g and g' , while MCES(g, g') is actually a non-induced subgraph of g (due to the missing edge $(0, 1)$).

Another important distinction, that is often made in MCS algorithms, is whether or not the resulting MCS of two graphs must be connected. In the former case the MCS consists of one single connected component only, while in the latter case the MCS is also allowed to consist of more than one connected component. Note that this distinction is independent of the MCS definition actually employed (i.e., MCIS or MCES).

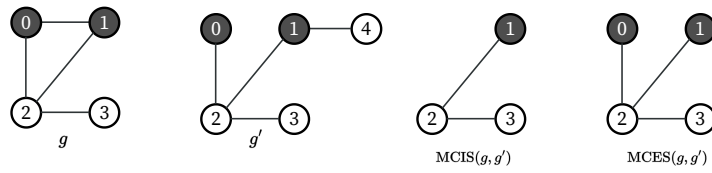


Fig. 5.5: Two graphs g and g' and the maximum common node induced subgraph, MCIS(g, g'), and the maximum common edge subgraph MCES(g, g').

5.5 Relation Between MCS and Matching-Graphs

After providing empirical evidence that the pruned version of the matching-graphs indeed produces relevant subgraphs of the originals, we now explore whether or not it is possible to extend the definition of the matching-graph to fit the definition of the maximum common subgraph (MCS). For more than two decades it is known that graph edit distance computation is equivalent to the MCS problem when used with the following set of cost functions [203].

- $c(u \rightarrow u') = \begin{cases} 0 & , \text{ if } \mu(u) = \mu'(u') \\ \infty & \text{ otherwise} \end{cases} \forall u \in V \text{ and } u' \in V'$
- $c(u \rightarrow \varepsilon) = c(\varepsilon \rightarrow u') = 1, \forall u, u' \in V, V'$
- $c((u, v) \rightarrow (u', v')) = \begin{cases} 0 & , \text{ if } \nu((u, v)) = \nu'((u', v')) \\ \infty & \text{ otherwise} \end{cases} \forall (u, v) \in E \text{ and } (u', v') \in E'$
- $c((u, v) \rightarrow \varepsilon) = c(\varepsilon \rightarrow (u', v')) = 0, \forall (u, v) \in E \text{ and } (u', v') \in E'$

This set of costs, in combination with exact graph edit distance enforces a structural integrity of the graph, because substitutions of unequally labeled nodes and edges are forbidden (by making them cost ∞). In other words, only equally labeled nodes and edges are substituted with zero cost, while both node insertions and deletions have the same unit cost. Hence, $d_{\lambda_{\min}}(g, g')$ counts the number of nodes that need to be deleted and inserted in g and g' , respectively. It follows that the number of nodes of $\text{MCS}(g, g')$ can be computed via

$$|\text{MCS}(g, g')| = \frac{|V| + |V'| - d_{\lambda_{\min}}(g, g')}{2} \quad (5.2)$$

This procedure does not necessarily assume that the underlying MCS is an induced subgraph of both graphs g and g' (due to the zero cost of edge deletion and insertion). That is, we are actually referring to the concept of MCES.

Putting together the definition of a subgraph, the definition of a matching-graph, and the above defined cost model, it becomes clear that the proposed matching-graphs can be seen as a generalization of the concept of a common subgraph. A common subgraph of two graphs consists of nodes which occur identically in both graphs. In a matching-graph, however, a node is incorporated whenever the corresponding node is actually

substituted with another node – and these substitutions can, in general, also occur on nodes with unequal labels.

In the present section we compute the matching-graphs defined in Section 5.2 by means of the cost model proposed in [203]. Note, that in case only identical substitutions are allowed (i.e., $(u \rightarrow u')$ is only possible if $\mu(u) = \mu'(u')$) the two resulting matching-graphs are indeed identical, i.e., $m_{g \times g'} = m_{g' \times g}$. That is, because the nodes of the resulting matching-graphs only reflect substitutions with zero cost. This is, by definition, actually the case in our scenario and thus, we only consider one matching-graph $m_{g \times g'}$ per edit path in the following.

This means, that if one would employ an exact graph edit distance algorithm for the computation of the matching-graphs, one would obtain matching-graphs that are equal to the MCS (or more precisely to the MCES) of the underlying graphs. Such an approach would, however, be questionable, as it does not solve the general efficiency problems of MCS computation.

Thus, in the present chapter we switch to the approximation BP, which always yields a complete and admissible edit path in polynomial time that can be used to create the matching-graphs. This is actually a crucial advantage of the algorithm BP compared to other fast graph matching methods which either provide an invalid edit path or no edit path at all.

Remember that algorithm BP potentially delivers a suboptimal edit path with a higher cost than the optimal edit path. In conjunction with the cost model defined above this means that the suboptimal edit path found by BP must not necessarily contain all node substitutions actually possible. Hence, the size of the matching-graph obtained is (in the general case) smaller than, or equal to, the size of the MCES as defined in Equation 5.2.

However, by restricting both the cost model (as defined above) and the graphs so that they have labeled nodes and unlabeled edges only, the distance d_{BP} is equal to the exact edit distance $d_{\lambda_{\min}}$. Hence, for this special type of graphs, the size of the novel matching-graphs is actually equal to the size of the exact MCES. Moreover, as the size of the MCES builds an upper bound of the size of the MCIS, we can conclude that in this special case the size of the matching-graphs also builds an upper bound of the MCIS.

Moreover, due to the suboptimal nature of the underlying algorithm BP, the procedure can also lead to matching-graphs that consist of more than one connected component, even when the optimal MCS would consist of one connected component only.

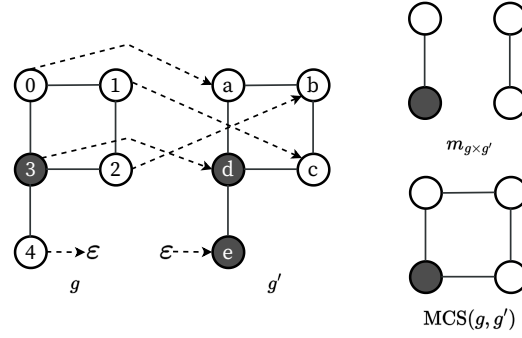


Fig. 5.6: Two graphs g, g' and a possible edit path $\lambda(g, g')$, with the corresponding matching-graph $m_{g \times g'}$ and the optimal MCS.

In Figure 5.6 we show an example of this phenomenon. We show two graphs g and g' , a possible edit path $\lambda(g, g') = \{0 \rightarrow a, 1 \rightarrow c, 2 \rightarrow b, 3 \rightarrow d, 4 \rightarrow \varepsilon, \varepsilon \rightarrow e\}$ as well as the resulting matching-graph $m_{g \times g'}$ and the MCS(g, g'). With the cost model described above, the cost of this edit path is equal to 2 (one node deletion and one node insertion). Obviously, by taking the global edge structure into account, an optimal edit path would carry out the substitutions $(1 \rightarrow b)$ and $(2 \rightarrow c)$, rather than $(1 \rightarrow c)$ and $(2 \rightarrow b)$, respectively. Although the number of nodes in $m_{g \times g'}$ is equal to the number of nodes in the MCS, we observe that $m_{g \times g'}$ is disconnected, whereas the MCS consists of a single connected component.

For very similar reasons, we might obtain isolated nodes in our matching-graphs. In the original implementation [35, 38] the isolated nodes are always removed. However, in the present application of MCS approximation it is not entirely evident whether or not these isolated nodes should be removed from the resulting matching-graphs. If we leave isolated nodes, we might obtain matching-graphs whose sizes are closer to the sizes of the MCS. On the other hand, by deleting isolated nodes one obtains subgraphs that are better connected and thus possibly better reflect the structural similarity of the graphs.

For the remainder of this chapter we denote the MCS approximated by means of matching-graphs by $\text{MCS}_{mg(L)}$ and $\text{MCS}_{mg(D)}$ depending on whether isolated nodes are left or deleted, respectively.

Table 5.2: The total number of graphs and classes per data set, as well as the average number of nodes and edges per graph.

Data set	Graphs	Classes	$\emptyset V $	$\emptyset E $
AIDS	2,000	2	15.69	16.20
Mutagenicity	4,337	2	30.32	30.77
NCI1	4,110	2	29.87	32.30
COX-2	467	2	41.22	43.45
PTC(MR)	344	2	14.29	14.69
PROTEINS	1,113	2	39.06	72.82
ENZYMES	600	6	32.63	62.14

5.6 Experimental Evaluation

The overall aim of the experimental evaluation is to research whether or not matching-graphs can be used as an approximation for the MCS of two graphs. In order to answer this question, we compare the matching-graphs to an exact MCS algorithm (described in Section 5.6.1), as well as to an existing MCS approximation (described in Section 5.6.2).

We evaluate the proposed approach on seven data sets (namely AIDS, Mutagenicity, NCI1, COX-2, PTC(MR), PROTEINS and ENZYMES) as described in Chapter 3. Some basic characteristics of the data sets such as the number of graphs, the number of classes, as well as the average number of nodes and edges, are shown in Table 5.2.

5.6.1 Comparison with Exact MCS Computation

First, we compare our novel approximation $\text{MCS}_{mg(\cdot)}(g, g')$ based on matching-graphs with the widely used MCSplit algorithm [197], that produces exact solutions $\text{MCS}_e(g, g')$. Note that the MCSplit algorithm creates – in contrast to our novel approach – induced maximum common subgraphs (which are potentially disconnected). Moreover, we can expect that the size of $\text{MCS}_{mg(L)}$ is – on the tested data sets with unlabeled edges – equal to the size of the MCES. Thus, our novel approximation builds an upper bound to the size of the true MCIS.

Due to the exponential time complexity of MCSplit and in order to make the experiment feasible, we randomly select a subset of $n = 50$ graphs per data set. Next we build all possible pairs, which results in $\frac{n(n-1)}{2} = 1,225$ pairs of graphs, for which the MCS is computed.

For practical reasons, we set a timeout of 1,000 seconds per computation.

On all data sets except ENZYMES, where about 55% of all computations lead to a timeout, the vast majority of MCS computations can be carried out without timeout. That is, on PROTEINS more than 85% and on the remaining data sets about 95% to 99.9% of all exact MCS computations can actually be conducted.

In Table 5.3 we show the average run time per MCS computation in seconds. On all data sets it is clearly visible that the exact computation to obtain $\text{MCS}_e(g, g')$ is by multiple orders of magnitude slower than the computation of the approximation $\text{MCS}_{mg(\cdot)}(g, g')$. On average over all data sets, the exact computation takes about 110 seconds per graph pair – our novel approximation needs for the same procedure only 0.05 seconds. The most significant difference can be observed on the PROTEINS data set, where our approximation is about 4,900 times faster than MCSplit.

Next, we compare the sizes of the MCS obtained by MCSplit and the matching-graphs. As a size comparison metric, we compute the error E produced by approximation $\text{MCS}_{mg(\cdot)}(g, g')$ w.r.t. the exact computation $\text{MCS}_e(g, g')$. Formally, the error E is defined as

$$E = \frac{||\text{MCS}_e(g, g')| - |\text{MCS}_{mg(\cdot)}(g, g')||}{|\text{MCS}_e(g, g')|} \quad (5.3)$$

A value of $E = 0$ means that the created approximation $\text{MCS}_{mg(\cdot)}(g, g')$ has equal size as the exact solution, and the maximum error of $E = 1$ means, that the approximation $\text{MCS}_{mg(\cdot)}(g, g')$ refers to an empty graph.

In Table 5.3, we observe that the average error E varies quite strongly depending on the actual data set. Moreover, the error also depends on whether or not isolated nodes are removed from the matching-graphs. On five out of seven data sets the error observed with $\text{MCS}_{mg(L)}$, is – in part substantially – lower than the error obtained by using $\text{MCS}_{mg(D)}$. Actually, the deletion of isolated nodes leads to a lower error on the PROTEINS and ENZYMES data sets only. Also the average error measured across all data sets is significantly lower when using $\text{MCS}_{mg(L)}$ instead of $\text{MCS}_{mg(D)}$.

In Figure 5.7 we show a qualitative comparison of the sizes of MCS_e and the sizes of $\text{MCS}_{mg(L)}$ and $\text{MCS}_{mg(D)}$ on the PTC(MR) data set (we observe similar plots on the other data sets). Each dot in the scatterplot represents a pair of graphs g, g' and for each of these pairs, we show on the x-axis the size of the exact $\text{MCS}_e(g, g')$ and on the y-axis the size of the approximate $\text{MCS}_{mg(\cdot)}(g, g')$. A dot on the diagonal line indicates that the size of our novel approximation is equal to the size of the exact result. Dots above the diagonal indicate that $\text{MCS}_{mg(\cdot)}(g, g')$ is larger than $\text{MCS}_e(g, g')$,

Table 5.3: A comparison of the MCSplit algorithm (MCS_e) with our approximations $\text{MCS}_{mg(\cdot)}$ based on matching-graphs in terms of the average run time per MCS computation in seconds, and the average error E . We distinguish between matching-graphs with and without isolated nodes $\text{MCS}_{mg(L)}$ and $\text{MCS}_{mg(D)}$, respectively. **Bold** values indicate lower errors.

Data Set	Time (s)		Average Error E	
	MCS_e	$\text{MCS}_{mg(\cdot)}$	$\text{MCS}_{mg(L)}$	$\text{MCS}_{mg(D)}$
AIDS	51.85	0.03	0.21	0.49
Mutagenicity	61.3	0.06	0.17	0.62
NCI1	108.12	0.05	0.15	0.67
COX-2	21.11	0.08	0.09	0.35
PTC(MR)	1.17	0.01	0.15	0.21
PROTEINS	147.44	0.03	0.29	0.28
ENZYMES	379.97	0.08	0.35	0.21
Mean	110.14 ± 128.96	0.05 ± 0.03	0.20 ± 0.09	0.40 ± 0.19

and likewise, each dot below the diagonal refers to a pair of graphs (g, g') with $\text{MCS}_{mg(\cdot)}(g, g') < \text{MCS}_e(g, g')$.

As expected, we observe that the size of matching-graphs with isolated nodes is always larger than, or equal to, the size of the true MCS. When removing the isolated nodes from the matching-graphs, we observe both over- and underestimations of the actual size of the exact MCS. These underestimations of the actual size of the MCIS are reflected in the larger errors observed for $\text{MCS}_{mg(D)}$ in Table 5.3.

5.6.2 Classification with MCS Based Dissimilarities

In this section, we evaluate the approximation $\text{MCS}_{mg(\cdot)}$ in the context of a classification scenario. We employ three dissimilarity metrics, namely d_{UGU} [203], d_{WGU} [204] and d_{MCS} [201] in combination with a k -nearest neighbour classifier (k -NN). The k -NN is particularly well suited for our purposes, as it directly operates on distance metrics without further training. All of the dissimilarity metrics are inverse proportional to the size of the MCS of the two graphs. This means that the larger the MCS, the smaller the dissimilarity and vice versa. Formally, given two graphs g and g' and their corresponding $\text{MCS}(g, g')$, the metrics are defined as follows

$$d_{\text{UGU}}(g, g') = |g| + |g'| - 2|\text{MCS}(g, g')| \quad (5.4)$$

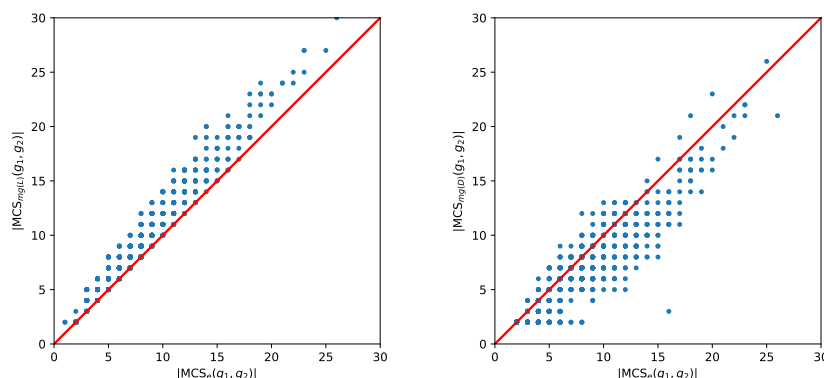


Fig. 5.7: Comparisons of the size of the exact MCS_e (x-axis) and the size of the approximated $\text{MCS}_{mg(\cdot)}$ (y-axis). We show both solutions, with and without isolated nodes.

$$d_{\text{WGU}}(g, g') = 1 - \frac{|\text{MCS}(g, g')|}{|g| + |g'| - |\text{MCS}(g, g')|} \quad (5.5)$$

$$d_{\text{MCS}}(g, g') = 1 - \frac{|\text{MCS}(g, g')|}{\max\{|g|, |g'|\}} \quad (5.6)$$

Due to their high computational complexity, exact MCS algorithms are not applicable for this evaluation. Therefore, we switch to a state of the art approximation algorithm stemming from the chemistry development kit (CDK) [198]. We denote this reference approximation algorithm with CDK and the resulting subgraph as MCS_{CDK} .

For this evaluation, we define a random 3:1:1 split for each data set for training, validation, and testing, respectively. Actually, only two parameters are tuned on the validation sets. First, for the computation of the $\text{MCS}_{mg(\cdot)}(g, g')$, we validate whether or not the isolated nodes are deleted. Second, for the k -NN classifier we optimize the number of neighbors k considered ($k \in \{1, 3, 5, 7\}$).

In Table 5.4 we show the classification accuracies achieved on the test sets. For each metric, the bold number indicates whether the reference system that operates on MCS_{CDK} or our novel system based on $\text{MCS}_{mg(\cdot)}$ performs better.

For the metric d_{UGU} we observe that our system outperforms the reference system on all data sets but PTC(MR). Two of these six improvements

Table 5.4: Classification accuracies of three metrics (d_{UGU} [203], d_{WGU} [204] and d_{MCS} [201]) that are based on the size of the MCS. For the reference system we use an approximation MCS_{CDK} [198] and compare it with our matching-graphs $\text{MCS}_{\text{mg}(\cdot)}$. Symbol \circ indicates a statistically significant improvement over the reference system (using a Z-test at significance level $\alpha = 0.05$). The **bold** number indicates the better performance for the corresponding metric.

Data Set	$k\text{-NN}(d_{\text{UGU}})$		$k\text{-NN}(d_{\text{WGU}})$		$k\text{-NN}(d_{\text{MCS}})$	
	MCS_{CDK}	$\text{MCS}_{\text{mg}(\cdot)}$	MCS_{CDK}	$\text{MCS}_{\text{mg}(\cdot)}$	MCS_{CDK}	$\text{MCS}_{\text{mg}(\cdot)}$
AIDS	94.0	100.0 \circ	96.0	100.0 \circ	98.0	100.0 \circ
Mutagenicity	68.4	72.4 \circ	70.2	70.2	72.7	72.0
NCI1	63.7	67.3	72.3	69.9	71.0	69.0
COX-2	78.5	80.7	78.5	82.8 \circ	78.5	79.6
PTC(MR)	57.1	55.7	47.6	48.6	50.0	48.6
PROTEINS	69.2	71.4	71.4	70.6	71.9	80.4 \circ
ENZYMES	20.8	22.5	19.2	21.7	15.8	20.0

Table 5.5: A comparison of the run times on all data sets, of the reference approximation MCS_{CDK} compared to our matching-graphs $\text{MCS}_{\text{mg}(\cdot)}$.

Data Set	Time (s)		
	MCS_{CDK}	$\text{MCS}_{\text{mg}(\cdot)}$	Gain (%)
AIDS	5,621	1,434	292%
Mutagenicity	100,024	29,397	240%
NCI1	37,445	21,793	72%
COX-2	1,149	586	96%
PTC(MR)	209	56	273%
PROTEINS	71,455	4,628	1444%
ENZYMES	7,109	940	656%

are statistically significant. For d_{WGU} our system outperforms the reference system on four data sets (twice with statistical significance). Using the same metric our approach performs equally well as MCS_{CDK} on Mutagenicity and only worse on NCI1 and PROTEINS (yet not statistically significant). For the last metric, d_{MCS} , we observe four improvements and three deteriorations when compared to the reference system. Two of these improvements, but none of the deteriorations, are statistically significant. Overall, we can report 14 improvements (6 being statistically significant) for all three metrics using the MCS approximation based on matching-graphs compared to the reference approximation CDK.

Finally, in Table 5.5, we compare the total run times to create both MCS approximations MCS_{CDK} and $\text{MCS}_{\text{mg}(\cdot)}$ for all data sets. It is clearly

visible that the proposed approximation is substantially faster than the algorithm CDK on all data sets. The improvement of the run time is the lowest on the NCI1 data set, where a gain in the run time of about 72% is observed. On the PROTEINS data set, on the other hand, where the gain in run time is about 1,444%, we observe the largest speed-up.

5.7 Conclusion and Future Work

In various pattern recognition frameworks, it is crucial to find similarities between pairs of entities. It is especially interesting to discover relevant substructures in a given class of graphs, in order to understand which graph substructures actually make up a class of patterns. In this chapter we explore the viability of matching-graphs for discovering substructures in general as well as the relation of matching-graphs to the well known concept of maximum common subgraph (MCS).

In order to do this, we perform two separate evaluations. First, we advance the creation of matching-graphs by means of an iterative algorithm. That is, starting with an initial set of matching-graphs, novel sets of matching-graphs are iteratively created through further matchings of matching-graphs. Furthermore, we select the matching-graphs of the highest quality after each iteration, in order to generate a set of matching-graphs that are as small as possible "cores" that represent a certain class.

In an experimental evaluation on two graph data sets, we empirically confirm that our novel approach is able to produce matching-graphs that accurately represent significant and frequent substructures of a given class. Moreover, through a qualitative evaluation we confirm that our novel procedure offers high potential for detecting novel and relevant substructures in sets of graphs. To the best of our knowledge this is the first time that a graph matching algorithm is employed for this specific task.

Second, we propose a specific version of the matching-graphs, in order to compare them to the problem of finding the MCS of two graphs, which is of great interest in several fields of pattern recognition, such as bioinformatics or chemoinformatics. Due to the exponential time complexity of exact solutions for the MCS problem, we focus on researching approximations that can deliver reasonably close results in a shorter time. We introduce matching-graphs as an approximate solution for the MCS problem, which represent a generalization of the MCS concept and can be used to approximate the MCS by means of suboptimal graph matchings.

The second experimental evaluation on seven graph data sets shows that matching-graphs generally outperform a state-of-the-art MCS approximation in terms of both classification accuracy and computation time. We also present an iterative algorithm for creating matching-graphs, which demonstrates the potential for detecting novel and relevant substructures in sets of graphs. This is the first time that a graph matching algorithm is employed for this specific task.

Several rewarding avenues for future work include evaluating different graph edit distance approximations for producing other matching-graphs and thus potentially more relevant substructures or better MCS approximations. Additionally, future work should involve adapting the process of creating matching-graphs to better fit the MCS problem and exploring the applicability of our novel approximation in the case of graphs with labeled edges. Furthermore, we plan to compare our method with well-known graph mining algorithms, and integrate the improved matching-graphs into a classification scheme, such as a distance-based classifier or a subgraph-kernel.

Matching-Graphs for Graph Augmentation

6

With matching-graphs in hand,
augmentation takes a stand.
From small data sets to
ensemble delight, it's a
graphically enhancing sight.

ChatGPT

6.1 Introduction

In 1985, Robert Mercer made his famous comment “There is no data like more data” [205] – Some researchers even go one step further by arguing that having more data is more important than developing better algorithms [206]. At least one can agree that labeled training data is one of the most pivotal prerequisites for the development and evaluation of supervised pattern recognition and machine learning algorithms. This is particularly true for deep learning methods [207, 208], which typically perform better the more examples of a given phenomenon a network is exposed to.

However, in real-world applications, we often face the limitation of available training data for various reasons. Some of these reasons include data scarcity or access restrictions, such as in domains with rare diseases or events, where occurrences of phenomena of interest are naturally rare, making it difficult to collect enough training data [209–211], or in domains such as the defence sector, where data access is restricted for security reasons. Another factor is the cost of data collection, as acquiring a large amount of high quality labeled data can be expensive, time consuming and resource intensive [212–214]. Data privacy and security concerns also come into play,

with privacy regulations such as GDPR¹ potentially limiting the amount of data available for training, especially when dealing with sensitive information such as medical records or personally identifiable information. Finally, imbalanced data presents another challenge, as real-world data is often imbalanced with some classes having far fewer instances than others. This results in a limited amount of training data for minority classes, and leads to poor performance on those classes when using standard machine learning algorithms [215–217].

Data augmentation techniques are a significant help with this problem. These techniques increase the amount of data by adding modified copies of the already existing entities to the training set. In the fields of computer vision [218–222] and natural language processing [223–227], for instance, data augmentation is widely used. However, the vast majority of these augmentation algorithms are mainly suitable for statistical data (e.g. feature vectors). Due to the structural nature of graphs, there are comparatively few approaches to graph augmentation.

Main contribution of the present chapter is that we introduce and research a novel and systematic way of increasing the amount of training data in graph-based pattern recognition scenarios. The basic idea is to compute matching-graphs for any pair of training graphs available. By additionally slightly randomizing the whole process of creating the matching-graphs, the amount of training data can hereby be increased to virtually any size. This allows for the creation of more diverse matching-graphs that extend beyond being mere subgraphs of the original graphs. Our first hypothesis is that this novel method provides a natural way to create realistic and relevant graphs that are actually useful during training of pattern recognition algorithms.

The proposed process of creating matching-graphs in order to enlarge training sets is similar in spirit to existing graph augmentation approaches. However most of these approaches augment the graphs by altering edge information only [228–231]. Moreover, these approaches often rely on a single sample of a graph. In our approach, however, we generate new graphs based on the information captured in the edit path resulting from matching pairs of graphs. The graphs generated this way include both edge and node modifications. In [232], an adaptation of the *Mixup algorithm* [233] for graphs is proposed, which is quite similar to our method. In addition, there are several neural network based approaches that are

¹<https://gdpr.eu/>

somewhat similar to our proposal but do not have the main goal of improving graph classification. Several approaches attempt, for instance, to leverage the power of *Variational Auto-Encoders* (VAEs) [234] for graph generation [235–237]. Last but not least, the success of *Generative Adversarial Networks* (GANs) [238] for image generation led to an adaptation for graph generation as well [239–241].

It is clear that data augmentation plays a crucial role in improving the performance of classification systems (especially for methodologies like *Graph Neural Networks* (GNNs) [17], which are highly dependent on the availability of data) by expanding and diversifying the available training data. *Ensemble learning* [242–244] is another methodology which potentially profits from data augmentation. Ensemble classifiers consist of multiple base classifiers, each trained on a different subset of the training data or using different learning algorithms. By combining the predictions of these individual classifiers, ensemble methods can often achieve better performance than any single classifier alone. A key factor in the success of ensemble classifiers is the diversity of the individual models, which can be achieved through data augmentation [245].

Therefore, our second hypothesis is that the large amount of possible matching-graphs in conjunction with an ensemble ensures the diversity of the individual classifiers and finally allows to build a robust ensemble. When a large and diverse data set is available, ensemble classifiers can take full advantage of data augmentation techniques to create multiple, as distinct as possible, subsets of the training data. These subsets can then be used to train individual base classifiers, ensuring that each model learns from a unique perspective of the data. This diversity in the training data helps to reduce overfitting and improve generalization, as the ensemble classifier effectively captures a more comprehensive view of the underlying data distribution. Note that the present chapter is based on three preliminary papers [40–42].

The remainder of the chapter is organized as follows. In Section 6.2, some necessary definitions and concepts are provided. Next, in Section 6.3, we introduce a novel adapted version of the matching-graph, that is based on only a partial edit path. We also show how we can use them to augment a given training set. In Section 6.4 we conduct an exhaustive experimental evaluation to provide empirical evidence that our approach of generating additional training samples is able to improve the classification accuracy of existing classification systems. In Section 6.5 we explore how our novel augmentation method can be applied to the generation of robust and diverse

ensembles. Finally in Section 6.6, we conclude the chapter and discuss some ideas for future work.

6.2 Basic Definitions

In this section, we provide some basic definitions and explanations of concepts used in this chapter, namely *Graph Neural Networks* (GNNs) [17].

GNNs provide a framework to use deep learning on graph structured data. The general goal of GNNs is to learn a vector h_v or h_g to represent each node $v \in V$ of a given graph or the complete graph g , respectively. That is, we obtain an embedding of either single nodes or complete graphs in a vector space. Based on this embedding, the individual nodes or the entire graph can then be classified. In our work, we focus on graph classification only.

In order to learn the graph embedding, we first need to learn the embedding of each node. In order to achieve this goal, GNNs usually follow a neighborhood aggregation strategy. That is, GNNs use a form of *neural message passing* in which messages are exchanged between the nodes of a graph [246]. The general idea is to iteratively update the representation of a node by aggregating the representation of its neighbors. During the i -th message-passing iteration, a hidden embedding $h_v^{(i)}$ that corresponds to each node $v \in V$ is updated according to the aggregated information from its neighborhood. Formally, this can be achieved by two differentiable functions.

- $m_{\mathcal{N}(v)}^{(i)} = \text{AGGREGATE}^{(i)}(\{h_u^{(i)} : u \in \mathcal{N}(v)\})$
- $h_v^{(i+1)} = \text{UPDATE}^{(i)}(h_v^{(i)}, m_{\mathcal{N}(v)}^{(i)})$

where $m_{\mathcal{N}(v)}^{(i)}$ is the "message" that is aggregated from the neighborhood $\mathcal{N}(v)$ of node v , and $\mathcal{N}(v)$ refers to the set of nodes adjacent to v . The feature vector representation of node v at the i -th iteration is $h_v^{(i)}$, and the initial representation $h_v^{(0)}$ is set to the vector $\mu(v)$ containing the original labels of v .

After N iterations of the message passing mechanism, this process produces node embeddings $h_v^{(N)}$ for each node $v \in V$. To get the embedding h_g for the complete graph, so-called *graph pooling* is necessary. Graph pooling combines the individual local node embeddings to one global embedding. That is, the pooling function maps the set of n node embeddings $\{h_{v_1}^{(N)}, \dots, h_{v_n}^{(N)}\}$ to the graph embedding h_g . The pooling function can sim-

ply be a sum (or mean) of the node embeddings, or a more sophisticated function [247].

In the present chapter, we employ three different GNN architectures that are all based on the above mentioned concepts. The first architecture, denoted as *GCN* from now on, contains three Graph Convolutional Layers [31]. A GCN layer takes the node feature matrix and the adjacency matrix as inputs and performs a localized convolution-like operation on the graph. A graph convolutional layer computes the new feature representation for each node by aggregating the features of its neighbors, typically using a weighted sum. The aggregation is followed by a transformation using a learnable weight matrix and an activation function. The second system is the Graph Isomorphism Network, denoted as *GIN*, introduced by Xu et al. [33]. GIN is a graph neural network architecture inspired by the Weisfeiler-Lehman graph isomorphism test [13], and is designed to capture graph structure more effectively than other GNN models. GIN layers aggregate neighbor features using a sum-based aggregation function, with an additional learnable "epsilon" parameter controlling the importance of a node's own features. The third architecture is the *GraphSAGE* network introduced by Hamilton et al. [248]. GraphSAGE is an inductive learning method for graph-structured data, capable of generating embeddings for previously unseen nodes or subgraphs.

All three algorithms are implemented using Pytorch Geometric and for *GIN* and *GraphSAGE* we use the implementations of [249]². For the final graph classification, we add a dropout layer to all three architectures and feed the graph embedding into a fully connected layer.

6.3 Augment Training Sets by Means of Matching-Graphs

Major contribution of the present chapter is that we propose an approach to increase the size of a given training set. The motivation for this increase comes from the fact that with more and diverse training data, one can usually train more robust classifiers. The proposed method for training set augmentation is based on *matching-graphs*.

Matching-graphs are built by extracting information on the matching of pairs of graphs and by formalizing and encoding this information in a data structure. Matching-graphs can be interpreted as denoised core structures of the underlying graphs. The idea of matching-graphs initially emerged

²<https://github.com/diningphil/gnn-comparison>

in [35] where they are employed for improving the overall quality of graph edit distance. The matching-graphs used in this chapter are adapted for graph set augmentation and are created in a slightly different way as originally proposed. The concept of matching-graphs is thoroughly explained in Chapter 4. This chapter extends and adapts the initial concept of a matching-graph for the purpose of graph augmentation.

Formally, we assume k sets of training graphs $G_{\omega_1}, \dots, G_{\omega_k}$ stemming from k different classes $\omega_1, \dots, \omega_k$. For all pairs of graphs stemming from the same class ω_l , the graph edit distance is computed by means of algorithm BP [85]. Hence, we obtain a (sub-optimal) edit path $\lambda(g, g') = \{e_1, \dots, e_s\}$ for each pair of graphs $g = (V, E)$ and $g' = (V', E') \in G_{\omega_l} \times G_{\omega_l}$. Each edit operation $e_i \in \lambda(g, g')$ can either be a substitution, a deletion or an insertion of a node including the corresponding edge edit operation.

Based on the edit path $\lambda(g, g')$ two matching-graphs $m_{g \times g'}$ and $m_{g' \times g}$ can now be built (one based on the source graph g and one based on the target graph g'). In order to create both $m_{g \times g'}$ and $m_{g' \times g}$, we initially define $m_{g \times g'} = g$ and $m_{g' \times g} = g'$. Then, we select a certain percentage $p_1 \in [0, 1]^3$ of all s edit operations available in $\lambda(g, g')$. Hence, we obtain a partial edit path $\tau(g, g') = \{e_1, \dots, e_t\} \subseteq \lambda(g, g')$ with $t = \lfloor p_1 \cdot s \rfloor$ edit operations only. Next, each edit operation $e_i \in \tau(g, g')$ is applied on graphs $m_{g \times g'}$ or $m_{g' \times g}$ according to the following rules.

- Case 1.** If e_i refers to a substitution ($v \rightarrow v'$), it is applied on both graphs $m_{g \times g'}$ and $m_{g' \times g}$. More precisely, the labels of the matching nodes $v \in V$ and $v' \in V'$, are exchanged in both $m_{g \times g'}$ and $m_{g' \times g}$. Note that this operation shows no effect when the two labels of the involved nodes are identical.
- Case 2.** If e_i refers to a deletion ($v \rightarrow \varepsilon$), e_i is applied on $m_{g \times g'}$ only, meaning that $v \in V$ is deleted in $m_{g \times g'}$.
- Case 3.** If e_i refers to an insertion ($\varepsilon \rightarrow v'$), e_i is applied on $m_{g' \times g}$ only. This means that the node $v' \in V'$ that is inserted according to the edit operation, is deleted in $m_{g' \times g}$ instead.

The rationale for the third rule is as follows. When inserting a node $v' \in V'$, it is not necessarily clear how v' should be connected to the remaining parts of the current graph (since it is possible that not all necessary edit operations have been carried out or have not been selected at all). By

³We use here the expression p_1 (with subscript 1), because later in the paper we will introduce a second probability p_2 .

deleting the node $v' \in V'$ in $m_{g' \times g}$ rather than inserting it in $m_{g \times g'}$ we can avoid this problem quite easily.

Both matching-graphs represent intermediate graphs between the two underlying training graphs. If p_1 is set to 1.0, all edit operations from the complete edit path $\lambda(g, g')$ are considered during the matching-graph creation. Note, however, that according to our rules, deletions and insertions are uniquely applied on the source or the target graph, respectively. Hence, in this particular parameter setting we obtain two matching-graphs that are subgraphs from the original graphs. With parameter values $p_1 < 1.0$, however, we obtain matching-graphs in which possibly some of the nodes are either deleted from g or g' and some other nodes are potentially altered according to their labeling (due to the applied substitutions).

Note that one can extract several partial edit paths $\tau(g, g')$ from one edit path $\lambda(g, g')$ using different values of p_1 . This in turn results in several matching-graphs based on the same edit path. In theory, one can create as many matching-graphs as edit operations are available in $\lambda(g, g')$.

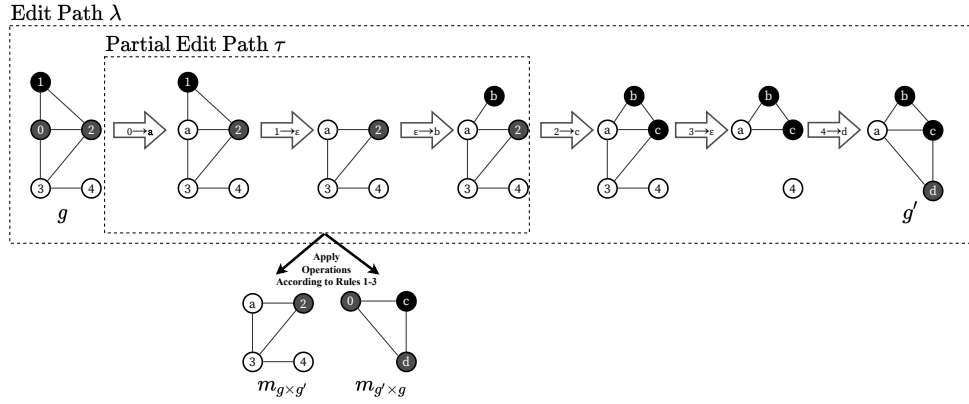


Fig. 6.1: An example of a complete edit path λ , a partial edit path τ , and the resulting matching-graphs $m_{g \times g'}$ and $m_{g' \times g}$.

Example 6.1. Figure 6.1 shows a visual example of the graph edit distance between two graphs g and g' and two possible resulting matching-graphs $m_{g \times g'}$ and $m_{g' \times g}$. The corresponding edit path is $\lambda = \{(0 \rightarrow a), (1 \rightarrow \varepsilon), (\varepsilon \rightarrow b), (2 \rightarrow c), (3 \rightarrow \varepsilon), (4 \rightarrow d)\}$. The possible matching-graphs $m_{g \times g'}$ and $m_{g' \times g}$, are created with $p_1 = 0.5$, resulting in the partial edit

path $\tau(g, g') = \{(0 \rightarrow a), (1 \rightarrow \varepsilon), (\varepsilon \rightarrow b)\}$, that consists of $t = 3$ edit operations. In this example it is clearly visible, that neither $m_{g \times g'}$ nor $m_{g' \times g}$ is a subgraph of g or g' , respectively.

Note that the proposed process can lead to isolated nodes (as observed, for example, in the second last graph in Figure 6.1). Based on the reasoning that we want to create graphs with nodes that are actually connected, we remove disconnected nodes from the matching-graphs whenever they occur.

Based on the process of creating two matching-graphs for pairs of graphs, we can now define an algorithm to augment a given training set with additional graphs. Algorithm 5 takes k sets of training graphs $G_{\omega_1}, \dots, G_{\omega_k}$ stemming from k different classes $\omega_1, \dots, \omega_k$ as input. The two **for** loops accomplish the following. For all pairs of graphs g, g' stemming from the same class ω_i , two matching-graphs $m_{g \times g'}$ and $m_{g' \times g}$ are built and added to the corresponding set of graphs (labeled with G_{ω_i}). Assuming n training graphs per class G_{ω_i} this results in $k \cdot n(n - 1)$ matching-graphs in total, which are directly used to augment the corresponding training sets $G_{\omega_1}, \dots, G_{\omega_k}$.

In Algorithm 5, the probability $p_1 \in [0, 1]$ used for the creation of the matching-graphs (see Line 5), is redefined for each iteration. This can be achieved, for example, by choosing p_1 randomly in each iteration. However, the probability p_1 can also be set to a fixed value for all iterations. In addition to this, inside the second **for** loop, just before the definition of p_1 , a further **for** loop could be defined, such that even more than one matching-graph could be created for each pair of graphs.

Algorithm 5: Graph Augmentation Algorithm

input : sets of graphs from k different classes $\mathcal{G} = \{G_{\omega_1}, \dots, G_{\omega_k}\}$
output: same sets augmented by matching-graphs

```

1 foreach set  $G_{\omega_i} \in \mathcal{G}$  do
2    $M = \{\}$ 
3   foreach pair of graphs  $g, g' \in G_{\omega_i} \times G_{\omega_i}$  do
4     Compute  $\lambda(g, g') = \{e_1, \dots, e_s\}$ 
5     Define  $p_1$  in  $[0, 1]$ 
6     Define  $\tau$  by selecting  $\lfloor p_1 \cdot s \rfloor$  edit operations from  $\lambda$ 
7     Build both matching-graphs  $m_{g \times g'}$  and  $m_{g' \times g}$  according to  $\tau$ 
8      $M = M \cup \{m_{g \times g'}, m_{g' \times g}\}$ 
9   end
10   $G_{\omega_i} = G_{\omega_i} \cup M$ 
11 end
```

6.4 Experimental Evaluation: Augmenting Training Sets

The overall aim of the following experimental evaluation is to answer the question, whether or not matching-graphs can be beneficially employed as a technique for automatically augmenting training sets of graphs. To this end, we perform two separate experimental evaluations. First, in Section 6.4.1, we use the proposed technique to augment very small training sets of graphs and verify whether this helps to substantially increase the accuracy of a classification algorithm. Second, in Section 6.4.2, we research whether or not matching-graphs can be used to make the training of graph neural networks more robust.

6.4.1 Augment Small Training Sets

For the first experiment (as described in [40]), we artificially decrease the size of all data sets. This is achieved by randomly selecting 10 training graphs per class for each data set. In order to avoid overly simple or very difficult data sets (that might be created by random chance), we repeat the random process of creating small data sets 20 times. The same accounts for training set augmentation by means of our matching-graphs which is also repeated 20 times.

As basic classification system a Support Vector Machine (SVM) that exclusively operates on a similarity kernel $\kappa(g, g') = -d_{BP}(g, g')$ is used [62]. Note that any other data-driven classifier could be used in our evaluation as well. However, we feel that the SVM is particularly suitable for our evaluation because of its pure and direct use of the underlying distance information.

The primary reference system is trained on the reduced training data, denoted as $SVM_R(-d_{BP})$. Our novel approach, denoted as $SVM_{R+}(-d_{BP})$ is trained on the same training samples of the reduced sets but uses also the created matching-graphs. For the sake of completeness we also compare our novel framework with a secondary reference system, viz. an SVM that has access to the full training sets of graphs (before the artificial reduction is carried out), denoted as $SVM_F(-d_{BP})$.

The evaluation is performed on four data sets (namely Mutagenicity, NCI1, COX-2 and PTC(MR)) as described in Chapter 3.

6.4.1.1 Validation of Metaparameters

In Table 6.1 an overview of all parameters that are optimized is presented. For algorithm BP, that approximates the graph edit distance, the cost for

Table 6.1: Description of all parameters and the evaluated values.

Parameter	Description	Evaluated Values
β	Scales node and edge costs for graph edit distance	$\{0.05, 0.10, \dots, 0.95\}$
p_1	Relative amount of edit operations selected λ	$\{0.25, 0.50, 0.75, 1.0\}$
C	Weighting parameter of the SVM	$\{10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 5 \cdot 10^{-3}, 10^{-2}, 5 \cdot 10^{-2}, 10^{-1}, 5 \cdot 10^{-1}, 10^0, 10^1, 10^2\}$

node and edge deletions, as well as a weighting parameter $\beta \in [0, 1]$ that is used to trade-off the relative importance of node and edge edit costs are often optimized [85, 182]. However, for the sake of simplicity we employ unit cost of 1.0 for deletions and insertions of both nodes and edges and optimize the weighting parameter β only (on all data sets).

For the creation of the matching-graphs – actually also dependant on the cost model – the same weighting parameter is independently optimized. Additionally, we optimize the probability p_1 (Algorithm 5, Line 5) of the edit path operations that are used for the matching-graph creation (the optimal value p_1 is then fixed for all iterations of the loop on Line 3 in Algorithm 5). For the SVM classifier itself, parameter C is optimized to trade off between the size of the margin and the number of misclassified training examples.

6.4.1.2 Test Results and Discussion

In Figure 6.2 we show the reference accuracies of $\text{SVM}_R(-d_{\text{BP}})$ as well as the accuracies of the proposed system $\text{SVM}_R+(-d_{\text{BP}})$ as bar charts for all 20 random iterations on all four data sets. The iterations are ordered from the worst to the best performing reference accuracy.

On the NCI1 and COX-2 data sets we observe that the SVM that relies on the matching-graphs performs better than, or at least equal as, the reference systems in all iterations. On the other two data sets Mutagenicity and PTC(MR) our system outperforms the reference system in 19 out of 20 iterations. In general, we report substantial improvements over the respective baselines for almost all iterations and data sets.

There is a tendency that the improvement is particularly large in iterations where the reference system performs poorly (most likely due to an

unfortunate random selection of training samples which in turn might lead to overfitting). This is visible on all the data sets, but it is particularly well observable on the COX-2 and PTC(MR) data sets. For example on the COX-2 data set, we outperform the reference system by about 30 percentage points during the first two iterations (from 47.3% to 77.4% and from 51.6% to 82.8%, respectively). On PTC(MR) we see a similar pattern for the first three iterations, where our system increases the accuracy with about 20 percentage points (from 41.4% to 62.8%, from 41.4% to 60% and from 47.1% to 70%). This emphasizes the usefulness of the systematic augmentation by means of matching-graphs, especially when small sets of rather poor training samples are available only.

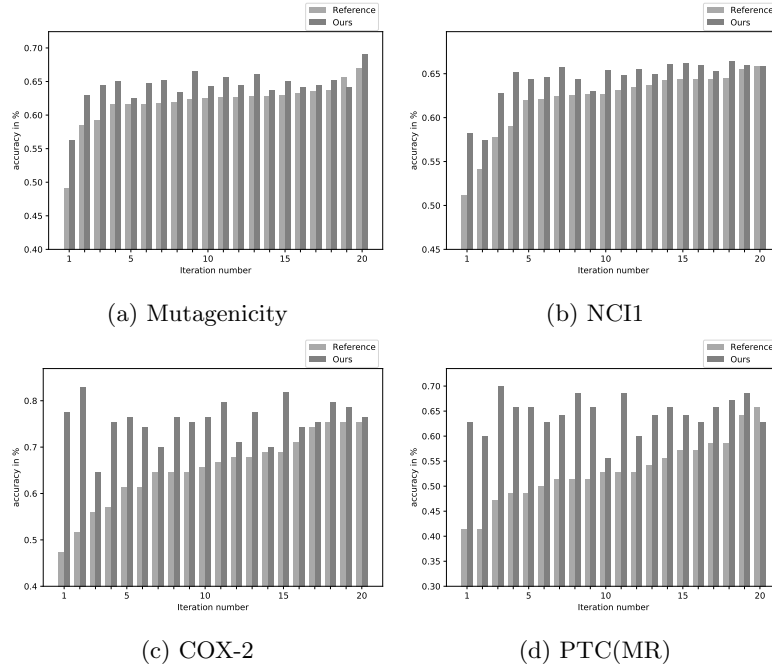


Fig. 6.2: Classification accuracies of all 20 iterations of the reference system (bright bars) compared to our novel system that additionally uses the matching-graphs for training (dark bars).

In Table 6.2 we compare the classification accuracy of our novel system with both reference systems in tabular form. To this end, we aggregate the results of the 20 iterations and show the mean classification accuracies

Table 6.2: Classification accuracies of two reference systems ($\text{SVM}_R(-d_{\text{BP}})$ and $\text{SVM}_F(-d_{\text{BP}})$) and our novel system $\text{SVM}_{R+}(-d_{\text{BP}})$. Symbols \textcircled{x} / \textcircled{y} indicate a statistically significant improvement and \textcircled{x} / \textcircled{y} indicate a statistically significant deterioration in x, y of the 20 iterations when compared with the first and second reference system, respectively (using a Z-test at significance level $\alpha = 0.05$).

Data Set	Reference Systems		Ours
	$\text{SVM}_R(-d_{\text{BP}})$	$\text{SVM}_F(-d_{\text{BP}})$	$\text{SVM}_{R+}(-d_{\text{BP}})$
Mutagenicity	61.8 ± 3.5	69.1	64.3 ± 2.4 $\textcircled{13}$ / $\textcircled{19}$
NCI1	62.0 ± 3.8	68.6	64.5 ± 2.4 $\textcircled{8}$ / $\textcircled{17}$
COX-2	65.2 ± 7.8	71.3	75.6 ± 4.3 $\textcircled{11}$ / $\textcircled{3}$
PTC(MR)	53.1 ± 6.3	54.3	64.6 ± 3.4 $\textcircled{8}$ / $\textcircled{5}$

(including the standard deviation). We observe that the mean classification accuracy of our approach is better than the first reference system on all data sets. On NCI1 our system outperforms the reference system in all iterations (as seen in Figure 6.2). Eight of the 20 improvements are statistically significant⁴. Eight out of 19, 11 out of 20, and 13 out of 19 improvements are statistically significant on the other data sets, respectively.

On Mutagenicity and NCI1 the novel system does not reach the classification accuracy of the second reference system $\text{SVM}_F(-d_{\text{BP}})$ that has access to the full training data. Most of the deteriorations are statistically significant. However, we have to keep in mind that it was not our main goal to improve a classifier that has access to the original training set, but to show that our approach is able to substantially improve a system that has access to a small data set only. From this point of view, it is remarkable that for the other two data sets, our novel approach outperforms the second reference system on average. We observe a substantial improvement of about 5 and 10 percentage points on COX-2 and PTC(MR), respectively.

In Table 6.2 it is also visible that the novel system becomes more robust, as the standard deviation is smaller for each data set for $\text{SVM}_{R+}(-d_{\text{BP}})$ compared to $\text{SVM}_R(-d_{\text{BP}})$.

⁴The statistical significance is computed via Z-test using a significance level of $\alpha = 0.05$.

6.4.2 Graph Augmentation for Neural Networks

The overall aim of the next experimental evaluation (as initially described in [41]) is to verify whether augmented sets help to make neural network based classifiers more robust. In order to answer this question, we evaluate the three GNN architectures, described in Section 6.2, with and without data augmentation. That is, the reference models are trained on the full original training sets (denoted by GCN_F , GIN_F , and $GraphSAGE_F$), whereas our novel models are trained with matching-graphs added to the training sets (denoted as GCN_F+ , GIN_F+ , and $GraphSAGE_F+$).

In order to counteract uncontrolled randomness of neural network initializations, each experiment is repeated five times and the average accuracy is finally reported (we use the same seeds for both the reference approach and the augmented approach).

The evaluation is performed on six data sets (namely Mutagenicity, NCI1, COX-2, PTC(MR), Cuneiform and Synthie) as described in Chapter 3.

6.4.2.1 Validation of Metaparameters

For each iteration of the experiment, the corresponding data set is split into a random training, validation, and test set, with a 3:1:1 split. As we primarily aim at comparing three different network architectures, once with the default training set and once with an augmented training set, we do not separately tune the hyper parameters. Instead, we use the parametrization proposed in [249]. The optimizer used is Adam with a learning rate of 0.001, and we use the Cross Entropy as loss function for all three models.

All Models are trained for 200 epochs (except for *Mutagenicity* and *NCI1*, where we train for 50 epochs only, due to computational limitations arising from the large number of graphs in these data sets). The models that perform the best on the validation sets are finally applied to the test sets. In this experiment, for each matching-graph that is created, the probability p_1 (Algorithm 5, Line 5) is randomly chosen in the interval $[0.1, 0.9]$.

6.4.2.2 Test Results and Discussion

In Table 6.3 we compare the mean classification accuracies of the three reference models with the augmented models using matching-graphs (obtained in five iterations).

Overall we observe that the augmentation process generally works well

Table 6.3: Classification accuracies of three models (GCN_F , GIN_F and $GraphSAGE_F$), compared to the same three models with augmented training sets (GCN_F+ , GIN_F+ and $GraphSAGE_F+$). Symbols \textcircled{x} or $\textcircled{\times}$ indicate a statistically significant improvement or deterioration in x of the five iterations when compared with the respective reference system (using a Z-test at significance level $\alpha = 0.05$).

Data Set	GCN_F	GCN_F+	GIN_F	GIN_F+	$GraphSAGE_F$	$GraphSAGE_F+$
Mutagenicity	79.1 ± 0.7	81.7 ± 1.1 $\textcircled{3}$	80.2 ± 0.9	81.2 ± 0.3 $\textcircled{1}$	77.6 ± 1.1	77.6 ± 0.7
NCI1	68.1 ± 1.8	71.8 ± 0.7 $\textcircled{5}$	74.0 ± 1.7	76.3 ± 0.8 $\textcircled{3}$	72.8 ± 0.4	73.1 ± 1.4 $\textcircled{1}$
COX-2	68.1 ± 4.0	70.9 ± 4.0 $\textcircled{1}$	73.4 ± 6.0	69.8 ± 1.2 $\textcircled{\times}$	68.8 ± 3.0	74.3 ± 3.0 $\textcircled{3}$
PTC(MR)	58.9 ± 7.1	64.0 ± 2.6 $\textcircled{1}$	62.0 ± 6.4	65.1 ± 1.6 $\textcircled{1}$	64.7 ± 4.5	62.0 ± 5.5
Cuneiform	62.7 ± 3.4	85.6 ± 1.3 $\textcircled{5}$	74.0 ± 4.1	82.9 ± 2.3 $\textcircled{4}$	49.3 ± 6.4	69.3 ± 5.3 $\textcircled{5}$
Synthie	73.9 ± 5.3	99.8 ± 0.5 $\textcircled{5}$	75.1 ± 6.1	95.9 ± 1.9 $\textcircled{5}$	48.0 ± 6.9	68.3 ± 2.3 $\textcircled{5}$

for all three GNN architectures. Using the simple GNN, the augmented approach outperforms the corresponding reference system on all data sets and all iterations. In total 20 out of the 30 improvements are statistically significant⁵. On the two data sets *Cuneiform* and *Synthie*, the improvements observed are quite impressive, with an increase of ≈ 23 and ≈ 26 percentage points, respectively. Using the augmented training sets in conjunction with the GIN_F model, we obtain a higher mean accuracy compared to the reference system on five out of six data sets.

In total the augmented model GIN_F+ statistically significantly outperforms the reference system GIN_F in 14 out of 30 iterations. Using $GraphSAGE_F$ trained on the augmented sets, we outperform the reference system on five out of six data sets according to the mean accuracy. While on the *Mutagenicity* and *PTC(MR)* data sets no statistically significant improvement can be observed, for *Cuneiform* and *Synthie* our fitted models work particularly well, with 5 statistically significant improvements. Finally, it can also be seen that the standard deviation of the accuracies for each data set is almost always smaller for the augmented approach. Hence, we can conclude that our system becomes in general more robust and stable when compared with the reference system.

In Figure 6.3 we show – as an example on the *Cuneiform* data set – the reference accuracies as well as the accuracies of our approach as bar charts for all five iterations (in light and dark gray, respectively). The iterations

⁵The statistical significance is computed via Z-test using a significance level of $\alpha = 0.05$.

are ordered from the worst to the best performing reference accuracy. We can clearly see that our approach outperforms the reference systems for all iterations on all models.

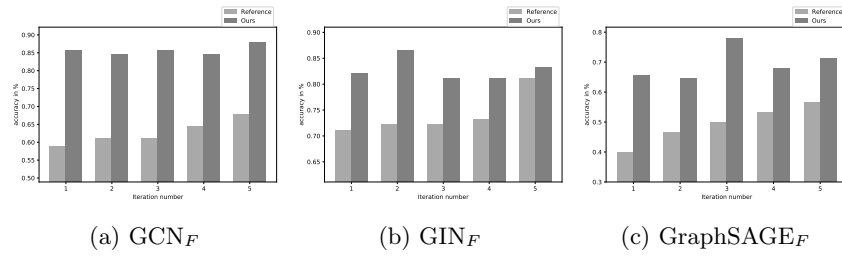


Fig. 6.3: Classification accuracies of the five iterations on the *Cuneiform* data set using three different models (light bars) compared to our system that is based on the same models, but augments the training set with matching-graphs (dark bars).

Regarding the results in Table 6.3, we also observe that the standard deviation of the accuracies is almost always smaller for the augmented approach (on all data sets). Hence, we can conclude that our system becomes in general more robust and stable when compared with the reference system.

Overall, we observe that the augmentation leads to the least improvements on the *GIN_F* model, followed by *GraphSAGE_F* and finally *GCN_F* with the largest improvements. The differences of the accuracies on the *GCN_F* model is the most striking, suggesting that the augmentation approach with matching-graphs helps to bridge the gap when no sophisticated network architecture is available or applicable.

6.5 Building an Ensemble with Matching-Graphs

After providing evidence of the graph augmentation capabilities of the matching-graphs in the previous section, we now explore the use of matching-graphs to create an ensemble classifier. Ensemble methods aim at combining several individual classifiers into one system. That is, an ensemble weighs the opinions of its individual members and combines their results to get the final decision [245]. Various ensemble methods have been proposed in the literature (e.g. Boosting [250] or Bagging [251]).

In the present Section, we propose to use matching-graphs in order to

build a bagging ensemble for graph classification. Thereby, the ensemble consists of multiple GNN classifiers that are trained on random subsets of the training data. The main contribution is to substantially increase the diversity of the bagging ensemble by means of matching-graphs. It is accepted that ensemble learning methods perform the best when a large diversity of the individual classifiers is given [245].

To this end, we propose to further optimize the augmentation method presented in Section 6.3 to generate even more diverse matching-graphs. These novel matching-graphs provide a natural way to create realistic, diverse and relevant graphs of a specific class. It is our main hypothesis that the large amount of possible matching-graphs in conjunction with a bagging procedure ensures the diversity of the individual classifiers and finally allows to build a robust ensemble.

First, in Section 6.5.1, we propose a novel version of the matching-graph that allows for even more diverse graphs, by allowing for the insertion of additional nodes. Second, in Section 6.5.2, we explain the process of using the matching-graphs in order to build a bagging ensemble. Finally, in Section 6.5.3, we research whether or not these improved matching-graphs can be used to build a robust and diverse bagging ensemble.

6.5.1 Matching-Graphs for Ensemble Learning

In Section 6.3, we proposed an adapted version of a matching-graph that represents a mixed version of both original graphs, without being just a sub-graph. However, this definition is still not optimal for the present purposes, since the resulting matching-graphs are always smaller than, or equal to, the original graphs. Hence, we now propose a further altered definition for matching-graphs more suited for the present context of increasing ensemble diversity.

Given a pair of graphs $g = (V, E, \mu, \nu)$ and $g' = (V', E', \mu', \nu')$ we follow the process described in Section 6.3 to build two matching-graphs $m_{g \times g'}$ and $m_{g' \times g}$ based on the partial edit path $\tau(g, g') = \{e_{(1)}, \dots, e_{(t)}\} \subseteq \lambda(g, g')$ with $t = \lfloor p_1 \cdot s \rfloor$ edit operations, where $t < s$ is the amount of randomly selected edit operations from $\lambda(g, g')$ based on a certain probability $p_1 \in [0, 1]$. Next, each edit operation $e_i \in \tau(g, g')$ is applied on graphs $m_{g \times g'}$ and $m_{g' \times g}$ according to the following three rules.

Case 1. e_i is a substitution ($v \rightarrow v'$): The labels of the matching nodes $v \in V$ and $v' \in V'$ are exchanged in both $m_{g \times g'}$ and $m_{g' \times g}$. Note that

this operation shows no effect, if the labels of the involved nodes are identical (i.e. $\mu(v) = \mu'(v')$).

Case 2. e_i is a deletion ($v \rightarrow \varepsilon$): Node $v \in V$

- is deleted in $m_{g \times g'}$.
- is inserted in $m_{g' \times g}$.

As we only execute parts of the edit path, it is possible that the adjacent nodes of v are not yet processed, which means that we do not know the edge structure of v in $m_{g' \times g}$. In this case, we perform a look-ahead to include edges from v to the corresponding nodes in $m_{g' \times g}$. Formally, for all node substitutions $(u \rightarrow v') \in \lambda(g, g')$, where node $u \in V$ is adjacent to node $v \in V$, we insert an edge between the inserted node v and node v' in $m_{g' \times g}$.

Case 3. e_i is an insertion ($\varepsilon \rightarrow v'$): Node $v' \in V'$

- is deleted in $m_{g' \times g}$.
- is inserted in $m_{g \times g'}$ (using a similar look-ahead technique as defined for Case 2).

The basic rationale to apply these rules is that we aim at creating matching-graphs that are indeed related to the underlying graphs, but also substantially differ to them in significant ways. This is achieved by allowing both insertions of nodes and swapping of node labels. Note here the difference to the initial rules described in Section 6.3 where no insertions are allowed.

Clearly, if p_1 is set to 1.0, $\tau(g, g')$ is equal to $\lambda(g, g')$, and thus all edit operations from the complete edit path are executed during the matching-graph creation. In this case, $m_{g' \times g}$ would be equal to the source graph g and $m_{g \times g'}$ would be equal to the target graph g' . For probabilities $p_1 < 1$, however, we obtain matching-graphs that are more diverse and particularly different from simple subgraphs (due to relabelled nodes and potential insertions). That is, when all edit operations of $\tau(g, g')$ are applied, both matching-graphs represent somehow intermediate graphs between g and g' .

Example 6.2. Figure 6.4 shows a visual example of an edit path $\lambda(g, g')$ between two graphs g and g' and two possible resulting matching-graphs $m_{g \times g'}$ and $m_{g' \times g}$. Both matching-graphs are created with the partial edit path that consists of $t = 3$ edit operations. In this example, it is clearly visible that neither $m_{g \times g'}$ nor $m_{g' \times g}$ is a subgraph of g or g' , respectively. Furthermore, the effects of the look-ahead technique is visible. More specifically, between the inserted node $b \in V'$ and node $3 \in V$ an edge is inserted,

even though the substitution ($3 \rightarrow c$) is not yet carried out.

Note that the proposed process can still lead to isolated nodes (despite look-ahead technique), that are deleted in the final matching-graph.

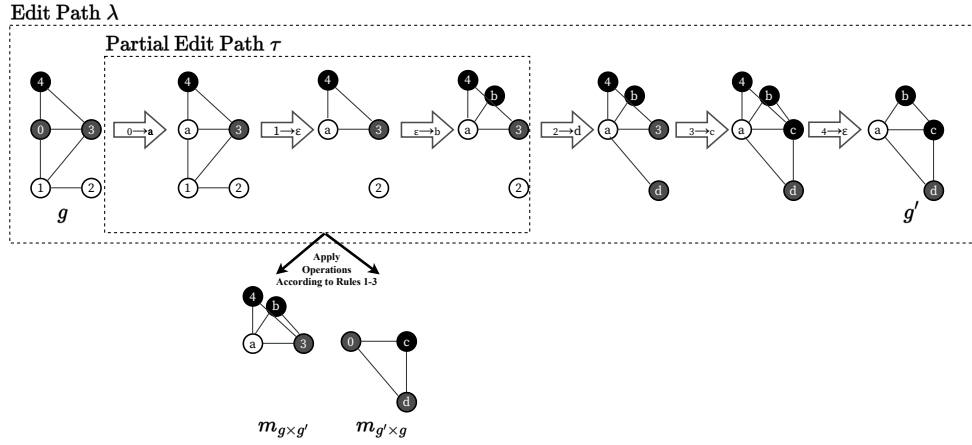


Fig. 6.4: An example of a complete edit path $\lambda(g, g')$, a partial edit path τ , and the resulting matching-graphs $m_{g \times g'}$ and $m_{g' \times g}$.

6.5.2 Building a Bagging Ensemble Using Matching-Graphs

Using the process of creating matching-graphs for any pair of graphs, we can augment a given training set with virtually any number of additional graphs. In order to do this, we conduct the basic steps formalized in Algorithm 5. The algorithm takes k sets of training graphs $G_{\omega_1}, \dots, G_{\omega_k}$ stemming from k different classes $\omega_1, \dots, \omega_k$, and builds two matching-graphs $m_{g \times g'}$ and $m_{g' \times g}$ for each possible graph pair $g, g' \in G_{\omega_i} \times G_{\omega_i}$. Note that for this version of the matching-graph, the probability $p_1 \in [0.1, 0.9]$ used for the creation of the matching-graphs is randomly defined for each pair of graphs g, g' (Algorithm 5, Line 5). Assuming n training graphs per class ω_i this algorithm results in $n(n-1)$ matching-graphs, which are directly used to augment the corresponding training set G_{ω_i} . Hence, rather than n graphs, we now have access to $n(n-1) + n = n^2$ graphs per class ω_i .

Based on the augmented sets, a bagging ensemble $\mathcal{E} = \{c_1, \dots, c_m\}$ with

m classifiers can now be built. Each classifier $c_i \in \mathcal{E}$ is trained only on a subset of all training graphs. To this end, each classifier c_i of the ensemble \mathcal{E} is trained on $\lfloor p_2 \times n^2 \rfloor$ randomly selected graphs from G_{ω_i} , where $p_2 \in [0, 1]$ is a predefined probability and n^2 is the number of graphs available in G_{ω_i} (i.e., we assume that G_{ω_i} is augmented to size $|G_{\omega_i}| = n^2$).

As base classifiers $c_i \in \mathcal{E}$, we use GNNs, which are – due to their inherent randomness – viable ensemble members. For this experiment we employ a model based on the GCN architecture [31], as described in Section 6.2. The individual model trained on the full training set is denoted as GCN_F . For the final graph classification, we add a dropout layer and feed the graph embedding into a fully connected layer. The outputs of the individual classifiers are then aggregated into one single decision by means of majority voting.

6.5.3 Experimental evaluation

This experimental evaluation is conducted on five data sets (namely NCI1, COX-2, PTC(MR), Cuneiform and Synthie) described in Chapter 3. Each data set is split into a training and test set according to a 4:1 split.

The novel ensemble (denoted as GCN-e+) uses the augmented training set and is built as described in Section 6.5.2. We set p_2 to 0.3 and we limit the amount of selected graphs to 100'000 per class. For each ensemble we create 100 classifiers, which are trained for 200 epochs (except for the *NCI1* data set, where we build 50 classifiers, trained for 50 epochs only, due to computational problems arising from the large number of graphs in this data set).

For all base classifiers (viz. GCNs) we use the Adam optimizer with an initial learning rate of 0.01, together with a CosineAnnealingLR scheduler. Furthermore, we use the Cross Entropy loss function. The batch size is set to 64. For the implementation of the ensemble we use the ensemble-pytorch library⁶ which we adapted to seamlessly work with PyTorch Geometric [252]⁷.

Figure 6.5 shows by means of box-plots the training accuracies of all individual classifiers available in the ensembles for all data sets. The diamonds above and below the boxes mark the 10% best and worst classifiers w.r.t. the accuracy. The training accuracy of the final ensemble is marked with a red cross. We observe that the diversity of the classifiers is the

⁶<https://ensemble-pytorch.readthedocs.io>

⁷<https://pytorch-geometric.readthedocs.io>

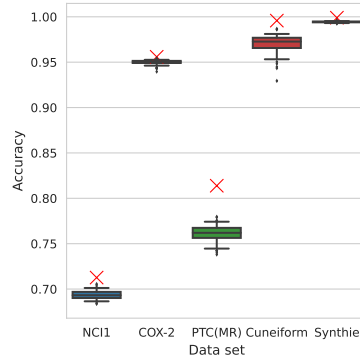


Fig. 6.5: Training accuracies of all individual classifiers of the ensemble for all data sets shown with a box-plot. The training accuracy of the ensemble is marked with a red cross.

largest for NCI1, PTC(MR), and Cuneiform. It is also clearly visible that for all data sets the training accuracy of the complete ensemble is better than the accuracy of the best individual member. This is already a clear indication for the usefulness of the defined ensemble.

6.5.3.1 Reference Systems

The overall aim of the present experimental evaluation is to answer the question, whether or not matching-graphs can be beneficially employed to build robust classifier ensembles. In order to answer this research question, we use three reference systems for comparisons with our novel approach GCN-e+.

- *Reference system 1* (denoted by GCN_F): This reference system is trained on the full training set to obtain a baseline for the classification accuracy. In order to counteract uncontrolled randomness during initialization, each experiment that uses this reference system is repeated five times and the average accuracy is finally reported.

We also perform an ablation study in order to empirically verify that the superiority of the proposed method is indeed based on the matching-graphs. To this end, we compare our novel ensemble GCN-e+ with two additional reference systems.

- *Reference system 2* (denoted by GCN-e): This reference system is virtually the same as our novel ensemble but has only access to the original training data without matching-graphs.
- *Reference system 3* (denoted by SINGL-e+): This reference system refers to the best individual classifier of the novel augmented ensemble.

A comparison with reference system 2 allows us to better assess whether the matching-graphs, or the ensemble by itself, is the important element of the whole process. A comparison with reference system 3 allows us to assess whether the ensemble outperforms the randomly generated members of the system – in other words, whether the ensemble actually also makes a difference.

6.5.3.2 Test Results and Discussion

In Table 6.4 we compare the novel ensemble GCN-e+ with the first reference system (a single GCN trained on the full training set). Remember that we run the GCN reference system five times to counteract randomness during initialization. This is why we report here the mean accuracies (\pm standard deviation).

We observe that GCN-e+ outperforms the reference system GCN in 18 out of 25 cases with statistical significance⁸. On the NCI1 data set, even though we observe an improvement in all five iterations, only four of them are statistically significant. On the COX-2 data set we get an improvement in four out of five iterations (two of them are actually statistically significant). On the PTC(MR) data set, we outperform the reference system in each iteration, however only two of the improvements are statistically significant. On Cuneiform and Synthie all of the improvements are statistically significant (10 out of 10 cases).

Next, in Table 6.5 we compare the novel ensemble with the other two reference systems (for the sake of an ablation study). First, we observe that the best single classifier of each ensemble (reference system 3) outperforms the baseline ensemble (reference system 2) on all data sets. This is a clear indication of the usefulness of the matching-graphs. Even more important, the proposed ensemble GCN-e+ outperforms the second reference ensemble GCN-e on all data sets. These improvements are statistically significant on four out of five data sets. This is a strong indication that the matching-graphs are the important factor in improving the classification accuracy,

⁸The statistical significance is computed via Z-test at significance level $\alpha = 0.05$.

Table 6.4: Average classification accuracy of reference system 1 (GCN_F) compared to our novel ensemble (GCN-e+). Symbol \textcircled{x} indicates a statistically significant improvement in x out of the five comparisons (using a Z-test at significance level $\alpha = 0.05$). Marked in bold is the best accuracy per data set.

Data set	Ref. System 1	Ours
	GCN_F	GCN-e+
NCI1	70.5 ± 1.0	74.0 $\textcircled{4}$
COX-2	70.4 ± 11.1	78.7 $\textcircled{2}$
PTC(MR)	62.3 ± 4.5	68.6 $\textcircled{2}$
Cuneiform	40.3 ± 20.5	96.7 $\textcircled{5}$
Synthie	76.8 ± 7.5	97.5 $\textcircled{5}$

Table 6.5: Classification accuracy of the ensemble without matching-graphs GCN-e , the individually best classifier SINGL-e+ and our ensemble with matching-graphs GCN-e+ . Symbols \circ/\circ indicate a statistically significant improvement over the second/third reference system using a Z-test at significance level $\alpha = 0.05$). Marked in bold is the best accuracy per data set.

Data set	Ref. System 2	Ref. System 3	Ours
	GCN-e	SINGL-e+	GCN-e+
NCI1	70.7	74.8	74.0 $\circ/-$
COX-2	73.4	77.6	78.5 $-/-$
PTC(MR)	61.4	62.9	68.6 \circ/\circ
Cuneiform	68.3	93.3	96.7 $\circ/-$
Synthie	64.2	93.8	97.5 $\circ/-$

rather than primarily the ensemble itself. However, when comparing our ensemble with the third reference system, it is also obvious that the ensemble still makes an important contribution – only on NCI1 is the best individual classifier better than the ensemble.

6.6 Conclusion

Access to sufficient training data is important and crucial but unfortunately not always given – so in real world applications one sometimes has to manage with very little labeled data. One possible solution to this problem is to systematically augment the data sets with artificial training data. The

process of augmentation is not only interesting in situations where little training data is available, but particularly for systems that crucially depend on large training sets (like, for instance, neural network based classifiers).

In case of statistical data representations, quite an amount of methods is available for the task of data augmentation. Due to the complex nature of graphs, however, research on graph augmentation lags behind that of its statistical counterparts. In this work, we focus on augmenting (small) training sets of graphs. The goal of this augmentation process is to make graph-based pattern recognition more robust and ultimately improve the downstream training of classification algorithms. The novelty of the present approach is that we use so-called matching-graphs for the augmentation process.

Matching-graphs, which can be pre-computed by means of (sub-optimal) graph edit distance computations, formalize the matching between two graphs. Thereby, they actually define a novel graph that encodes an intermediate representation of two given graphs. In the present approach, we systematically produce matching-graphs for each pair of training graphs and are thus able to substantially increase even very small sets of training graphs.

In this chapter we evaluate the novel procedure in two specific situations. First, as described in Section 6.3 and 6.4, we research the effects of the novel augmentation process. In order to do this, we start with the evaluation of very small graph sets. By means of an experimental evaluation on artificially reduced graph data sets, we empirically confirm that our novel approach is able to significantly outperform a classifier that has access to the small training set only. Moreover, in some particular cases we can even report that our novel approach is able to outperform a system that has access to the original set of training graphs. Next, we evaluate the novel approach of augmenting the training data in an experiment with three different GNN models. In this scenario, we assume the full data (i.e., we do not artificially reduce the training data). This experiment is particularly interesting because it is known that GNNs can be sensitive to the size of the training set. We empirically confirm that our novel approach is able to improve all three GNN architectures in general. The vast majority of the observed improvements is statistically significant.

In the second scenario, as described in Section 6.5, we focus on building an ensemble of classifiers via bagging. One of the main problems in building an ensemble is that large and diverse data sets are needed. We propose to use the augmentation technique based on matching-graphs in order to build

a robust and diverse ensemble. By means of an experimental evaluation, we empirically confirm that our novel approach significantly outperforms three related reference systems, viz. a single GNN classifier, a bagging ensemble trained on the original training set, as well as the best individual classifier stemming from the novel ensemble. Hence, we conclude that matching-graphs provide a versatile way to generate large sets of additional graphs in order to build a diverse and robust ensemble.

For future work we see several rewarding avenues that can be pursued. First, one can explore whether other heuristics can be used in order to make the matching-graphs even more diverse. Second, it would be interesting to see if the matching-graphs can be used in conjunction with other GNNs (e.g. Triplet loss networks or others). Third, the augmentation capabilities for node classification problems, as well as for regression problems could be investigated. Furthermore in terms of ensembles, other ensemble modalities (rather than bagging), as well as other aggregation techniques to combine the results could be explored (rather than majority voting).

Conclusion and Future Work

7

As conclusion draws near, the path becomes clear. Through the journey of graphs, we did steer, ending this thesis with cheer.

ChatGPT

Pattern recognition is an important field of research. It can be divided into statistical and structural pattern recognition. Statistical pattern recognition relies on data that is in the form of feature vectors, while structural pattern recognition can make use of more versatile representations, such as graphs, for instance. A graph is a data structure that consists of nodes that are, in turn, connected by edges. The nodes and edges can contain additional characteristics called labels. Compared to feature vectors, graphs provide a powerful alternative that is able to also capture the structure of an object, which is actually crucial in diverse applications. This is the case, for example, when molecular structures or social networks have to be formally represented.

It is often necessary to compare pairs of graphs in order to recognize an underlying pattern. This process is commonly referred to as graph matching. More specifically, graph matching is the process of evaluating the dissimilarity between two graphs. Traditionally, this is done by finding a correspondence between similar substructures of the two graphs that are matched. The matching process usually yields a dissimilarity or similarity score that indicates the proximity of the two graphs.

Roughly speaking, there are two types of graph matching available. Namely, exact and inexact graph matching. Exact graph matching algorithms require the graphs being compared to have strict correspondences

between them or their subparts, in order for a matching to be successful. However, since data in the real world is usually error-prone and far from perfect, the resulting graphs are often imprecise and contain noise. This makes exact matching methods infeasible in real-world applications as two real-world graphs representing the same class of objects cannot be expected to be completely identical in a large part of their structure.

Inexact (or error-tolerant) graph matching is a versatile alternative to exact graph matching, as these algorithms are robust to noise and offer an inherent error tolerance. In recent years, a large number of error-tolerant graph matching methods have been proposed. Some of these methods are, however, only applicable to specific types of graphs. Graph edit distance is one of the most commonly used and well-established inexact graph matching methods available. The main advantage of graph edit distance is its flexibility, as it is able to process all types of graphs. It follows the idea of transforming one graph into another, given a set of edit operations. Each of these operations is assigned a certain cost, and the cheapest possible way to transform one graph into another is then called the edit distance. The greater this distance, the more dissimilar the graphs are. Furthermore, graph edit distance provides more than merely a dissimilarity score, as it also provides us with a so-called edit path. An edit path contains the set of edit operations used to transform one graph into another graph.

The edit path provides crucial information about the similarity of two graphs. The aim of this thesis is to extract and encode this particular information to generate a meta-graph, called the matching-graph. This graph can be used for reasoning about graphs, classifying graphs, and generating new graphs. The basic procedure of the present thesis is as follows. First, it computes the graph edit distance between several pairs of graphs. Based on the resulting matchings, the matching-graphs are generated, formalizing the stable parts of the underlying graph pairs.

This thesis evaluates the matching-graphs in three different scenarios. In the first scenario (see Chapter 4), the concept is thoroughly evaluated for classification using two different strategies. The first strategy is to use the matching-graphs in a distance-based classifier. In the second strategy, the matching-graphs are used to build a graph embedding which is then used for classification. The evaluation is carried out through various experiments on several graph data sets. The results show that the classification systems based on the matching-graphs significantly outperform the reference systems and are able to match or outperform several state-of-the-art classifiers.

In the second scenario (see Chapter 5), the matching-graphs are evaluated regarding their quality and compared in detail to the maximum common subgraph. In a qualitative experimental evaluation, we show that the matching-graphs are able to reveal significant and frequent substructures of a given class and have a high potential for detecting novel and relevant substructures in sets of graphs. Furthermore, a special version of the matching-graph is constructed and compared to the maximum common subgraph. It is shown that this version can be used to efficiently approximate the maximum common subgraph. In an experimental evaluation on several graph data sets we empirically confirm the benefit of this approximation method.

In the last scenario (see Chapter 6), the matching-graphs are used for the purpose of graph augmentation. This is because in the graph domain there is often a lack of data. Furthermore, with the advent of graph neural networks, the availability of large data sets becomes even more important. To perform the augmentation task, we propose an adapted version of the matching-graphs, which is able to generate different but still similar graphs, based on graph pairs. This process is evaluated in three different experiments, where we first apply the method to very small graph data sets. Then the method is applied to full graph data sets to obtain large and diverse data sets for use with graph neural networks. Finally, the augmentation method is used to build a robust bagging classification ensemble. In all three experiments, we can empirically confirm that the matching-graphs can indeed be used to augment graph data sets to produce more robust classifiers.

In order to summarize all of the evaluations conducted, Table 7.1 gives a complete overview of all classification experiments performed throughout this thesis. The dash (–) indicates that the experiment has not been performed on the corresponding data set. The experiments termed *Distance-Based Classifiers* and *Embedding-Based Classifiers* stem from Chapter 4. The *MCS Dissimilarity Classifiers* originate from Chapter 5. Finally, the *Augmentation Classifiers* as well as the *Ensemble Classifiers* are presented in Chapter 6. As one of the main goals of matching-graphs is to improve (any) existing classification approach by using matching-graphs in addition to an original graph, the following table only summarizes our approaches with the most comparable reference system, based on the same classifier as a basis (e.g. k -NN, SVM or a specific GNN architecture).

In the 76 experiments performed, our approach outperforms the most comparable baseline 66 times in total. For the distance-based classifiers, the improvement is statistically significant in 9 out of 14 cases and for the

embedding-based classifiers in 6 out of 14 cases. The classifiers based on the MCS dissimilarity metrics are able to outperform the reference system with statistical significance in 6 out of 21 cases. The augmentation approaches are statistically significant in a total of 89 out of 170 iterations and the ensemble classifiers in 18 out of 25 iterations. In terms of absolute accuracy, we can see that there is no clear winner among all systems. However, it is very evident that the matching-graphs almost always help to improve a given baseline classifier.

Overall, the novel concept of a matching-graph turns out to be a useful tool in several graph-based pattern recognition scenarios. Yet, there are several things that could be explored and/or optimized in future work. First, it could be explored how different graph edit distance approximation algorithms affect the quality of the matching-graphs. Furthermore, due to the great flexibility of the matching-graphs, the optimization possibilities are almost endless. For example, it could be explored how different cost functions affect the quality of the matching-graphs. Each of the experiments can also be explored further. For example, matching-graphs could be used in conjunction with different modalities of graph neural networks, such as triplet loss networks, or with different ensemble methods besides bagging. Furthermore, matching-graphs could potentially be applied to the task of node classification, or regression problems, as well as to tasks like dictionary learning and drug discovery.

Conclusion and Future Work

125

Table 7.1: Summary table of all classification accuracies of all experiments performed in Chapters 4, 5 and 6. A dash (—) as entry indicates that the experiment has not been performed on the corresponding data set. Values that are *underlined* indicate, that the approach performs equal or better than the corresponding reference approach. The best result per data set is shown in bold face. Symbol \circ indicates a statistically significant improvement and \bullet indicates a statistically significant deterioration over the corresponding reference system. If multiple iterations have been performed, symbols \otimes indicate a statistically significant improvement and \otimes indicate a statistically significant deterioration in x of all iterations when compared with the corresponding reference system. Significance is calculated using a Z-test at significance level $\alpha = 0.05$.

Data Set	Distance-Based Classifiers		Embedding-Based Classifiers		MCS Dissimilarity Classifiers				Augmentation Classifiers			Ensemble Classifiers	
	k -NN(d_{x1})	SVM($-d_{x1}$)	k -NN($s_{d(0)}$)	SVM($\kappa_{s(d)}$)	k -NN($d_{(cat)}$)	k -NN($d_{(cat)}$)	k -NN($d_{(cat)}$)	k -NN($d_{(cat)}$)	SVM R + ($-d_{BP}$)	GCN F +	GIN F +	GraphSAGE F +	GCN-e+
AIDS	<u>99.2\circ</u>	<u>99.2\circ</u>	<u>99.2\circ</u>	<u>99.6\circ</u>	<u>100.0\circ</u>	<u>100.0\circ</u>	<u>100.0\circ</u>	<u>100.0\circ</u>	—	—	—	—	—
Mutagenicity	73.0 \circ	70.4 \circ	74.8 \circ	76.3 \circ	72.4 \circ	70.2 \circ	72.7 \circ	72.7 \circ	64.3 \pm 2.4 \otimes	81.7 \pm 1.1 \otimes	81.2 \pm 0.3 \otimes	77.6 \pm 0.7	—
NCI	<u>77.4\circ</u>	68.8 \circ	70.1 \circ	70.7 \circ	67.3 \circ	60.9 \circ	60.9 \circ	60.9 \circ	64.5 \pm 2.4 \otimes	71.8 \pm 0.7 \otimes	76.3 \pm 0.8 \otimes	73.1 \pm 1.4 \otimes	74.0 \otimes
COX-2	81.7 \circ	81.0 \circ	80.0 \circ	78.5 \circ	80.7 \circ	<u>82.8\circ</u>	70.0 \circ	70.0 \circ	<u>75.6 \pm 4.3 \otimes</u>	70.9 \pm 4.0 \otimes	69.8 \pm 1.2 \bullet	74.3 \pm 3.0 \otimes	75.7 \otimes
PTC(MR)	65.7 \circ	67.1 \circ	58.6 \circ	61.4 \circ	55.7 \circ	48.6 \circ	48.6 \circ	48.6 \circ	<u>64.6 \pm 3.4 \otimes</u>	<u>64.0 \pm 2.6 \otimes</u>	<u>65.1 \pm 1.6 \otimes</u>	62.0 \pm 5.5	<u>68.6 \otimes</u>
PROTEINS	—	—	—	—	71.1 \circ	70.0 \circ	<u>80.4\circ</u>	—	—	—	—	—	—
ENZYMES	—	—	—	—	<u>22.3\circ</u>	21.7 \circ	20.0 \circ	—	—	—	—	—	—
Letter	91.7 \circ	92.5 \circ	90.8 \circ	<u>93.2\circ</u>	—	—	—	—	—	—	—	—	—
IMDB	<u>68.0\circ</u>	<u>66.0\circ</u>	59.0 \circ	<u>68.5\circ</u>	—	—	—	—	—	—	—	—	—
Cineform	—	—	—	—	—	—	—	—	—	85.6 \pm 1.3 \otimes	82.9 \pm 2.3 \otimes	69.3 \pm 5.3 \otimes	<u>96.7 \otimes</u>
Synthetic	—	—	—	—	—	—	—	—	—	<u>99.8 \pm 0.5 \otimes</u>	<u>95.9 \pm 1.9 \otimes</u>	<u>68.3 \pm 2.3 \otimes</u>	<u>97.5 \otimes</u>

Appendix A

TSNE Visualizations

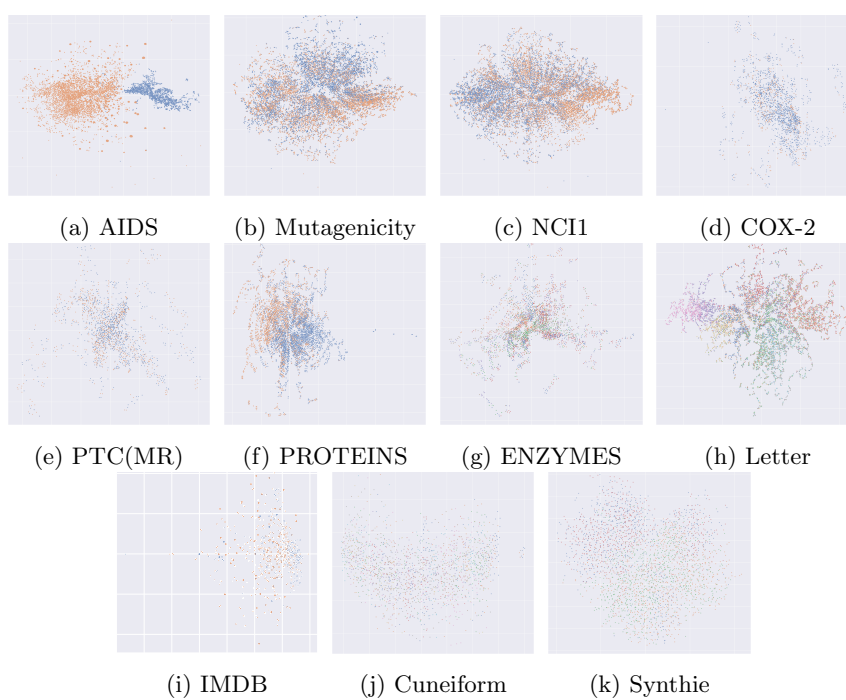


Fig. A.1: T-SNE visualization of all data sets. The individual colors represent the individual classes of the data set.

Bibliography

- [1] Siyue Xie, Haifeng Hu, and Yongbo Wu. Deep multi-path convolutional neural network joint with salient region attention for facial expression recognition. *Pattern Recognit.*, 92:177–191, 2019.
- [2] Kin Gwn Lore, Adedotun Akintayo, and Soumik Sarkar. Llnet: A deep autoencoder approach to natural low-light image enhancement. *Pattern Recognit.*, 61:650–662, 2017.
- [3] Chongyi Li, Jichang Guo, Fatih Porikli, and Yanwei Pang. Lightennet: A convolutional neural network for weakly illuminated image enhancement. *Pattern Recognit. Lett.*, 104:15–22, 2018.
- [4] Ya Jing, Junbo Wang, Wei Wang, Liang Wang, and Tieniu Tan. Relational graph neural network for situation recognition. *Pattern Recognit.*, 108:107544, 2020.
- [5] Sana Ullah Khan, Naveed Islam, Zahoor Jan, Ikram Ud Din, and Joel J. P. C. Rodrigues. A novel deep learning based framework for the detection and classification of breast cancer using transfer learning. *Pattern Recognit. Lett.*, 125:1–6, 2019.
- [6] Antonio Candelieri, Bruno G. Galuzzi, Ilaria Giordani, and Francesco Archetti. Vulnerability of public transportation networks against directed attacks and cascading failures. *Public Transp.*, 11(1):27–49, 2019.
- [7] Anffany Chen, Hauke Brand, Tobias Helbig, Tobias Hofmann, Stefan Imhof, Alexander Fritzsche, Tobias Kießling, Alexander Stegmaier, Lavi K. Upreti, Titus Neupert, Tomáš Bzdušek, Martin Greiter, Ronny Thomale, and Igor Boettcher. Hyperbolic matter in electrical circuits with tunable complex phases. *Nature Communications*, 14(1):622, 2023.
- [8] Maryam Khalid and Akane Sano. Exploiting social graph networks for emotion prediction. *Scientific Reports*, 13(1):6069, 2023.
- [9] Paul Maergner, Vinaychandran Pondenkandath, Michele Alberti, Marcus Liwicki, Kaspar Riesen, Rolf Ingold, and Andreas Fischer. Offline signature verification by combining graph edit distance and triplet networks. In Xiao Bai, Edwin R. Hancock, Tin Kam Ho, Richard C. Wilson, Battista Biggio, and Antonio Robles-Kelly, editors, *Structural, Syntactic, and Statistical Pattern Recognition - Joint IAPR International Workshop, S+SSPR 2018*,

- Beijing, China, August 17-19, 2018, *Proceedings*, volume 11004 of *Lecture Notes in Computer Science*, pages 470–480. Springer, 2018.
- [10] Weidong Xie, Wei Li, Shoujia Zhang, Linjie Wang, Jinzhu Yang, and Dazhe Zhao. A novel biomarker selection method combining graph neural network and gene relationships applied to microarray data. *BMC Bioinform.*, 23(1):303, 2022.
 - [11] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *Int. J. Pattern Recognit. Artif. Intell.*, 18(3):265–298, 2004.
 - [12] Pasquale Foggia, Gennaro Percannella, and Mario Vento. Graph matching and learning in pattern recognition in the last 10 years. *Int. J. Pattern Recognit. Artif. Intell.*, 28(1), 2014.
 - [13] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, 2011.
 - [14] Paul Maergner, Vinaychandran Pondenkandath, Michele Alberti, Marcus Liwicki, Kaspar Riesen, Rolf Ingold, and Andreas Fischer. Combining graph edit distance and triplet networks for offline signature verification. *Pattern Recognit. Lett.*, 125:527–533, 2019.
 - [15] Manuel Curado, Francisco Escolano, Miguel Angel Lozano, and Edwin R. Hancock. Early detection of alzheimer’s disease: Detecting asymmetries with a return random walk link predictor. *Entropy*, 22(4):465, 2020.
 - [16] Thomas Gärtner. *Kernels for Structured Data*, volume 72 of *Series in Machine Perception and Artificial Intelligence*. WorldScientific, 2008.
 - [17] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.
 - [18] Shinji Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(5):695–703, 1988.
 - [19] Marco Carcassoni and Edwin R. Hancock. Weighted graph-matching using modal clusters. In Wladyslaw Skarbek, editor, *Computer Analysis of Images and Patterns, 9th International Conference, CAIP 2001 Warsaw, Poland, September 5-7, 2001, Proceedings*, volume 2124 of *Lecture Notes in Computer Science*, pages 142–151. Springer, 2001.
 - [20] Terry Caelli and Serhiy Kosinov. Inexact graph matching using eigen-subspace projection clustering. *Int. J. Pattern Recognit. Artif. Intell.*, 18(3):329–354, 2004.
 - [21] U Kang, Martial Hebert, and Soonyong Park. Fast and scalable approximate spectral graph matching for correspondence problems. *Inf. Sci.*, 220:306–318, 2013.
 - [22] Martin A. Fischler and Robert A. Elschlager. The representation and matching of pictorial structures. *IEEE Trans. Computers*, 22(1):67–92, 1973.
 - [23] Richard C. Wilson and Edwin R. Hancock. Structural matching by discrete relaxation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(6):634–648, 1997.

- [24] Bin Luo and Edwin R. Hancock. Structural graph matching using the EM algorithm and singular value decomposition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(10):1120–1136, 2001.
- [25] Andrew D. J. Cross, Richard C. Wilson, and Edwin R. Hancock. Inexact graph matching using genetic search. *Pattern Recognit.*, 30(6):953–970, 1997.
- [26] Yuan-Kai Wang, Kuo-Chin Fan, and Jorng-Tzong Horng. Genetic-based search for error-correcting graph isomorphism. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 27(4):588–597, 1997.
- [27] K. G. Khoo and Ponnuthurai N. Suganthan. Multiple relational graphs mapping using genetic algorithms. In *Proceedings of the 2001 Congress on Evolutionary Computation, CEC 2001, COEX, Seoul, Korea, May 27-30, 2001*, pages 727–733. IEEE, 2001.
- [28] Nils M. Kriege and Petra Mutzel. Subgraph matching kernels for attributed graphs. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012.
- [29] Thomas Gärtner, Peter A. Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In Bernhard Schölkopf and Manfred K. Warmuth, editors, *Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003, Proceedings*, volume 2777 of *LNCS*, pages 129–143. Springer, 2003.
- [30] Karsten M. Borgwardt and Hans-Peter Kriegl. Shortest-path kernels on graphs. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), 27-30 November 2005, Houston, Texas, USA*, pages 74–81. IEEE Computer Society, 2005.
- [31] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.
- [32] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *CoRR*, abs/1710.10903, 2017.
- [33] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.
- [34] Horst Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognit. Lett.*, 1(4):245–253, 1983.
- [35] Mathias Fuchs and Kaspar Riesen. Matching of matching-graphs - A novel approach for graph classification. In *25th International Conference on Pattern Recognition, ICPR 2020, Virtual Event / Milan, Italy, January 10-15, 2021*, pages 6570–6576. IEEE, 2020.
- [36] Mathias Fuchs and Kaspar Riesen. Iterative creation of matching-graphs –

- finding relevant substructures in graph sets. In *In Proceedings of the 25th Iberoamerican Congress on Pattern Recognition, CIARP25 2021*, 2021.
- [37] Mathias Fuchs and Kaspar Riesen. Graph embedding in vector spaces using matching-graphs. In Nora Reyes, Richard Connor, Nils M. Kriege, Daniyal Kazempour, Ilaria Bartolini, Erich Schubert, and Jian-Jia Chen, editors, *Similarity Search and Applications - 14th International Conference, SISAP 2021, Dortmund, Germany, September 29 - October 1, 2021, Proceedings*, volume 13058 of *Lecture Notes in Computer Science*, pages 352–363. Springer, 2021.
 - [38] Mathias Fuchs and Kaspar Riesen. A novel way to formalize stable graph cores by using matching-graphs. *Pattern Recognit.*, 131:108846, 2022.
 - [39] Mathias Fuchs and Kaspar Riesen. Matching-graphs for approximate maximum common subgraph computation. *Pattern Recognition Letters*, 2023. Currently under review for possible publication.
 - [40] Mathias Fuchs and Kaspar Riesen. Graph augmentation for small training sets using matching-graphs. *ICPRAI - 3rd International Conference on pattern Recognition and Artificial Intelligence*, 2022.
 - [41] Mathias Fuchs and Kaspar Riesen. Graph augmentation for neural networks using matching-graphs. In Neamat El Gayar, Edmondo Trentin, Mirco Ravanelli, and Hazem Abbas, editors, *Artificial Neural Networks in Pattern Recognition - 10th IAPR TC3 Workshop, ANNPR 2022, Dubai, United Arab Emirates, November 24-26, 2022, Proceedings*, volume 13739 of *Lecture Notes in Computer Science*, pages 3–15. Springer, 2022.
 - [42] Mathias Fuchs and Kaspar Riesen. Matching-graphs for building classification ensembles. In *Graph-Based Representations in Pattern Recognition - 13th IAPR-TC-15 International Workshop, GbRPR 2023, Vietri sul Mare, Italy, September 6-8, 2023. Proceedings*, 2023. Currently under review for possible publication.
 - [43] Kaspar Riesen and Horst Bunke. *Graph Classification and Clustering Based on Vector Space Embedding*, volume 77 of *Series in Machine Perception and Artificial Intelligence*. WorldScientific, 2010.
 - [44] Kaspar Riesen. *Structural Pattern Recognition with Graph Edit Distance - Approximation Algorithms and Applications*. Advances in Computer Vision and Pattern Recognition. Springer, 2015.
 - [45] Martin Grohe and Pascal Schweitzer. The graph isomorphism problem. *Commun. ACM*, 63(11):128–134, 2020.
 - [46] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
 - [47] Mario Vento. A long trip in the charming world of graphs for pattern recognition. *Pattern Recognit.*, 48(2):291–301, 2015.
 - [48] Ingo Wegener. *Complexity theory - exploring the limits of efficient algorithms*. Springer, 2005.
 - [49] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. On finding lowest common ancestors in trees. *SIAM J. Comput.*, 5(1):115–132, 1976.
 - [50] Baida Zhang, Yuhua Tang, Junjie Wu, and Shuai Xu. Ld: a polynomial time algorithm for tree isomorphism. *Procedia Engineering*, 15:2015–2020,

2011.

- [51] Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.*, 25(1):42–65, 1982.
- [52] Xiaoyi Jiang and Horst Bunke. Optimal quadratic-time isomorphism of ordered graphs. *Pattern Recognit.*, 32(7):1273–1283, 1999.
- [53] John E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In Robert L. Constable, Robert W. Ritchie, Jack W. Carlyle, and Michael A. Harrison, editors, *Proceedings of the 6th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1974, Seattle, Washington, USA*, pages 172–184. ACM, 1974.
- [54] Charles J. Colbourn. On testing isomorphism of permutation graphs. *Networks*, 11(1):13–21, 1981.
- [55] Peter J. Dickinson, Horst Bunke, Arek Dadej, and Miro Kraetzl. On graphs with unique node labels. In Edwin R. Hancock and Mario Vento, editors, *Graph Based Representations in Pattern Recognition, 4th IAPR International Workshop, GbRPR 2003, York, UK, June 30 - July 2, 2003, Proceedings*, volume 2726 of *Lecture Notes in Computer Science*, pages 13–23. Springer, 2003.
- [56] Peter J. Dickinson, Horst Bunke, Arek Dadej, and Miro Kraetzl. Matching graphs with unique node labels. *Pattern Anal. Appl.*, 7(3):243–254, 2004.
- [57] Jurij Mihelic and Uros Cibej. An experimental evaluation of refinement techniques for the subgraph isomorphism backtracking algorithms. *Open Comput. Sci.*, 11(1):33–42, 2021.
- [58] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(10):1367–1372, 2004.
- [59] Michael Stauffer, Andreas Fischer, and Kaspar Riesen. Keyword spotting in historical handwritten documents based on graph matching. *Pattern Recognit.*, 81:240–253, 2018.
- [60] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society: series B (methodological)*, 39(1):1–22, 1977.
- [61] Melanie Mitchell. *An introduction to genetic algorithms*. MIT Press, 1998.
- [62] Michel Neuhaus and Horst Bunke. *Bridging the Gap between Graph Edit Distance and Kernel Machines*, volume 68 of *Series in Machine Perception and Artificial Intelligence*. WorldScientific, 2007.
- [63] Nils M. Kriege and Christopher Morris. Recent advances in kernel-based graph classification. In Yasemin Altun et al., editor, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18-22, 2017, Proceedings, Part III*, volume 10536 of *LNCS*, pages 388–392. Springer, 2017.
- [64] David Haussler et al. Convolution kernels on discrete structures. Technical report, Citeseer, 1999.
- [65] Chris Watkins. Dynamic alignment kernels. In *Advances in Large Margin Classifiers*, pages 39–50. MIT Press, 1999.
- [66] Karsten M. Borgwardt, Tobias Petri, S. V. N. Vishwanathan, and Hans-

- Peter Kriegel. An efficient sampling scheme for comparison of large graphs. In Paolo Frasconi, Kristian Kersting, and Koji Tsuda, editors, *Mining and Learning with Graphs, MLG 2007, Firenze, Italy, August 1-3, 2007, Proceedings*, 2007.
- [67] Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M. Borgwardt. Efficient graphlet kernels for large graph comparison. In David A. Van Dyk and Max Welling, editors, *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, 2009, Florida, USA, April 16-18, 2009*, volume 5 of *JMLR Proceedings*, pages 488–495. JMLR.org, 2009.
- [68] Ponnuthurai N. Suganthan, Eam Khwang Teoh, and Dinesh P. Mital. Pattern recognition by homomorphic graph matching using hopfield neural networks. *Image Vis. Comput.*, 13(1):45–60, 1995.
- [69] Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Trans. Neural Networks*, 8(3):714–735, 1997.
- [70] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009.
- [71] Wen-Hsiang Tsai and King-Sun Fu. Error-correcting isomorphisms of attributed relational graphs for pattern analysis. *IEEE Trans. Syst. Man Cybern.*, 9(12):757–768, 1979.
- [72] Alberto Sanfeliu and King-Sun Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Trans. Syst. Man Cybern.*, 13(3):353–362, 1983.
- [73] M. A. Eshera and King-Sun Fu. A graph distance measure for image analysis. *IEEE Trans. Syst. Man Cybern.*, 14(3):398–408, 1984.
- [74] Roberto M. Cesar, Endika Bengoetxea, and Isabelle Bloch. Inexact graph matching using stochastic optimization techniques for facial feature recognition. In *16th International Conference on Pattern Recognition, ICPR 2002, Quebec, Canada, August 11-15, 2002*, pages 465–468. IEEE Computer Society, 2002.
- [75] R. Ambauen, Stefan Fischer, and Horst Bunke. Graph edit distance with node splitting and merging, and its application to diatom identification. *GbrPR*, 2726:95–106, 2003.
- [76] Tamal Dey, Herbert Edelsbrunner, Sumanta Guha, and Dmitry Nekhayev. Topology preserving edge contraction. *Publications de l’Institut Mathématique*, 66, 1999.
- [77] Cristina Gomila and Fernand Meyer. Tracking objects by graph matching of image partition sequences. In *Proc. 3rd IAPR-TC15 Workshop Graph-Based Representations in Pattern Recognition*, pages 1–11, 2001.
- [78] Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.
- [79] Albert Solé-Ribalta, Francesc Serratosa, and Alberto Sanfeliu. On the graph edit distance cost: Properties and applications. *Int. J. Pattern Recognit. Artif. Intell.*, 26(5), 2012.

- [80] Vladimir I Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union, 1966.
- [81] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.
- [82] Michel Neuhaus and Horst Bunke. Automatic learning of cost functions for graph edit distance. *Inf. Sci.*, 177(1):239–247, 2007.
- [83] Xavier Cortés and Francesc Serratosa. Learning graph-matching edit-costs based on the optimality of the oracle’s node correspondences. *Pattern Recognit. Lett.*, 56:22–29, 2015.
- [84] Elena Rica, Susana Álvarez, and Francesc Serratosa. On-line learning the graph edit distance costs. *Pattern Recognit. Lett.*, 146:55–62, 2021.
- [85] Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vis. Comput.*, 27(7):950–959, 2009.
- [86] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2):100–107, 1968.
- [87] Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. In Maria De Marsico, Mário A. T. Figueiredo, and Ana L. N. Fred, editors, *ICPRAM 2015 - Proceedings of the International Conference on Pattern Recognition Applications and Methods, Volume 1, Lisbon, Portugal, 10-12 January, 2015*, pages 271–278. SciTePress, 2015.
- [88] David B. Blumenthal, Nicolas Boria, Johann Gamper, Sébastien Bogleux, and Luc Brun. Comparing heuristics for graph edit distance computation. *VLDB J.*, 29(1):419–458, 2020.
- [89] Michel Neuhaus, Kaspar Riesen, and Horst Bunke. Fast suboptimal algorithms for the computation of graph edit distance. In Dit-Yan Yeung, James T. Kwok, Ana L. N. Fred, Fabio Roli, and Dick de Ridder, editors, *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshops, SSPR 2006 and SPR 2006, Hong Kong, China, August 17-19, 2006, Proceedings*, volume 4109 of *Lecture Notes in Computer Science*, pages 163–172. Springer, 2006.
- [90] Kaspar Riesen, Andreas Fischer, and Horst Bunke. Combining bipartite graph matching and beam search for graph edit distance approximation. In Neamat El Gayar, Friedhelm Schwenker, and Cheng Suen, editors, *Artificial Neural Networks in Pattern Recognition - 6th IAPR TC 3 International Workshop, ANNPR 2014, Montreal, QC, Canada, October 6-8, 2014. Proceedings*, volume 8774 of *Lecture Notes in Computer Science*, pages 117–128. Springer, 2014.
- [91] Raj Reddy. Speech understanding systems: A summary of results of the five-year research effort at carnegie mellon university. *Pittsburgh, Pa*, 1977.
- [92] Derek Justice and Alfred O. Hero III. A binary linear programming formulation of the graph edit distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(8):1200–1214, 2006.

- [93] Richard Myers, Richard C. Wilson, and Edwin R. Hancock. Bayesian graph edit distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(6):628–635, 2000.
- [94] Andrea Torsello and Edwin R. Hancock. Computing approximate tree edit distance using relaxation labeling. *Pattern Recognit. Lett.*, 24(8):1089–1097, 2003.
- [95] Martin A. Fischler and Robert A. Elschlager. The representation and matching of pictorial structures. *IEEE Trans. Computers*, 22(1):67–92, 1973.
- [96] Maria C. Boeres, Celso C. Ribeiro, and Isabelle Bloch. A randomized heuristic for scene recognition by graph matching. In Celso C. Ribeiro and Simone L. Martins, editors, *Experimental and Efficient Algorithms, Third International Workshop, WEA 2004, Angra dos Reis, Brazil, May 25-28, 2004, Proceedings*, volume 3059 of *Lecture Notes in Computer Science*, pages 100–113. Springer, 2004.
- [97] Sébastien Sorlin and Christine Solnon. Reactive tabu search for measuring graph similarity. In Luc Brun and Mario Vento, editors, *Graph-Based Representations in Pattern Recognition, 5th IAPR International Workshop, GbRPR 2005, Poitiers, France, April 11-13, 2005, Proceedings*, volume 3434 of *Lecture Notes in Computer Science*, pages 172–182. Springer, 2005.
- [98] Nicolas Boria, David B. Blumenthal, Sébastien Bougleux, and Luc Brun. Improved local search for graph edit distance. *Pattern Recognit. Lett.*, 129:19–25, 2020.
- [99] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *Proc. VLDB Endow.*, 2(1):25–36, 2009.
- [100] Kaspar Riesen, Stefan Fankhauser, and Horst Bunke. Speeding up graph edit distance computation with a bipartite heuristic. In Paolo Frasconi, Kristian Kersting, and Koji Tsuda, editors, *Mining and Learning with Graphs, MLG 2007, Firenze, Italy, August 1-3, 2007, Proceedings*, 2007.
- [101] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.
- [102] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.
- [103] Roy Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, 1987.
- [104] Rainer E. Burkard, Mauro Dell’Amico, and Silvano Martello. *Assignment Problems*. SIAM, 2009.
- [105] Dimitri P Bertsekas. A distributed algorithm for the assignment problem. *Lab. for Information and Decision Systems Working Paper, MIT*, 1979.
- [106] V. Srinivasan and Gerald L. Thompson. Cost operator algorithms for the transportation problem. *Math. Program.*, 12(1):372–391, 1977.
- [107] Francesc Serratosa. Fast computation of bipartite graph matching. *Pattern Recognit. Lett.*, 45:244–250, 2014.

- [108] Andreas Fischer, Ching Y. Suen, Volkmar Frinken, Kaspar Riesen, and Horst Bunke. Approximation of graph edit distance based on hausdorff matching. *Pattern Recognit.*, 48(2):331–343, 2015.
- [109] Daniel P. Huttenlocher, Gregory A. Klanderman, and William Rucklidge. Comparing images using the hausdorff distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(9):850–863, 1993.
- [110] David B. Blumenthal, Sébastien Bogleux, Johann Gamper, and Luc Brun. Ring based approximation of graph edit distance. In Xiao Bai, Edwin R. Hancock, Tin Kam Ho, Richard C. Wilson, Battista Biggio, and Antonio Robles-Kelly, editors, *Structural, Syntactic, and Statistical Pattern Recognition - Joint IAPR International Workshop, S+SSPR 2018, Beijing, China, August 17-19, 2018, Proceedings*, volume 11004 of *Lecture Notes in Computer Science*, pages 293–303. Springer, 2018.
- [111] Sébastien Bogleux, Benoit Gaüzère, David B. Blumenthal, and Luc Brun. Fast linear sum assignment with error-correction and no cost constraints. *Pattern Recognit. Lett.*, 134:37–45, 2020.
- [112] Francesc Serratosà. Speeding up fast bipartite graph matching through a new cost matrix. *Int. J. Pattern Recognit. Artif. Intell.*, 29(2):1550010:1–1550010:17, 2015.
- [113] Pau Riba, Andreas Fischer, Josep Lladós, and Alicia Fornés. Learning graph edit distance by graph neural networks. *Pattern Recognit.*, 120:108132, 2021.
- [114] Kilian Q. Weinberger and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. *J. Mach. Learn. Res.*, 10:207–244, 2009.
- [115] Runzhong Wang, Tianqi Zhang, Tianshu Yu, Junchi Yan, and Xiaokang Yang. Combinatorial learning of graph edit distance via dynamic embedding. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 5241–5250. Computer Vision Foundation / IEEE, 2021.
- [116] Michael Stauffer, Thomas Tschachtli, Andreas Fischer, and Kaspar Riesen. A survey on applications of bipartite graph edit distance. In Pasquale Foggia, Cheng-Lin Liu, and Mario Vento, editors, *Graph-Based Representations in Pattern Recognition - 11th IAPR-TC-15 International Workshop, GbRPR 2017, Anacapri, Italy, May 16-18, 2017, Proceedings*, volume 10310 of *Lecture Notes in Computer Science*, pages 242–252, 2017.
- [117] Stefan Fankhauser, Kaspar Riesen, and Horst Bunke. Speeding up graph edit distance computation through fast bipartite matching. In Xiaoyi Jiang, Miquel Ferrer, and Andrea Torsello, editors, *Graph-Based Representations in Pattern Recognition - 8th IAPR-TC-15 International Workshop, GbRPR 2011, Münster, Germany, May 18-20, 2011. Proceedings*, volume 6658 of *Lecture Notes in Computer Science*, pages 102–111. Springer, 2011.
- [118] Development Therapeutics Program. Aids antiviral screen, 2004. Accessed: 01-05-2023.
- [119] Kaspar Riesen and Horst Bunke. IAM graph database repository for graph based pattern recognition and machine learning. In Niels da Vitoria Lobo et

- al., editor, *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshop, SSPR & SPR 2008, Orlando, USA, December 4-6, 2008. Proceedings*, volume 5342 of *LNCS*, pages 287–297. Springer, 2008.
- [120] Jeroen Kazius, Ross McGuire, and Roberta Bursi. Derivation and validation of toxicophores for mutagenicity prediction. *Journal of Medicinal Chemistry*, 48(1):312–320, 2005.
- [121] R.H. Garrett and C.M. Grisham. *Biochemistry*. Cengage Learning, 2008.
- [122] Natiobnal Center for Biotechnology Information. The pubchem project. Accessed: 01-05-2023.
- [123] Nikil Wale and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China*, pages 678–689. IEEE Computer Society, 2006.
- [124] Nikil Wale, Ian A. Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowl. Inf. Syst.*, 14(3):347–375, 2008.
- [125] Jeffrey J. Sutherland, Lee A. O’Brien, and Donald F. Weaver. Spline-fitting with a genetic algorithm: A method for developing classification structure-activity relationships. *J. Chem. Inf. Comput. Sci.*, 43(6):1906–1915, 2003.
- [126] Ravi G Kurumbail, Anna M Stevens, James K Gierse, Joseph J McDonald, Roderick A Stegeman, Jina Y Pak, Daniel Gildehaus, Julie M Iyashiro, Thomas D Penning, Karen Seibert, et al. Structural basis for selective inhibition of cyclooxygenase-2 by anti-inflammatory agents. *Nature*, 384(6610):644–648, 1996.
- [127] Costantino Iadecola and Mihaela Alexander. Cerebral ischemia and inflammation. *Current opinion in neurology*, 14(1):89–94, 2001.
- [128] Siobhan Sutcliffe and Michel A. Pontari. Chapter 2 - inflammation and infection in the etiology of prostate cancer. In Jack H. Mydlo and Ciril J. Godec, editors, *Prostate Cancer (Second Edition)*, pages 13–20. Academic Press, San Diego, second edition edition, 2016.
- [129] Cleveland Clinic. Cox-2 inhibitors. <https://my.clevelandclinic.org/health/drugs/23119-cox-2-inhibitors>, 2022. Accessed: 2023-05-01.
- [130] File:6cox.png. <https://commons.wikimedia.org/wiki/File:6COX.png>, 2023. Accessed: 2023-05-13.
- [131] File:cyclooxygenase-2.png. <https://upload.wikimedia.org/wikipedia/commons/e/e6/Cyclooxygenase-2.png>, 2023. Accessed: 2023-05-13.
- [132] Gregory W. Kauffman and Peter C. Jurs. QSAR and k-nearest neighbor classification analysis of selective cyclooxygenase-2 inhibitors using topologically-based numerical descriptors. *J. Chem. Inf. Comput. Sci.*, 41(6):1553–1560, 2001.
- [133] Christoph Helma, Ross D. King, Stefan Kramer, and Ashwin Srinivasan. The predictive toxicology challenge 2000-2001. *Bioinform.*, 17(1):107–108, 2001.
- [134] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical in-*

- formation and computer sciences, 28(1):31–36, 1988.
- [135] Peter D Sun, Christine E Foster, and Jeffrey C Boyington. Overview of protein structural and functional folds. *Current Protocols in Protein Science*, Chapter 17(1):Unit 17.1, May 2004. Cited By :26.
 - [136] Gabriel S. Brandt. Secondary structure. In Robert D. Wells, Judith S. Bond, Judith Klinman, and Bettie Sue Siler Masters, editors, *Molecular Life Sciences: An Encyclopedic Reference*, pages 1089–1097, New York, NY, 2018. Springer New York.
 - [137] Helen M. Berman, John D. Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The protein data bank. *Nucleic Acids Res.*, 28(1):235–242, 2000.
 - [138] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003.
 - [139] Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alexander J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. In *Proceedings Thirteenth International Conference on Intelligent Systems for Molecular Biology 2005, Detroit, MI, USA, 25-29 June 2005*, pages 47–56, 2005.
 - [140] Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic Acids Res.*, 32(Database-Issue):431–433, 2004.
 - [141] Jeremy M. Berg, John L. Tymoczko, and Lubert Stryer. *Biochemistry*. W.H. Freeman and Company, 8 edition, 2015.
 - [142] V.K. Chaugule, M.L. Rennie, H. Walden, C. Arkinson, O. Kamarainen, and R. Toth. Crystal structure of human ube2t e54r mutant. *Nature Chemical Biology*, 15:1114–1122, 2019.
 - [143] V.K. Chaugule, M.L. Rennie, H. Walden, C. Arkinson, O. Kamarainen, and R. Toth. Crystal structure of human ube2t e54r mutant. Protein Data Bank, 2019. Initial deposition on 28 March 2019, initial release on 16 October 2019, latest revision on 4 March 2020.
 - [144] D.L. Ollis and D. Pathak. Dienelactone hydrolase at 2.8 angstroms. *Journal of Molecular Biology*, 215:403–410, 1990.
 - [145] D.L. Ollis and D. Pathak. Dienelactone hydrolase at 2.8 angstroms. Protein Data Bank, 1996. Initial deposition on 14 March 1996, initial release on 17 August 1996, latest revision on 13 July 2011.
 - [146] Shuichi Kawashima and Minoru Kanehisa. Aaindex: Amino acid index database. *Nucleic Acids Res.*, 28(1):374, 2000.
 - [147] H. Li, W.A. Hallows, J.S. Punzi, K.W. Pankiewicz, K.A. Watanabe, and B.M. Goldstein. Crystallographic studies of isosteric nad analogues bound to alcohol dehydrogenase: specificity and substrate binding in two ternary complexes. *Biochemistry*, 1995.
 - [148] H. Li, W.A. Hallows, J.S. Punzi, K.W. Pankiewicz, K.A. Watanabe, and B.M. Goldstein. Crystallographic studies of isosteric nad analogues bound to alcohol dehydrogenase: specificity and substrate binding in two ternary

- complexes. Protein Data Bank, 1995. Initial deposition on 13 December 1993, initial release on 3 June 1995, latest revision on 29 November 2017.
- [149] Y. Peng and V.C. Yee. Methyltransferase. *Biochemistry*, 2011.
- [150] Y. Peng and V.C. Yee. Methyltransferase. Protein Data Bank, 2011. Initial deposition on 25 April 2011, initial release on 7 September 2011, latest revision on 21 September 2011.
- [151] D.A. Lang, P. Stadler, A. Kovacs, F. Paltauf, and B.W. Dijkstra. New crystal form of pseudomonas glumae (formerly chromobacterium viscosum atcc 6918) lipase. Protein Data Bank, 1999. Initial deposition on 27 April 1999, initial release on 6 May 1999, latest revision on 27 November 2019.
- [152] M. Ho and K.N. Allen. Structural studies of acetoacetate decarboxylase. *Nature*, 2008.
- [153] M. Ho and K.N. Allen. Structural studies of acetoacetate decarboxylase. Protein Data Bank, 2008. Initial deposition on 27 November 2007, initial release on 23 December 2008, latest revision on 20 October 2021.
- [154] P.D. Kiser, D.T. Lodowski, M. Golczak, and K. Palczewski. Crystal structure of bovine rpe65 at 1.9 angstrom resolution. *Journal of Biological Chemistry*, 2010.
- [155] P.D. Kiser, D.T. Lodowski, M. Golczak, and K. Palczewski. Crystal structure of bovine rpe65 at 1.9 angstrom resolution. Protein Data Bank, 2010. Initial deposition on 29 November 2009, initial release on 2 February 2010, latest revision on 1 November 2017.
- [156] I. Sharon and T.M. Schmeing. Crystal structure of cyanophycin synthetase 2 from gloeotheca citrifomis. *ACS Chemical Biology*, 2022.
- [157] I. Sharon and T.M. Schmeing. Crystal structure of cyanophycin synthetase 2 from gloeotheca citrifomis. Protein Data Bank, 2022. Initial deposition on 20 December 2021, initial release on 2 March 2022, latest revision on 30 March 2022.
- [158] JEAN-LUC FAUCHÈRE, Marvin Charton, Lemont B Kier, Arie Verloop, and Vladimir Pliska. Amino acid side chain parameters for correlation studies in biology and pharmacology. *International journal of peptide and protein research*, 32(4):269–278, 1988.
- [159] Hilda Cid, Marta Bunster, Mauricio Canales, and Felipe Gazitúa. Hydrophobicity and structural classes in proteins. *Protein Engineering, Design and Selection*, 5(5):373–375, 1992.
- [160] Richard Grantham. Amino acid difference formula to help explain protein evolution. *science*, 185(4154):862–864, 1974.
- [161] Marvin Charton and Barbara I Charton. The structural dependence of amino acid hydrophobicity parameters. *Journal of theoretical biology*, 99(4):629–644, 1982.
- [162] IMDb. Imdb: The internet movie database, 1990. Accessed: 2023-05-01.
- [163] Liya Su, Yepeng Yao, Zhigang Lu, and Baoxu Liu. Understanding the influence of graph kernels on deep learning architecture: A case study of flow-based network attack detection. In *18th IEEE International Conferences On Trust, Security And Privacy and On Big Data Science And Engineering, TrustCom/BigDataSE 2019, Rotorua, New Zealand, August 5-8, 2019*,

- pages 312–318. IEEE, 2019.
- [164] Tom Odla. On properties of a well-known graph or what is your ramsey number? *Annals of the New York Academy of Sciences*, 328(1):166–172, 1979.
 - [165] Valerio Arnaboldi, Marco Conti, Massimiliano La Gala, Andrea Passarella, and Fabio Pezzoni. Ego network structure in online social networks and its impact on information diffusion. *Comput. Commun.*, 76:26–41, 2016.
 - [166] Nils M. Kriege, Matthias Fey, Denis Fisseler, Petra Mutzel, and Frank Weichert. Recognizing cuneiform signs using graph based methods. In *International Workshop on Cost-Sensitive Learning, COST@SDM 2018, San Diego, California, USA, May 5, 2018*, volume 88 of *Proceedings of Machine Learning Research*, pages 31–44. PMLR, 2018.
 - [167] D Fisseler, F Weichert, G Müller, and M Cammarosano. Towards an interactive and automated script feature analysis of 3d scanned cuneiform tablets. *Scientific Computing and Cultural Heritage*, page 16, 2013.
 - [168] Christopher Bromhead Fleming Walker. *Cuneiform*, volume 3. Univ of California Press, 1987.
 - [169] Collection object: 1989,0130.4. https://www.britishmuseum.org/collection/object/W_1989-0130-4, 1989. Accessed: 2023-05-01.
 - [170] G. Müller. Hethitologie-portal mainz. <http://www.hethiter.net>, 2000.
 - [171] Christopher Morris, Nils M. Kriege, Kristian Kersting, and Petra Mutzel. Faster kernels for graphs with continuous attributes via hashing. In Francesco Bonchi, Josep Domingo-Ferrer, Ricardo Baeza-Yates, Zhi-Hua Zhou, and Xindong Wu, editors, *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, pages 1095–1100. IEEE Computer Society, 2016.
 - [172] Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.*, 5(1):17–60, 1960.
 - [173] Edgar N Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.
 - [174] Isabelle Guyon. Design of experiments of the nips 2003 variable selection benchmark. In *NIPS 2003 workshop on feature extraction and feature selection*, volume 253, page 40, 2003.
 - [175] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020.
 - [176] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.
 - [177] Geoffrey E. Hinton and Sam T. Roweis. Stochastic neighbor embedding. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada]*, pages 833–840. MIT Press, 2002.
 - [178] A. Gilioz and K. Riesen. Graph-based vs. vector-based classification: A fair comparison. In *13th IAPR-TC15 International Workshop on Graph-Based*

- Representations in Pattern Recognition*, 2023. Currently under review for possible publication.
- [179] Mathias Fuchs and Kaspar Riesen. A novel way to formalize stable graph cores by using matching-graphs. *Pattern Recognit.*, 131:108846, 2022.
 - [180] Albert Solé-Ribalta and Francesc Serratosa. Graduated assignment algorithm for multiple graph matching based on a common labeling. *Int. J. Pattern Recognit. Artif. Intell.*, 27(1), 2013.
 - [181] Mukund Deshpande, Michihiro Kuramochi, Nikil Wale, and George Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Trans. Knowl. Data Eng.*, 17(8):1036–1050, 2005.
 - [182] Kaspar Riesen and Horst Bunke. Classification and clustering of vector space embedded graphs. In *Emerging topics in computer vision and its applications*, pages 49–70. World Scientific, 2012.
 - [183] Mohamed Moussaoui, Montaceur Zaghdoud, and Jalel Akaichi. A survey of uncertainty handling in frequent subgraph mining algorithms. In *12th IEEE/ACS International Conference of Computer Systems and Applications, AICCSA 2015, Marrakech, Morocco, November 17-20, 2015*, pages 1–8. IEEE Computer Society, 2015.
 - [184] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In Djamel A. Zighed, Henryk Jan Komorowski, and Jan M. Zytkow, editors, *Principles of Data Mining and Knowledge Discovery, 4th European Conference, PKDD 2000, Lyon, France, September 13-16, 2000, Proceedings*, volume 1910 of *LNCIS*, pages 13–23. Springer, 2000.
 - [185] Diane J. Cook and Lawrence B. Holder. Substructure discovery using minimum description length and background knowledge. *J. Artif. Intell. Res.*, 1:231–255, 1994.
 - [186] Horst Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognit. Lett.*, 18(8):689–694, 1997.
 - [187] Xiaoyi Jiang, Andreas Münger, and Horst Bunke. On median graphs: Properties, algorithms, and applications. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(10):1144–1151, 2001.
 - [188] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(10):1367–1372, 2004.
 - [189] Rifky Yunus Krisnabayu, Achmad Ridok, and Agung Setia Budi. Hepatitis detection using random forest based on SVM-RFE (recursive feature elimination) feature selection and SMOTE. In *SIET '21: 6th International Conference on Sustainable Information Engineering and Technology 2021, Malang, Indonesia, September 13 - 14, 2021*, pages 151–156. ACM, 2021.
 - [190] Matteo Togninalli, Elisabetta Ghisu, Felipe Llinares-López, Bastian Rieck, and Karsten Borgwardt. Wasserstein weisfeiler-lehman graph kernels. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32 (NeurIPS)*, pages 6436–6446. Curran Associates, Inc., 2019.
 - [191] Mahito Sugiyama, M. Elisabetta Ghisu, Felipe Llinares-López, and

- Karsten M. Borgwardt. graphkernels: R and python packages for graph comparison. *Bioinform.*, 34(3):530–532, 2018.
- [192] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized explainer for graph neural network. In Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.
- [193] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Anal. Appl.*, 13(1):113–129, 2010.
- [194] Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *Appl. Netw. Sci.*, 5(1):6, 2020.
- [195] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 32(1):4–24, 2021.
- [196] Viggo Kann. *On the approximability of NP-complete optimization problems*. PhD thesis, Royal Institute of Technology Stockholm, 1992.
- [197] Ciaran McCreesh, Patrick Prosser, and James Trimble. A partitioning algorithm for maximum common subgraph problems. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 712–719. ijcai.org, 2017.
- [198] Christoph Steinbeck, Yongquan Han, Stefan Kuhn, Oliver Horlacher, Edgar Luttmann, and Egon L. Willighagen. The chemistry development kit (CDK): an open-source java library for chemo- and bioinformatics. *J. Chem. Inf. Comput. Sci.*, 43(2):493–500, 2003.
- [199] Kai Lu, Yi Zhang, Kai Xu, Yinghui Gao, and Richard C. Wilson. Approximate maximum common sub-graph isomorphism based on discrete-time quantum walk. In *22nd International Conference on Pattern Recognition, ICPR 2014, Stockholm, Sweden, August 24-28, 2014*, pages 1413–1418. IEEE Computer Society, 2014.
- [200] Takeshi Kawabata. Build-up algorithm for atomic correspondence between chemical structures. *J. Chem. Inf. Model.*, 51(8):1775–1787, 2011.
- [201] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognit. Lett.*, 19(3-4):255–259, 1998.
- [202] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized explainer for graph neural network. In Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.
- [203] Horst Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognit. Lett.*, 18(8):689–694, 1997.
- [204] Walter D. Wallis, Peter Shoubridge, Miro Kraetzl, and D. Ray. Graph distances using graph union. *Pattern Recognit. Lett.*, 22(6/7):701–704, 2001.
- [205] Frederick Jelinek. Some of my best friends are linguists. *Lang. Resour.*

- Evaluation*, 39(1):25–34, 2005.
- [206] Michele Banko and Eric Brill. Mitigating the paucity-of-data problem: Exploring the effect of training corpus size on classifier performance for natural language processing. In *Proceedings of the First International Conference on Human Language Technology Research, HLT 2001, San Diego, California, USA, March 18-21, 2001*. Morgan Kaufmann, 2001.
 - [207] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *J. Big Data*, 6:60, 2019.
 - [208] F. Pereira, P. Norvig, and A. Halevy. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(02):8–12, mar 2009.
 - [209] Julia Schaefer, Moritz Lehne, Josef Schepers, Fabian Prasser, and Sylvia Thun. The use of machine learning in rare diseases: a scoping review. *Orphanet journal of rare diseases*, 15:1–10, 2020.
 - [210] James K Stoller. The challenge of rare diseases. *Chest*, 153(6):1309–1314, 2018.
 - [211] Frank Cremer, Barry Sheehan, Michael Fortmann, Arash N Kia, Martin Mullins, Finbarr Murphy, and Stefan Materne. Cyber risk and cybersecurity: A systematic review of data availability. *The Geneva Papers on Risk and Insurance-Issues and Practice*, 47(3):698–736, 2022.
 - [212] Ivan Habernal, Henning Wachsmuth, Iryna Gurevych, and Benno Stein. The argument reasoning comprehension task: Identification and reconstruction of implicit warrants. *arXiv preprint arXiv:1708.01425*, 2017.
 - [213] Yuxuan Zhang, Huan Ling, Jun Gao, Kangxue Yin, Jean-Francois Lafleche, Adela Barriuso, Antonio Torralba, and Sanja Fidler. Datasetgan: Efficient labeled data factory with minimal human effort. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 10145–10155. Computer Vision Foundation / IEEE, 2021.
 - [214] Daisuke Takaishi, Hiroki Nishiyama, Nei Kato, and Ryu Miura. Toward energy efficient big data gathering in densely distributed sensor networks. *IEEE Trans. Emerg. Top. Comput.*, 2(3):388–397, 2014.
 - [215] Marcelino Lázaro and Aníbal R. Figueiras-Vidal. Neural network for ordinal classification of imbalanced data by minimizing a bayesian cost. *Pattern Recognition*, 137:109303, 2023.
 - [216] Haibo He and Edwardo A. Garcia. Learning from imbalanced data. *IEEE Trans. Knowl. Data Eng.*, 21(9):1263–1284, 2009.
 - [217] Yanmin Sun, Andrew K. C. Wong, and Mohamed S. Kamel. Classification of imbalanced data: a review. *Int. J. Pattern Recognit. Artif. Intell.*, 23(4):687–719, 2009.
 - [218] Mingle Xu, Sook Yoon, Alvaro Fuentes, and Dong Sun Park. A comprehensive survey of image augmentation techniques for deep learning. *CoRR*, abs/2205.01491, 2022.
 - [219] Vanchinbal Chinbat and Seung-Hwan Bae. Ga3n: Generative adversarial autoaugment network. *Pattern Recognition*, 127:108637, 2022.
 - [220] F. Dornaika, D. Sun, K. Hammoudi, J. Charafeddine, A. Cabani, and C. Zhang. Object-centric contour-aware data augmentation using superpixels of varying granularity. *Pattern Recognition*, 139:109481, 2023.

- [221] Devesh Walawalkar, Zhiqiang Shen, Zechun Liu, and Marios Savvides. Attentive cutmix: An enhanced data augmentation approach for deep learning based image classification. In *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*, pages 3642–3646. IEEE, 2020.
- [222] Jang-Hyun Kim, Wonho Choo, and Hyun Oh Song. Puzzle mix: Exploiting saliency and local statistics for optimal mixup. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 5275–5285. PMLR, 2020.
- [223] Markus Bayer, Marc-André Kauffhold, and Christian Reuter. A survey on data augmentation for text classification. *ACM Comput. Surv.*, 55(7):146:1–146:39, 2023.
- [224] Steven Y. Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard H. Hovy. A survey of data augmentation approaches for NLP. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Findings of the Association for Computational Linguistics: ACL/IJCNLP 2021, Online Event, August 1-6, 2021*, volume ACL/IJCNLP 2021 of *Findings of ACL*, pages 968–988. Association for Computational Linguistics, 2021.
- [225] Gözde Gül Sahin and Mark Steedman. Data augmentation via dependency tree morphing for low-resource languages. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun’ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 5004–5009. Association for Computational Linguistics, 2018.
- [226] Jason W. Wei and Kai Zou. EDA: easy data augmentation techniques for boosting performance on text classification tasks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 6381–6387. Association for Computational Linguistics, 2019.
- [227] Yudianto Sujana and Hung-Yu Kao. Lida: Language-independent data augmentation for text classification. *IEEE Access*, 11:10894–10901, 2023.
- [228] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.
- [229] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver J. Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 11015–11023. AAAI Press, 2021.
- [230] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring

- and relieving the over-smoothing problem for graph neural networks from the topological view. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 3438–3445. AAAI Press, 2020.
- [231] Jiajun Zhou, Jie Shen, Shanqing Yu, Guanrong Chen, and Qi Xuan. M-evolve: Structural-mapping-based data augmentation for graph classification. *IEEE Trans. Netw. Sci. Eng.*, 8(1):190–200, 2021.
- [232] Joonhyung Park, Hajin Shim, and Eunho Yang. Graph transplant: Node saliency-guided graph mixup with local structure preservation. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 7966–7974. AAAI Press, 2022.
- [233] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.
- [234] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.
- [235] Thomas N. Kipf and Max Welling. Variational graph auto-encoders. *CoRR*, abs/1611.07308, 2016.
- [236] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In Vera Kurková, Yannis Manolopoulos, Barbara Hammer, Lazaros S. Iliadis, and Ilias Maglogiannis, editors, *Artificial Neural Networks and Machine Learning - ICANN 2018 - 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I*, volume 11139 of *Lecture Notes in Computer Science*, pages 412–422. Springer, 2018.
- [237] Lin Guo and Qun Dai. Graph clustering via variational graph embedding. *Pattern Recognit.*, 122:108334, 2022.
- [238] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680, 2014.
- [239] Adyasha Maharana and Mohit Bansal. Grada: Graph generative data augmentation for commonsense reasoning. In Nicoletta Calzolari, Chu-Ren Huang, Hansaem Kim, James Pustejovsky, Leo Wanner, Key-Sun Choi,

- Pum-Mo Ryu, Hsin-Hsi Chen, Lucia Donatelli, Heng Ji, Sadao Kurohashi, Patrizia Paggio, Nianwen Xue, Seokhwan Kim, Younggyun Hahm, Zhong He, Tony Kyungil Lee, Enrico Santus, Francis Bond, and Seung-Hoon Na, editors, *Proceedings of the 29th International Conference on Computational Linguistics, COLING 2022, Gyeongju, Republic of Korea, October 12-17, 2022*, pages 4499–4516. International Committee on Computational Linguistics, 2022.
- [240] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. Graphgan: Graph representation learning with generative adversarial nets. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 2508–2515. AAAI Press, 2018.
 - [241] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *CoRR*, abs/1805.11973, 2018.
 - [242] Omer Sagi and Lior Rokach. Ensemble learning: A survey. *WIREs Data Mining Knowl. Discov.*, 8(4), 2018.
 - [243] Domor Mienye Ibomoiye and Yanxia Sun. A survey of ensemble learning: Concepts, algorithms, applications, and prospects. *IEEE Access*, 10:99129–99149, 2022.
 - [244] Cha Zhang and Yunqian Ma. *Ensemble Machine Learning: Methods and Applications*. Springer Publishing Company, Incorporated, 2012.
 - [245] Xibin Dong, Zhiwen Yu, Wenming Cao, Yifan Shi, and Qianli Ma. A survey on ensemble learning. *Frontiers Comput. Sci.*, 14(2):241–258, 2020.
 - [246] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.
 - [247] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.
 - [248] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 1024–1034, 2017.
 - [249] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.
 - [250] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.
 - [251] Leo Breiman. Bagging predictors. *Mach. Learn.*, 24(2):123–140, 1996.

- [252] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Erklärung

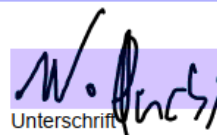
gemäß Art. 18 PromR Phil.-nat. 2019

Name/Vorname:	Fuchs Mathias
Matrikelnummer:	09-923-764
Studiengang:	Computer Science
	Bachelor <input type="checkbox"/> Master <input type="checkbox"/> Dissertation <input checked="" type="checkbox"/>
Titel der Arbeit:	The Matching-Graph
LeiterIn der Arbeit:	PD. Dr. Kaspar Riesen

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe r des Gesetzes über die Universität vom 5. September 1996 und Artikel 69 des Universitätsstatuts vom 7. Juni 2011 zum Entzug des Dokortitels berechtigt ist. Für die Zwecke der Begutachtung und der Überprüfung der Einhaltung der Selbständigkeitserklärung bzw. der Reglemente betreffend Plagiate erteile ich der Universität Bern das Recht, die dazu erforderlichen Personendaten zu bearbeiten und Nutzungshandlungen vorzunehmen, insbesondere die Doktorarbeit zu vervielfältigen und dauerhaft in einer Datenbank zu speichern sowie diese zur Überprüfung von Arbeiten Dritter zu verwenden oder hierzu zur Verfügung zu stellen.

Bern, 20.07.2023

Ort/Datum


Unterschrift