Pattern Recognition on Reduced Graphs

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Anthony Gillioz

von Isérable VS

Leiter der Arbeit:

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

Pattern Recognition on Reduced Graphs

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Anthony Gillioz

von Isérable VS

Leiter der Arbeit:

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

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Bern, den 27.05.2024

Der Dekan:
Prof. Dr. Marco Herwegh

Abstract

In both everyday life and business, a huge amount of data is generated, underscoring the need for researching and developing efficient and accurate methods to automatically process these massive amounts of data. The data generated is often inherently complex, making traditional feature vectors not well suited for data representation. Graphs provide a general and versatile data representation that can be used to represent a wide range of complex systems. In addition, a variety of graph-based pattern recognition algorithms have been proposed in recent years, further strengthening this approach.

One of the main challenges hindering the widespread use of graph-based pattern recognition, is the expensive computation time required by most of the graph-based algorithms. Although approximation algorithms have been proposed for various tasks, the computation time remains a challenge, in particular when the graph sizes increase. In this thesis, we aim to address the problem of computation time by reducing the input size of the underlying graphs through an approximation of the data. In this particular case, data approximation refers to reducing the size of a graph while maintaining its essential properties. The major hypothesis of this thesis is as follows. As the input problem is reduced, the overall computation cost of graph-based pattern recognition algorithm should be reduced as well. The crucial question is, of course, what happens to the accuracy of the respective methods when they operate on reduced rather than on the original data.

In the present thesis, four graph reduction methods are introduced and thoroughly evaluated in the context of graph-based pattern recognition. The first technique is based on sampling the most important nodes within a graph. To this end, centrality measures to compute scores for each node are used. Next, one can remove a given percentage of the nodes with

the lowest scores to generate reduced versions of the graphs. The second method is based on a spectral clustering algorithm of the graphs, which identifies communities within a graph. Those communities are aggregated into super-nodes, and the inter-community edges are aggregated into super-edges to create reduced graphs. The third method uses a modified graph neural network to learn the importance of each node in the graph from the data. Based on these learned scores, the graph reduction method removes the nodes with the least importance. The fourth method uses a compression-based distance metric. In this approach, graphs are reduced with a compression algorithm and a distance is derived out of this compression.

To evaluate the benefits and imitations of the four reduction methods, we perform comprehensive empirical evaluations of all reduction methods on different real-world datasets. In this context, we compare both their classification accuracy and their computational efficiency with the results obtained on the original graphs. A wide variety of graph classifiers are used. In general, the evaluation confirms that even highly reduced graphs maintain satisfactory classification accuracies and can significantly speed up graph-based pattern recognition. Furthermore, we utilize the reduced graphs in various novel graph matching frameworks with the general aim to improve the overall classification accuracy, and we succeed in this endeavor in several scenarios.

Acknowledgments

My heartfelt thanks go to my supervisor, PD Dr. Kaspar Riesen, for his invaluable guidance and encouragement throughout this journey. I am especially thankful for his role in helping me face my deepest fear: "the blank page syndrome". With perseverance and a willingness to face this challenge head-on, I gradually gained confidence in my writing abilities. As a result, I made significant progress in overcoming the initial obstacles and moving forward with my work. The guidance and support of PD Dr. Riesen were pivotal in this process, and I am sincerely thankful for his mentorship. I extend my gratitude to Prof. Dr. Benoit Gaüzère for serving as the co-referee of this thesis and to Prof. Dr. Timo Kehrer for supervising the examination process. Additionally, I would like to acknowledge the funding provided by the Swiss National Science Foundation under Project 200021_188496.

A big shout-out to my colleagues of the Pattern Recognition Group for their camaraderie, support, and shared experiences. In particular, I would like to thank Dr. Mathias Fuchs, my co-PhD student. Mathias has always been open to discuss and share new ideas on a wide range of topics, extending beyond technical matters. For this, I am truly thankful.

I owe a debt of gratitude to the faculty and staff of the Institute of Computer Science, especially Dr. Peppo Brambilla, Bettina Choffat, and Prof. Dr. Thomas Studer. Their consistently positive attitude, moral support and assistance with numerous technical and administrative challenges have been invaluable. Additionally, I appreciate their role in providing a friendly environment during our coffee breaks, which has contributed to a positive working atmosphere.

Je tiens à remercier ma mère, mon frère et ma sœur pour leur soutien et leur gentillesse tout au long de ma thèse. Votre encouragement m'a donné la force et la motivation nécessaires pour poursuivre cette aventure (pour

ça, merci maman). To the friends I have made along the way, I would like to express my sincere gratitude. Thank you for always providing me with a space to clear my mind, both during the good and the bad times. Your friendship and support have been invaluable throughout this journey.

Last but not least, I want to extend my gratitude to Sabrina Selva for her always positive attitude (except when we have to leave early in the morning, lmao XD). Thank you for your unwavering support during the final, stressful months of the thesis. Your encouragement has always been a source of strength to me.

Contents

Introduction 1

> J'ai été élevé selon le principe que l'oisiveté est mère de tous vices. Comme j'étais un enfant pétris de vertu, je croyais tout ce qu'on me disait, et je me suis ainsi doté d'une conscience qui m'a contraint à peiner au travail toute ma vie.
>
> *Éloge de l'oisiveté* (1932), Bertrand Russell

Pattern Recognition is the ability to identify, analyze, and interpret recurring patterns in data. This process happens both in the human brain and through computer algorithms. For humans, it helps us navigate the world, make sense of information, and learn new things. For computers, it enables them to automate tasks, make predictions, and perform complex analyses. Humans do it intuitively, while computers rely on algorithms to identify patterns in data. Patterns can be simple, like the stripes on a zebra, or complex, like the trends in the stock market.

The human brain is constantly engaged in pattern recognition and uses it in our everyday life. For instance, when we meet a friend, our brain compares the visual information with the facial patterns in our memory, which enables recognition. Spoken and written words also follow patterns (e.g., words, grammar or syntax), which our brain learns to decode. Even when we are listening to music, our brain recognizes patterns of melody, rhythm and harmony. These patterns help us to anticipate the notes or beats that follow and give meaning to the music we hear. By analyzing features and comparing them to known patterns, we can extract meaning from data and make informed decisions.

As mentioned above, computers use algorithms to recognize patterns. Pattern recognition algorithms can be used to automate tasks, make pre-

dictions, and perform complex analyses. In general, pattern recognition algorithms can be divided into different categories based on the way they actually recognize the patterns. Some algorithms rely on a set of fine-tuned features, while others rely on features learned directly from data. One of the main advantages of computer algorithms over the human brain is their ability to process large amounts of data, recognize hidden complex patterns, and uncovering subtle trends humans might miss. Moreover, pattern recognition algorithms can perform complex computations and repetitive tasks both quickly and accurately, making them invaluable tools in various research areas. Actually, automatic pattern recognition has proven to be superior to manual analysis in many situations [1].

In areas where vast amounts of data are generated pattern recognition algorithms are even more valuable. For instance, in a field such as bioinformatics, the advent of genome sequencing techniques has generated vast amounts of data that can be used to describe complex life at the molecular level [2]. This genetic data is important for advancing medical research and understanding the complex interactions between genes in microbial communities. Similarly, the data-driven forecasting has also begun in the field of weather forecasting. Modern satellite systems and complex networks of sensors collect vast amounts of stratospheric data, including temperature, humidity, and wind speed, giving meteorologists a better understanding of weather patterns [3]. By analyzing this data, meteorologists can make more accurate weather forecasts and predict long-term climate trends and potential natural disaster. However, the exponential growth of data brings with it new challenges, ranging from the development of efficient management and analysis tools to ethical implications in data use.

Pattern recognition has become one of the cornerstones of artificial intelligence and its importance cannot be overestimated, as it allows machines to efficiently recognize patterns and learn from large amounts of data. Pattern recognition can be divided into two main approaches, each of which offers different perspectives and has its own strengths and weaknesses depending on the scenario. The first approach is *statistical pattern recognition*. This approach analyzes the statistical properties of features in the data, such as means, variances and co-occurrences. The statistical approach is characterized by its ease of implementation and interpretation, its efficiency in dealing with large data sets and its robustness to noise and data variability. The second approach is *structural pattern recognition*, which focuses on analyzing spatial or relational arrangements of data elements to describe the data components. This approach has the advantage that it can de-

scribe more complex data elements (including relationships). In addition, structural shape recognition provides valuable information about the internal structure of shapes, which improves the understanding of complex data models. In the remainder of the present thesis, we focus our attention on structural pattern recognition.

Graph matching is an important part of structural pattern recognition, in which correspondences between graphs or graph-like structures (e.g. trees) are compared and found. Graphs are mathematical representations of nodes connected by edges and can be used to model a variety of structured data, such as networks, molecular structures and relational databases. In the context of pattern recognition, graph matching algorithms aim to recognize similarities and correspondences between (sub-)graphs, enabling, for instance, pattern matching or relationship inference.

Over the last four decades, structural pattern recognition with graphs has evolved significantly, leading to the development of various *graph matching algorithms* [4; 5], *graph kernels* [6; 7], and *graph neural networks* [8; 9].

- Graph matching algorithms have evolved to accurate and efficient methods for finding matches between graph components. Former methods typically involved subgraph isomorphism [10] or graph edit distance [11]. More, recent developments have led to the emergence of spectral graph matching methods [12; 13; 14] and approaches based on continuous optimization [15; 16; 17].
- Graph kernels aim to define similarity measures between graphs by embedding them in high-dimensional feature spaces where conventional machine learning algorithms can operate. Different type of graph kernels have been proposed, including random walk kernels [18], shortest-path kernels [19], or Weisfeiler-Lehman subtree kernels [20], each describing a different aspect of graph structure and topology.
- Graph neural networks have been shown to be an effective class of models for learning representations of graph data. Graph neural networks use the structure of a graph to send information messages between nodes, thus allowing to represent complex dependencies and interactions in the graph. Architectures such as graph convolutional network [21], graph attention network [22], and gated graph recurrent network [23] have been proposed and have achieved state-of-the-art performance on a variety of graph-based tasks.

Graph matching algorithms, graph kernels and graph neural networks have shown promising results in overcoming various challenges inherent to structural pattern recognition. Their application span a wide range of fields, including signature verification [24], biological network predictions [25], social network analysis [26], and others.

However, graph-based techniques in pattern recognition are still limited in some cases due to expensive computational requirements. Actually, the computational cost of pattern analysis and detection using graph-based representation is high due to their ability to model complex structural relationships. For instance, graph matching algorithms aim to find correspondences between nodes and edges of different graphs, which can be computationally expensive as the number of possible correspondences increases exponentially depending on the size of the graph [27]. Similarly, graph kernels compute pairwise similarities between graphs by embedding them into higher dimensional feature spaces, which increases the computational requirements, especially for large graphs [7]. In addition, graph neural networks have computational time issues as they iteratively pass information between neighboring nodes. Especially for large, densely connected graphs, this requires significant computational resources and time for learning and evaluation [9]. As a result, the computational challenges hinder the widespread use of graph-based methods in real-world applications.

To address the computational issues of graphs in pattern recognition, approximation techniques have been proposed for both graph matching [11; 28] and graph kernels [29; 30]. Additionally, for graph neural networks more efficient training procedures have been researched [31]. However, working with large graphs remains a challenge even with these more efficient algorithms.

A complementary strategy to improve the efficiency of graph-based pattern recognition is to use simplified graphs. The simplification can be achieved through *graph reduction* [32; 33; 34] and involves reducing the number of nodes and edges in the graph. However, the main difficulty lies in finding graph reduction techniques that preserves the main topology and properties of the original graph [33]. There exists three major strategies to perform graph reduction, namely *graph summarization*, *graph coarsening*, and *hierarchical graph reduction*.

Graph summarization [32] is a graph reduction strategy that can be used to discover complex patterns in graphs. Graph summarization methods consist of selecting the most relevant nodes in the graph structure, and

ultimately removing the nodes with the lowest importance in a sampling strategy. Graph summarization eases the discovery of complex patterns in structural data and is employed in a wide range of applications, such as *community detection* [35], *classification* [36], and *visualization* [37].

Graph coarsening [33; 34] is the second prominent graph reduction strategy. Graph coarsening methods consist of clustering nodes together into super-nodes and aggregating the inter-cluster edges into super-edges. That is, unlike graph summarization, graph coarsening techniques do not remove nodes and/or edges while reducing the graphs, but rather merge substructures. Graph coarsening methods have found application in machine learning, where coarsened graphs are used to speed up the training of graph neural networks [33] or in electrical networks, where coarsening techniques are used to obtain lower dimensional electrically equivalent circuits [38].

Hierarchical graph representation [39] is a third prominent graph reduction approach. This approach makes use of graph summarization and graph coarsening methods to progressively reduce the original graphs as the number of reduction levels increases. The key concept of hierarchical graph representation methods is to construct a pyramid of subgraphs and then use the subgraphs at the highest level to perform the graph matching. For instance, in [40; 41], hierarchical representation for graphs is used in a pattern recognition context. The authors propose to embed graphs into a vector space and use a community detection method to find the nodes to merge. Thereby, they create a representation that encodes the abstract information while preserving the relationship with the initial graph.

In the present thesis, we propose and analyze various graph reduction methods from both graph summarization and graph coarsening in the context of structural pattern recognition. The main goal of the present thesis is to generate reduced graphs that, when used with graph-based pattern recognition methods (i.e., graph matching, graph kernels, or graph neural networks), can maintain, or even outperform, the results achieved with the original graphs. In order to verify whether we have achieved our goal, we conduct thorough evaluations using datasets stemming from a wide range of domains.

The remainder of the thesis is organized as follows. First, Chapter 2 outlines the theoretical background necessary to understand the details of this thesis. Next, we present 28 graph datasets used throughout the thesis in Chapter 3 (the graph datasets represent chemical compounds as well as data from bioinformatics, computer vision, and social media net-

works). In Chapter 4, we introduce a first graph reduction method that is based on centrality measures. We also describe how the generated reduced graphs can be employed for graph-based pattern recognition (this chapter is the first major part of the present contribution and is is based on four papers [42; 43; 44; 45]). Following this, in Chapter 5, we introduce and analyze a novel graph reduction method that uses spectral clustering (this chapter is based on one journal paper [46]). The final part of the thesis is presented in Chapter 6, where we propose two further graph reduction methods using graph neural networks and a compression based distance metric (this chapter summarizes two preliminary papers [47; 48]). Finally, Chapter 7 draws general conclusions and provides a thorough list of ideas for future research activities.

Graph Based Pattern Recognition

2

> La crise est le moment où l'ancien ordre du monde s'estompe et où le nouveau doit s'imposer en dépit de toutes les résistances et de toutes les contradictions. Cette phase de transition est justement marquée par de nombreuses erreurs et de nombreux tourments.
>
> *Cahiers de prison* (1948), Antonio Gramsci

This chapter closely follows the theory and structure presented in [11; 49]. It provides the essential theoretical foundation on which the main contribution of this thesis is based (described in Chapters 4, 5 and 6)[1]. The present chapter is structured as described in the following paragraph, with a visual summary shown in Fig. 2.1.

First, Section 2.1 gives a basic overview of the essential theoretical foundations of machine learning. Subsequently, the distinction between statistical and structural learning is discussed in detail in Section 2.2. In particular, the focus is on graph data structures, as it is the main object of investigation of the present thesis. A common task related to graphs, and a prominent one addressed in this thesis, is to perform some kind of matching between any two (sub)graphs. Therefore, in Section 2.3 the concept of graph matching and the main differences between exact and inexact graph matching are explained in detail. In Section 2.4, we present three families of popular graph-based pattern recognition, namely *Graph Edit Distance*, *Graph Kernel*, and *Graph Neural Network*. These three families of graph algorithms are used in association with three classifiers to perform the final classification (namely, a *k-Nearest Neighbor* classifier, a *Support Vector Ma-*

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

chine, and a *Neural Network*). Throughout the remainder of this thesis, all of these graph classifiers play a pivotal role and serve as a verification protocol to evaluate the performance of the proposed graph reduction methods. Finally, in Section 2.5, we introduce the general concept of graph reduction and explain the main challenges of graph reduction in the context of pattern recognition.

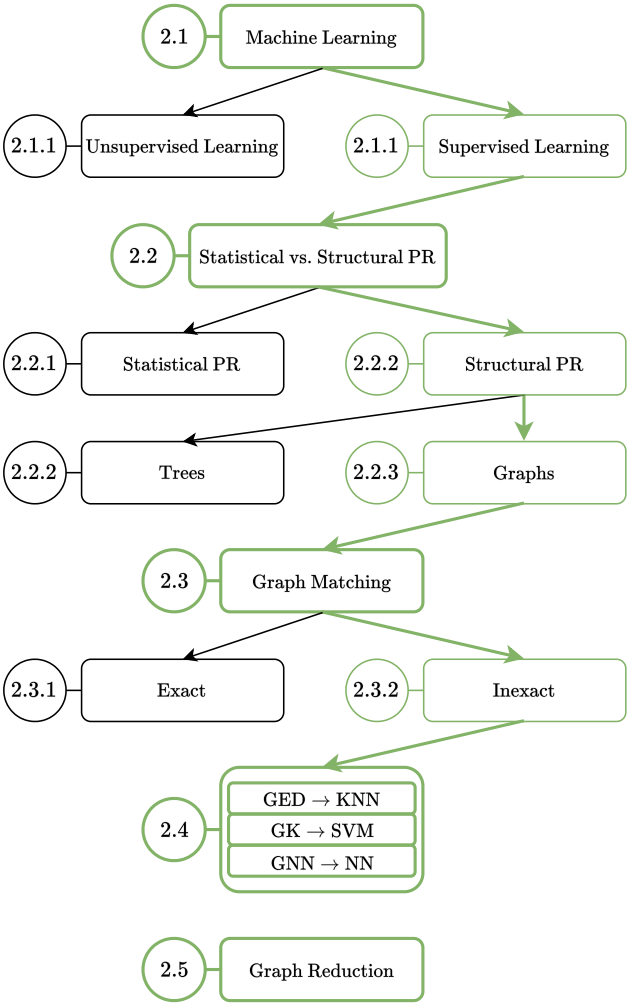
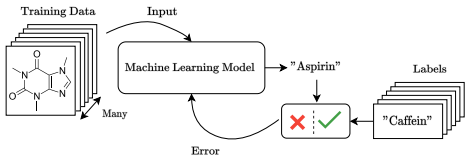


Fig. 2.1: Diagram illustrating the organization of Chapter 2.

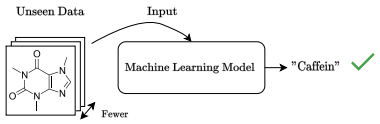
2.1 Machine Learning

Machine Learning is a subfield of *Artificial Intelligence* that enables algorithms to improve their performance in a data-driven manner. To be more precise, the general machine learning process involves fitting mathematical models on data to identify patterns and, ultimately, to make accurate predictions or decisions. Machine learning algorithms differ from other conventional methods in that they do not rely on a rigid set of rules but on learning patterns and relationships directly from data, allowing them to improve their performance over time.

Standard machine learning algorithms operate in two main stages. Fig. 2.2[2] illustrate those two key stages. First, machine learning algorithms iteratively learn from the data in the *training phase*. In this phase, machine learning algorithms are fed a large amount of data so that they learn to recognize the relevant patterns in the data. Once the algorithm is trained, it can be used in the *inference phase*, which involves making predictions or decisions on previously unseen data.



(a) Training Phase



(b) Inference Phase

Fig. 2.2: (a) Training Phase: The machine learning algorithm processes the training data to learn patterns and relationships within the dataset. (b) Inference Phase: The trained model applies the learned knowledge to make predictions on new, unseen data, proving its ability to generalize beyond the training data.

[2]Image adapted from: `https://developer-blogs.nvidia.com/wp-content/uploads/2015/08/training_inference1.png`

2.1.1 *Learning Paradigms*

In machine learning, there exist many common learning paradigms. In the following paragraphs, we describe those that are most commonly used.

Supervised learning algorithms are machine learning methods that train on a set of labeled data, where each input has a known output. The algorithm learns to predict the output for new inputs based on the patterns it has learned from the training data. Formally, for a set of labeled data $\mathcal{D} = \{(x_1, y_1), ..., (x_n, y_n)\}$, where $x_i \in \mathcal{X}$ represents the input and $y_i \in \mathcal{Y}$ the corresponding output, a supervised learning algorithm is a function f with parameters θ that maps the input x_i to a possible output y_i.

Note that both the input and output spaces $\mathcal{X}$ and $\mathcal{Y}$ can belong to any domain, e.g., these spaces can be continuous, discrete, or a graph space (as used in the present thesis and formally defined in Section 2.2.3). The objective of training a supervised learning algorithm is to find the optimal values of the parameters θ that minimize the total loss over the entire dataset.

Formally, we aim at optimizing

$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{n} L(f(x_i; \theta), y_i) \tag{2.1}$$

where $L(\cdot)$ is a loss function that measures the error between the predicted output $f(x_i; \theta) = \hat{y}_i$ and the true output y_i. The loss function is typically chosen among convex functions so that the optimization problem can be efficiently solved. The definition of the actual loss function may vary according to the learning task in question (e.g., the loss can be the *Mean Square Error* [50], or the *Cross-Entropy* [51], to name two prominent examples).

Common examples of supervised learning algorithms include *linear regression* [51], *logistic regression* [51], *Bayesian decision* [51], *support vector machines* [51], and *neural networks* [52]. Each of these algorithms approaches the learning problem in a different way, but all use the basic definition of supervised learning described above.

Unsupervised Learning involves discovering essential patterns, relationships, and structures of the data without explicitly accessing the true output. In this scenario, we have access to a given dataset $\mathcal{D} = \{x_1, \ldots, x_n\}$, where each $x_i \in \mathcal{X}$ represents a data point without corresponding output y_i. The main objective of an unsupervised learning algorithm is to learn a function that captures the underlying structure of the

data. This includes, for instance, tasks such as learning the entire distribution that generated the data (e.g., by density estimation [53], or similar).

Another prominent task in unsupervised learning is *dimensionality reduction* [54], which involves transforming high-dimensional data points $x_i \in \mathbb{R}^n$ into lower-dimensional representations $\hat{x}_i \in \mathbb{R}^m$ using a transformation function $f : \mathbb{R}^n \to \mathbb{R}^m$ with $n > m$. The aim of this mapping is to accurately represent the data in the lower-dimensional space. That is, dimensionality reduction methods generally aim to find a mapping f that retains certain attributes of the data, such as distance or local neighborhood structure, to ensure that the low-dimensional representation accurately captures the relevant information in the data.

A third fundamental task performed in an unsupervised setting is *data clustering* [55; 56], which involves dividing data into groups, or clusters, based on some similarity measure. Formally, given a dataset $\mathcal{D} = \{x_1, \ldots, x_n\}$, the objective is to partition the data into k clusters, denoted as $\mathcal{C} = \{C_1, \ldots, C_k\}$, so that each cluster contains at least one data point (non-empty clusters), no data point belongs to more than one cluster (mutual exclusivity), and all data points belong to at least one cluster (comprehensive coverage). Clustering algorithms attempt to minimize a defined criterion (i.e. a predefined similarity measure) for points within the same cluster (intra-cluster points), aiming for similarity of feature attributes. At the same time, clusterings aim to maximize the criterion for points in different clusters (inter-cluster points), which implies dissimilarity in feature space.

There are different types of clustering algorithms available, including centroid-based methods (e.g., K-means [55]), density-based methods (e.g., DBSCAN [57]), and hierarchical methods (e.g., agglomerative clustering [55]), each with its mathematical formulation and approach to data partitioning.

Semi-supervised Learning tackles the challenge of data labeling, a task known for its tedious nature [58]. It aims to extract features from the data using a minimal set of labeled examples. Some semi-supervised learning methods use unsupervised information from the data to infer pseudo-labels and thus augmenting the pool of labeled data in the dataset [59]. This is typically done in an iterative process. Initially, the unlabeled data are pseudo-labeled, followed by an evaluation phase to assess the quality of the labeling. This process is repeated, progressively refining the labels for the unlabeled data, until a certain criterion is met.

Reinforcement learning is a type of machine learning in which an agent interacts with its environment [60]. This interaction involves the agent performing actions and then receiving feedback based on these actions. More precisely, the agent performs actions, observes the environment's response, and learns from its experiences to improve its policy, which is a set of rules for selecting actions [60]. Reinforcement learning algorithms typically use a trade-off between *exploration* and *exploitation* to learn the optimal policy. Exploration involves trying out new actions to learn more about the environment, while exploitation involves performing the actions known to be most likely to lead to high rewards. In recent years, many reinforcement learning techniques have been developed, including *Q-learning* [61], *Policy Iteration* [62], and *Deep Reinforcement Learning* [63].

2.2 Statistical vs. Structural Pattern Recognition

The theory of machine learning provided in the previous section is, in general, applicable to any data regardless of the actual representation formalism. Machine learning methods are based on the principle of learning how to match the data representation to the desired output. Nevertheless, data representation is a key factor in the performance of any machine learning algorithm. It is, therefore, essential to determine the appropriate data representation for a given learning task. However, determining the optimal representation is a difficult task, as it depends on the underlying problem. The data representation for a given problem can be chosen at two different levels.

The first level essentially determines the appropriate data structure for modeling the problem at hand. The following subsections describe the main data representations used in modern machine learning scenarios, viz. statistical and structural approaches. Statistical methods (described in Subsection 2.2.1) focus on modeling and classifying patterns in a statistical domain (e.g., the real vector space $\mathbb{R}^n$). Structural methods (described in Subsection 2.2.2), particularly in the context of graph-based pattern recognition, analyze the structural relationships between the components of graphs.

The second level of data representation is that – once the data structure is chosen – the representation is refined or transformed. This can be done statically via a *feature selection* [64] process that iteratively refines the optimal features to be used for data representation. More advanced methods,

such as *deep learning* [52], simultaneously learn data representation and pattern recognition directly from raw data.

In the present thesis, we focus primarily on structural representations (in particular, graph-based data structures) and present various novel approaches for graph refinement and graph reduction.

2.2.1 *Statistical Pattern Recognition*

Statistical pattern recognition relies heavily on probability theory and statistics. It treats patterns as the results of random processes and aims to model the underlying statistical distributions of the data [51]. In statistical pattern recognition, data is generally represented as vectors in an n-dimensional space $\mathbb{R}^n$. Each dimension of this space corresponds to a feature, or attribute, that characterizes the objects or entities under consideration.

Representing data as feature vectors $x \in \mathbb{R}^n$ provides efficient mathematical operations in a vector space and enables the use of a wide range of algorithmic tools for statistical pattern recognition. For example, images can be formally defined as a discrete grid of pixel vectors (for 2D images or for 3D images like MRI scans). In this case, each pixel vector represents the color (or the intensity) of a single pixel in the image. The matrix dimensions correspond to the resolution of the image, and the number of columns in the matrix corresponds to the number of color channels in the image. This formal definition allows for the rigorous mathematical treatment of images and underlies various techniques used in computer vision [65], and related fields.

Representation as feature vectors, however, has two major limitations. The first is that vectors, since they represent a predetermined set of features, must have a constant length regardless of the size or complexity of the data they represent. The second limitation is that vectors cannot easily describe binary, or higher-order, relationships that may exist between different components of the underlying data. These drawbacks become significant when dealing with data characterized by complex structural relationships rather than a fixed set of features.

2.2.2 *Structural Pattern Recognition*

When the underlying data consists of both features and relationships that might exist between different parts of the data, we consider this to be *struc-*

tural pattern recognition. That is, structural pattern recognition highlights relationships and structures within the data. It is therefore particularly well-suited for applications where understanding the arrangement or relationships between components is essential [4; 5]. One of the best ways to formalize such data is to use trees or, more generally, graphs.

A *Tree* is a hierarchical data structure made up of nodes and edges. Each node can have zero or more child nodes, and each edge connects a parent node to a child node. In case of a rooted tree, the root node is the top-level node in the tree and has no parent nodes. The tree data structure is a versatile tool for representing and organizing data in a hierarchical form, enabling information to be retrieved, analyzed, and navigated efficiently [66].

A *Graph* is a ubiquitous data structure that can be used to represent a wide variety of problems. A graph is a mathematical structure used to model pairwise relationships between a set of objects. Objects are typically represented by nodes, while the relations between them are then represented by edges. This basic framework enables us to model complex systems and capture complicated connections and dependencies between different elements.

Graph-based pattern representation overcomes major limitations of feature vectors. That is, graphs can capture not only features but also binary relationships and data arrangement. Moreover, they are not restricted to a fixed size, allowing adaptation to the size and complexity of different data objects. However, the use of graphs can lead to increased algorithmic complexity compared to feature vectors – actually one of the major challenges in graph-based pattern recognition. Nevertheless, the flexibility and expressive power of graphs make them an invaluable framework in pattern recognition in many fields, from social networks analysis [26] to transportation system modeling [67], and from biological network predictions [25] to recommendation systems [68].

2.2.3 *Basic Definitions on Graphs*

The present thesis is based on graph-based model representations[3], so graphs are the main data structure used in the remainder of this thesis.

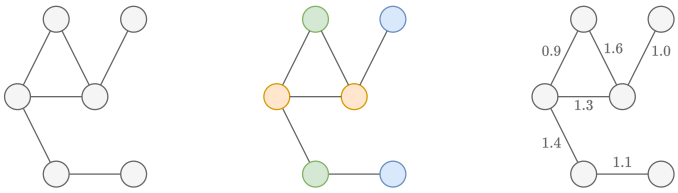
[3]Different fields, such as Mathematics, network analysis, and pattern recognition, work with graphs and often use similar terminology and definitions. In this work we adopt the nomenclature of pattern recognition, i.e., entities are called *nodes*, while relationships are called *edges*.

In this section, we provide some basic definitions on graphs that are important for the present research.

Definition 2.1 (Graph). *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,*
- *$\mu : V \to L_V$ is the node labeling function, and*
- *$\nu : E \to L_E$ is the edge labeling function.*

In this definition, sets L_V and L_E represent the labels for the nodes and edges, respectively. A node (and edge) labeling function $\mu : V \mapsto L_V$ (and $\nu : E \mapsto L_E$) is defined in the case of labeled nodes (and labeled edges). For example, the label alphabets L_V and L_E for both nodes and edges can be given by the set of integers $L = \{1, 2, 3, \dots\}$, the vector space $L = \mathbb{R}^n$, or a set of symbolic labels $L = \{\alpha, \beta, \gamma, \dots\}$. Three graph examples are displayed in Fig. 2.3.



(a) Unlabeled (b) Node labeled (c) Edge weighted

Fig. 2.3: Examples of different graph types (a) Unlabeled graph, (b) Node labeled graph where each color represents a different node label, (c) Edge weighted graph.

In some applications, it might be suitable to define *empty "nodes"* and/or *empty "edges"*, both denoted by ε. The size of a graph G is typically defined as the number of available nodes in G and is thus denoted by $|V|$. Edges are given by pairs of nodes $(v_i, v_j) \in V \times V$. If the graph is directed, an edge is defined as $(v_i, v_j) \in E$ with a start node $v_i \in V$ and an end node $v_j \in V$. Otherwise, an edge is defined as $(v_i, v_j) \in E \leftrightarrow (v_j, v_i) \in E$ in the case of undirected graphs. The neighbors of a node $v \in V$ are the nodes that are directly connected to v by exactly one edge and the neighborhood

of a node v is defined as $\mathcal{N}(v) = \{u \in V : (v, u) \in E\}$, where (v, u) is an edge in the undirected graph G.

In this thesis, we focus on *simple, undirected graphs*, that is, graphs with at most one edge between pairs of nodes and no self-loops (i.e., edges between a node and itself). Note, however, that the present research is in general applicable to any kind of graph (i.e., directed, undirected, labeled, unlabeled, etc.) For more information on the actual graph datasets used, see Chapter 3.

Another important concept to consider is that of a *subgraph*. A subgraph G' is derived from a graph G by selectively removing certain nodes with their connected edges, and potentially some additional edges from G.

Definition 2.2 (Subgraph). *Let's have two graphs $G = (V, E, \mu, \nu)$ and $G' = (V', E', \mu', \nu')$. Graph G' is a subgraph of G, denoted by $G' \subseteq G$, if*

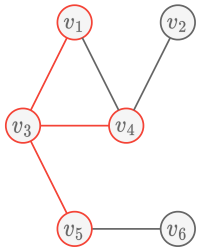
(1) $V' \subseteq V$,
(2) $E' \subseteq E$,
(3) $\mu'(u) = \mu(u)$ and $\nu'(e) = \nu(e)$ for all $u \in V'$ and for all $e \in E'$.

A *walk* w is any sequence of nodes $v_0, v_1, \ldots, v_k$ such that $(v_{i-1}, v_i) \in E$ for all $i = 1, \ldots, k$. The *length* of walk w corresponds to the number of edges in w. A walk w is termed *path* p if $v_i \neq v_j$ for all $i, j = 1, \ldots, k$ with $i \neq j$. That is, a path is a walk with distinct nodes. A *shortest-path* $p_{\min}(v_i, v_j)$ is the path of minimum length starting at node v_i and ending at node v_j. A graph $G = (V, E)$ is termed *connected*, if there exists at least one path connecting every pair of nodes $v_i, v_j \in V \times V$. Otherwise, that is if at least one pair of nodes $v_i, v_j \in V \times V$ exists that is not connected via a path, G is called *disconnected*. Visual examples of these graph concepts can be found in Fig. 2.4 (a) to (e).

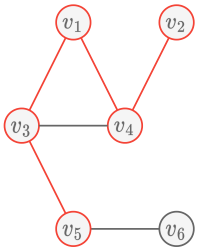
The structure of a graph $G = (V, E)$ is often encoded by means of the adjacency, degree, or Laplacian matrix. The *adjacency matrix* $\mathbf{A}$ of a graph $G = (V, E)$ with n nodes $V = \{v_1, \ldots, v_n\}$ is an $n \times n$ matrix $\mathbf{A} = [a_{ij}]$, where

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

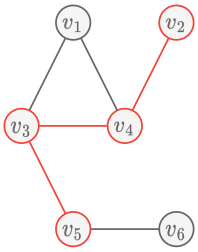
In the case of a simple, undirected graph, the adjacency matrix $\mathbf{A}$ is symmetric and the diagonal entries are zero. The *degree matrix* $\mathbf{D} = [d_{ij}]$ of the same graph $G = (V, E)$ is defined by



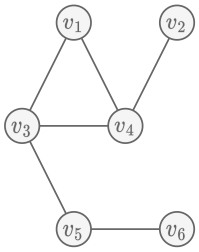
(a) Walk



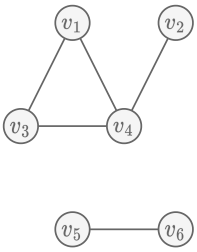
(b) Path $p(v_2, v_5)$



(c) Shortest-path $p_{\min}(v_2, v_5)$



(d) Connected graph G_1



(e) Disconnected graph G_2

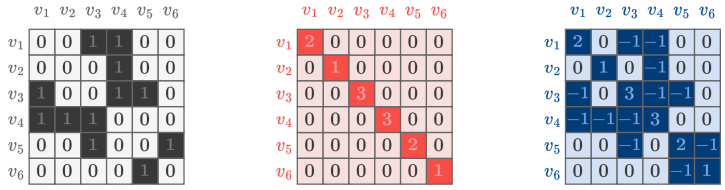
Fig. 2.4: Examples of different graph concepts: (a) Walk, (b) Path, (c) Shortest-path, (d) Connected graph, and (e) Disconnected graph.

$$d_{ij} = \begin{cases} \deg(v_i), & \text{if } i = j \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

The *Laplacian matrix* $\mathbf{L} = [l_{ij}]$ of the simple graph $G = (V, E)$ is defined element-wise as

$$l_{ij} = \begin{cases} \deg(v_i), & \text{if } i = j \\ -1, & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0, & \text{otherwise.} \end{cases} \tag{2.4}$$

In Fig. 2.5, we illustrate an example of the adjacency matrix $\mathbf{A}$, the degree matrix $\mathbf{D}$ and the Laplacian matrix $\mathbf{L}$ for the connected graph G_1 shown in Fig. 2.4 (d).



(a) Adjacency matrix $\mathbf{A}$ (b) Degree matrix $\mathbf{D}$ (c) Laplacian matrix $\mathbf{L}$

Fig. 2.5: Examples of the different matrix representations of the connected graph G_1 in Fig. 2.4 (d): (a) Adjacency matrix, (b) Degree matrix, and (c) Laplacian matrix

The *graph domain* $\mathcal{G} = \{G^{(1)}, \ldots, G^{(N)}\}$ is defined as a set of N graphs contained in a given dataset. Assuming that N i.i.d. classified training graphs $\mathcal{D} = \{G_i, y_i\} \subseteq (\mathcal{G} \times \mathcal{Y})$, where $\mathcal{G}$ represents the graph graph domain and $\mathcal{Y}$ the corresponding class label alphabet. The *graph classification task* consists of learning a model $f : \mathcal{G} \to \mathcal{Y}$ that assigns a class label $y \in \mathcal{Y}$ to any input graph $G \in \mathcal{G}$.

2.3 Graph Matching

When working with graphs, it is essential to be familiar with *graph matching*, especially when graphs have to be compared. Graph matching is the

problem of finding a correspondence between two graphs. This basically means finding a mapping between the nodes of two graphs that preserves – more or less stringent – the underlying edge structure. Hence, graph matching involves finding correspondences, or alignments, between the nodes and edges of two or more graphs. The concept of graph matching finds applications in different fields, such as chemistry [69], computer vision [24], and biology [25], where it is used to compare networks, recognize objects, or analyze molecular structures. Efficient algorithms and techniques [20] are used to meet the computational challenges of graph matching, making it an important framework for obtaining meaningful information from complex graph data.

Graph matching can be broadly categorized into two categories: *exact graph matching*, which requires an exact one-to-one correspondence between the nodes and edges, and *inexact graph matching* (also known as *error-tolerant graph matching*), which allows for some degree of tolerance in the correspondence found.

Several methods from inexact graph matching are relevant to this thesis. For the sake of completeness, however, we also provide a concise overview of three exact methods in Section 2.3.1. This includes *graph isomorphism*, *subgraph isomorphism*, as well as the concept of *maximum common subgraph*. Following this, Subsection 2.3.2 delves into inexact graph matching, providing general explanations of this concept. Last but not least, Section 2.4 presents three graph-based classification schemes that are based on three error-tolerant graph matching paradigms.

2.3.1 *Exact Graph Matching*

Exact graph matching involves determining whether two graphs, or parts of them, are identical in terms of node and edge arrangement. Yet, the way a graph is typically represented does not depend on a specific order of its nodes. Hence, it is not possible to determine whether two graphs, G and G' are equal simply by comparing their respective adjacency matrices, $\mathbf{A}$ and $\mathbf{A}'$. Furthermore, it is not often practical to examine all possible node arrangements in the adjacency matrix, as the number of possible permutations increases factorially. For instance, in a simple graph $G = (V, E)$ with $|V| = n$, there are $n!$ potential node permutations that would need to be checked.

The identity of two graphs G and G' is generally determined by finding a *graph isomorphism* [10; 49], which is a function that maps the nodes of

a source graph G to the nodes of the target graph G' in such a way that the adjacency relationship between the nodes is preserved.

Definition 2.3 (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* $f : V \to V'$ *satisfying*

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

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

Based on the definition of graph isomorphism, it is clear that two isomorphic graphs G and G' share the same structure and labels. The bijective mapping must preserve edge connections and ensure consistency in node and edge labels. A node v in graph G corresponds to $f(v)$ in G' only if their labels are the same, denoted as $\mu(v) = \mu'(f(v))$. This applies to edges as well, requiring that their labels remain identical after mapping, i.e., $\nu((v_i, v_j)) = \nu'(f(v_i, v_j))$. Furthermore, connected nodes $(v_i, v_j) \in E$ must have corresponding connected nodes $(f(v_i), f(v_j))$ in E'.

Graph isomorphism is a difficult problem, and there is no polynomial-time algorithm known [27]. In other words, there is no known way to quickly determine whether two graphs are identical in terms of structure and labels. Note that the graph isomorphism problem has not been demonstrated to belong to the class $\mathcal{P}$, and it stands as one of the most significant decision problems for which it remains unproven whether it falls into $\mathcal{P}$ or is $\mathcal{NP}$-complete [70]. Strong assumptions suggest that it may not be $\mathcal{NP}$-complete, implying that there is no efficient algorithm to solve it and checking if a given potential isomorphism is valid cannot be done efficiently either [71].

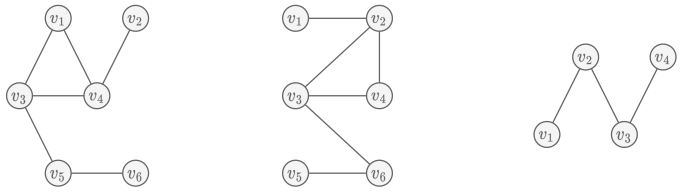
The uncertain computational complexity makes the graph isomorphism problem to a prominent open question in computational theory. However, in practical pattern recognition scenarios, specialized algorithms have been devised for certain types of graphs, offering manageable computation times. These algorithms apply to trees [66; 72], bounded-valence graphs [73], ordered graphs [74], planar graphs [75], permutation graphs [76], and graphs

with unique node labels [77; 78].

Subgraph isomorphism involves determining whether a given graph is identically contained within another graph in such a way that the adjacency relationship between the nodes of the smaller graph is preserved. In other words, a subgraph isomorphism is a one-to-one correspondence between the nodes of the first graph and the nodes of a subgraph of the other graph, such that every edge in the subgraph has a corresponding edge in the subgraph of the larger graph. Subgraph isomorphism is a more general problem than graph isomorphism, because it allows for the second graph to have additional nodes and edges.

Definition 2.4 (Subgraph Isomorphism). *Let $G = (V, E, \mu, \nu)$ and $G' = (V', E', \mu', \nu')$ be graphs. An injective function $f : V \to V'$ 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 $G \subseteq G'$.*

Subgraph isomorphism is computationally more complex than graph isomorphism. It involves not only verifying if a permutation of graph G matches graph G', but also determining if G can be matched to any subgraph of G' with a size equal to the number of nodes in G. Unlike graph isomorphism, subgraph isomorphism is classified as $\mathcal{NP}$-complete [27], signifying that it belongs to a set of computationally difficult problems for which no known polynomial-time solution exists.



(a) Graph G (b) Graph G' (c) Graph G''

Fig. 2.6: Examples of both graph and subgraph isomorphism.

In Fig. 2.6 (a) and (b), there are two graphs G and G'. We observe that G' is isomorphic to G, which means that there exists a bijective mapping between the nodes of G and G' that preserves the edges' structure.

Moreover, in Fig. 2.6 (c) there is a graph G'', which is deemed subgraph isomorphic to G. This means that G'' contains a set of nodes and edges that, when matched appropriately, can be superimposed onto G to form a subgraph within G that preserves structural relationships.

The *maximum common subgraph* (MCS) problem is the problem of finding the largest subgraph that is contained in both of two given graphs. This subgraph must have the same structure and labels as the corresponding subgraphs in the two original graphs. The MCS problem is a generalization of the graph isomorphism problem, which asks whether two graphs are identical in terms of structure and labels. The MCS problem is more general because it allows for two graphs to have different sizes. Mathematically, the MCS problem can be defined as follows.

Definition 2.5 (Maximum Common Subgraph). *Given two graphs $G = (V, E)$ and $G' = (V', E')$, the MCS problem is to find the largest graph $G_c = (V_c, E_c)$ that is subgraph isomorphic to both graphs G and G'.*

The MCS problem is known to be $\mathcal{NP}$-complete [27], which means that finding the optimal solution may require exponential time in the worst case. Therefore, heuristic and approximation algorithms are often employed in practice [79].

2.3.2 *Inexact (Error-tolerant) Graph Matching*

Inexact (or error-tolerant) involves finding similarities between graph G and G', even if they do not match exactly. This is in contrast to exact graph matching, which requires the two graphs to be identical in terms of both structure and labels. Inexact graph matching takes into account the fact that real data often contains errors in the translation from observation, noise, or variations that prevent an exact match [80]. This principle allows a degree of flexibility in graph matching and thus can often handle real-world scenarios better than exact matching paradigms.

Through the years many methods have been proposed to perform inexact graph matching. In this section, we briefly review two prominent families of inexact graph-matching techniques and we refer to [4; 5; 81] for more extensive reviews on different graph-matching methods.

Spectral methods [14; 82; 83] is a first prominent class of error-tolerant graph matching algorithms. These methods use the *spectral properties* of graphs to match them. Spectral properties of graphs are captured by the

eigenvalues and eigenvectors of the adjacency or Laplacian matrices. In [82], the authors use spectral properties of the underlying structural matrices to construct a vector space onto which the nodes of the graphs are embedded. This vector space is then used to find potential matches. In [83] another spectral graph matching algorithm is presented. This algorithm uses regularized quadratic relaxation. It starts by forming a similarity matrix from the spectral embeddings of the two graphs. This matrix then defines a regularized quadratic program, which is subsequently solved to achieve the graph matching. The authors of [14] introduce an error-tolerant graph matching that relies upon spectral features that encode a graph as a bag of partial node coverages.

A second prominent family of graph matching is *Continuous Graph Matching* [15; 16; 17]. This alternative approach transforms the graph matching problem, typically from a discrete optimization problem, into a continuous, nonlinear optimization problem. Basically, this is achieved by allowing the elements of a $|V| \times |V|$ permutation matrix $\mathbf{P} = (p_{ij})$ to take on continuous values between 0 and 1, rather than restricting them to discrete $\{0, 1\}$-values. Given the assumption of continuity, continuous optimization provides the opportunity to employ calculus-based techniques. This enables the use of continuous, nonlinear optimization techniques (e.g., gradient descent methods [84] or interior-point methods [85]).

In [15], for instance, a novel binary linear programming approach is introduced for exact Graph Edit Distance computation between graphs (see Section 2.4.1 for details on the concept of Graph Edit Distance). The introduced formulation is highly versatile and capable of handling both directed and undirected labeled graphs. Additionally, a continuous relaxation of domain constraints provides an efficient lower-bound approximation of Graph Edit Distance. The authors of [16] make the observation that in traditional graph matching, only one-to-one node correspondences are considered. Yet, in practical scenarios, perfect matches are often unattainable. It becomes more important to explore many-to-many correspondences, where groups of nodes in one graph correspond to clusters in the other. In [16] the many-to-many graph matching is formulated as a discrete optimization problem and they introduce an approximate algorithm based on a continuous relaxation approach to tackle the combinatorial nature of the problem.

In [17], the authors prove that an exact solution of the indefinite relaxation typically leads to the optimal permutation, whereas a standard convex relaxation tends to fall short. These findings imply that starting the indefinite algorithm with the convex optimum could enhance practical per-

formance. The authors experimentally confirm these theoretical insights, showcasing superior results across both benchmark and real-world datasets.

The present thesis is based on three important paradigms of inexact graph matching, namely Graph Edit Distance, Graph Kernel and Graph Neural Networks. These concepts are therefore explained in detail in the following three sections in conjunction with their corresponding classification scheme.

2.4 Graph Classifiers

2.4.1 *Distance-based Graph Classifier*

The specific definition of a dissimilarity (or vice versa a similarity) measure between two graphs depends on the problem at hand. A graph dissimilarity measure can be based on various graph properties, such as the number of common nodes, edges or subgraphs, as well as the node and edge labels, or the topological structure. A general dissimilarity measure can be defined by

$$d : \mathcal{G} \times \mathcal{G} \to \mathbb{R}^+, \tag{2.5}$$

such that $d(G, G')$ quantifies the dissimilarity between G and G'.

Dissimilarity measures for graphs are often defined upon a found graph matching. In the present thesis, we employ *Graph Edit Distance* (GED) [86; 87] as basic graph dissimilarity paradigm. GED was proposed in the early 1980s and can be interpreted as a standard dissimilarity measure for graphs. Its high degree of flexibility makes it easily applicable to a broad range of problems and GED has gained interest in a broad range of problems and applications [88; 89].

In contrast to other distance measures for graphs, GED provides more information than merely a dissimilarity score. In its formal definition, it computes an *edit path* which gives us the important interaction on how the substructures of the graphs actually match with each other (e.g., [90; 91]).

The basic idea behind GED is to find the minimum amount of *edit operations* required to transform graph G into graph G'. We represent the three fundamental edit operations, which are widely used (namely, *insertion*, *deletion*, and *substitution*), as follows:

- The substitution of two nodes $v \in V$ and $v' \in V'$ is denoted as $(v \to v')$.

- The deletion of node $v \in V$ is expressed as $(v \to \varepsilon)$.
- The insertion of node $v' \in V'$is denoted $(\varepsilon \to v')$.

A similar notation is used for the corresponding edge edit operations.

Definition 2.6 (Graph Edit Distance). *The graph edit distance between source graph $G = (V, E)$ and target graph $G' = (V', E')$ is defined by*

$$GED(G, G') = \min_{\lambda \in \Lambda(G,G')} \sum_{e_i \in \lambda} c(e_i), \tag{2.6}$$

where $\Lambda(G, G')$ denotes the set of all complete edit paths transforming G into G'.

An *edit path* $\lambda(G, G')$ between G and G' is a set $\{e_1, \ldots, e_k\}$ of k edit operations e_i that are necessary to convert a source graph G into a target graph G'. With $\Lambda(G, G')$ we denote the set of all edit paths transforming G into G'. A cost function $c(e_i)$ associated with each edit operation e_i is generally used to formalize the severity of operation e_i. GED can now be formally defined as follows.

The selection of edit costs $c(e_i)$ is a pivotal aspect of GED computations, and it heavily relies on the nature of the specific application. For instance, in the field of chemistry, where graphs represent molecular structures, the costs could be determined by the intricacy of chemical transformations. This might include operations like bond formation, cleavage, or the introduction of new functional groups [92]. By customizing the edit costs to suit the specific domain, GED becomes a powerful tool for accurately assessing dissimilarities between graphs in a meaningful and contextually relevant manner.

Exact solutions for the computation of graph edit distance $\mathrm{GED}(G, G')$ are often obtained with methods based on combinatorial search procedures that possibly check all matches of all nodes of G to all nodes of G'. In these formulations, GED searches the optimal edit path $\lambda_{\min} \in \Lambda(G, G')$ in the set of all admissible edit paths $\Lambda(G, G')$. Due to the exponential number of admissible edit paths, this type of computation of GED has an exponential computational complexity.

The complexity of graph edit distance optimization is actually known to be $\mathcal{NP}$-complete for general graphs [27]. This, in general, hinders its application to large-scale problems. However, in recent years several

approximate, or *suboptimal*, algorithms for GED problem have been proposed [28; 93; 94]. These algorithms offer polynomial, rather than exponential, run-times. Yet, in contrast to *optimal* algorithms for GED, suboptimal algorithms do not guarantee to find the global minimum of the GED, but only a local one.

In [11], a sub-optimal algorithm for GED (termed BP-GED) is proposed. This algorithm approximates GED by solving a linear sum assignment problem on graph nodes (including their local substructures). To this end, graphs are specifically formatted and the optimal assignment of local substructures is exploited for a fast approximation of GED. BP-GED is based on a fast (optimal) optimization method that maps the nodes and their local structure of one graph to the nodes and their local structure of another graph. The BP-GED algorithm has cubic time complexity and is a widely used method in the field of graph-based pattern recognition [95].

The traditional approach for distance-based graph classification is given by the *k-Nearest Neighbor* (k-NN) algorithm. The k-NN algorithm is a popular supervised machine learning technique that can be used for both classification and regression tasks [51]. It operates on the principle of proximity, where it predicts the target value of a given data point by considering the k-nearest data points in the training set (in our specific case, the data points are represented by means of graphs).

Formally, let us consider a supervised learning task with a training dataset $\mathcal{D}$ consisting of n graphs, where each graph is represented as $G_i \in \mathcal{G}$ with its corresponding target label $y_i \in \mathcal{Y}$. Given an unknown graph G, k-NN identifies the k graphs in the training set that are closest to G according to some chosen distance measure. These k data points are denoted as $N_k(G)$. For a classification task, the predicted class for G is determined by a majority vote among the classes of its k-nearest neighbors:

$$y = \arg\max_{c} \sum_{G_i \in N_k(G)} \mathbb{I}(y_i = c) \tag{2.7}$$

where $\mathbb{I}$ is the indicator function and c represents the class labels.

The choice of using the k-NN algorithm in graph-based pattern recognition is motivated by the fact that this particular algorithm relies directly on a distance function, which can readily be defined in any graph domain $\mathcal{G}$ (e.g., by means of GED in the context of this thesis).

2.4.2 *Kernel-based Graph Classifier*

Graph Kernels [6] constitute another prominent family of graph classification algorithms. Roughly speaking, a graph kernel is a measure of similarity between graphs that compares their underlying structures. Kernels are often used to construct matrices, known as kernel matrices, where the (i, j)-th entry represents the similarity between the i-th and j-th graphs. For these matrices to be valid, they must be positive semi-definite. Specifically, a kernel matrix $\mathbf{K}$ is positive semi-definite, if and only if, for any vector $v \in \mathbb{R}^n$ (where n is the number of graphs), the following inequality holds:

$$v^T \mathbf{K} v \geq 0. \tag{2.8}$$

This property is essential because it guarantees that the kernel matrix $\mathbf{K}$ encodes a valid notion of similarity between the graphs. Formally, a valid graph kernel is a symmetric, positive semi-definite function $\kappa : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ defined on the graph domain $\mathcal{G}$.

The vast majority of graph kernels proposed in the literature are instances of so-called *convolution kernels* [6]. Given two graphs G and G', the idea of the convolution framework is to decompose G and G' into substructures and evaluate a kernel between each pair of such substructures. Using a convolution operation, these similarities are then turned into a kernel function on the complete graphs. Prominent examples are, for instance, walk kernels [30], cycle kernels [96], or subgraph kernels [97], to name just three examples.

The power of the kernel framework (regardless of the graph kernel actually used) is based on the following observation. Given a graph kernel κ, there exists a function $\phi : \mathcal{G} \to \mathcal{H}$ mapping graphs from $\mathcal{G}$ to a Hilbert space $\mathcal{H}$ such that $\kappa(G, G') = \langle \phi(G), \phi(G') \rangle$ for all $G, G' \in \mathcal{G}$. Thus, a graph kernel computes an implicit embedding of the graphs in a Hilbert space $\mathcal{H}$. The shortcut for the computation of the dot product in an embedding space $\mathcal{H}$ is known as *kernel trick*. The impact and practical relevance of the kernel trick is large. In particular, any algorithm that can be reformulated entirely in terms of dot products can be directly accessed via any graph kernel. An abstract example of kernel trick usage is displayed in Fig. 2.7[4].

In the present thesis, we employ *Support Vector Machines* (SVMs) [98] in combination with graph kernels for graph classification. SVMs are popular supervised machine learning algorithms, which construct a hyperplane,

[4]Image adapted from [49]

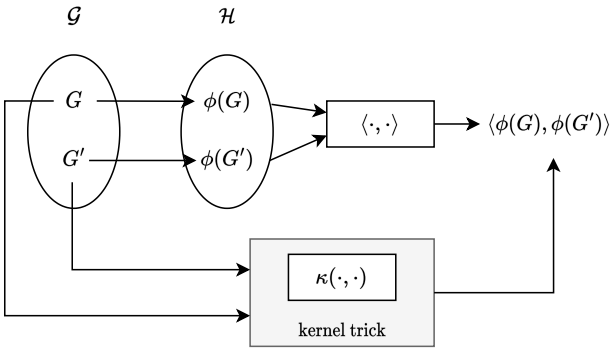


Fig. 2.7: Explicit comparison between mapping G and G' in a feature space $\mathcal{H}$ using ϕ and the subsequent dot product induced by the shortcut kernel trick.

or set of hyperplanes, in a high dimensional space, which can be used for both regression or classification. The use of a graph kernel in conjunction with SVM is motivated by the fact that with SVMs both training and classification can be entirely reformulated in terms of pairwise dot products. Hence, the kernel trick is fully applicable and SVMs are able to handle non-vectorial data by implicitly mapping the data into a high-dimensional feature space $\mathcal{H}$.

In the present thesis, we make use of two widely applied graph kernels, viz. the *Shortest-Path kernel* [19] and the *Weisfeiler-Lehman kernel* [20], which are briefly reviewed next.

Shortest-Path Kernel [19] consists of deriving a kernel $\kappa_{\mathrm{SP}} : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ based on both attributes and length of the shortest paths between pairs of nodes $(v_i, v_j) \in V \times V$ in both graphs to be compared.

Formally, in order to compute $\kappa_{\mathrm{SP}}(G, G')$ one first applies the so-called *Floyd-transformation* [99] (see Alg. 1) on the underlying graphs $G = (V, E)$ and $G' = (V', E')$ to obtain the corresponding *shortest-path graphs* $S = (V, E_S)$ and $S' = (V', E'_S)$, respectively. The graphs S and S' are weighted versions of G and G' such that they share the same node sets V and V'. Yet, in contrast to G, there is a weighted edge $(v_i, v_j) \in E_S$, if and only if, there exists a path in G, that actually connects nodes v_i and v_j. The weight w_{ij} of edge $(v_i, v_j) \in E_S$ corresponds to the length of the shortest path between nodes v_i and v_j. The same accounts for graph $G' = (V', E')$ and $S' = (V', E'_S)$, respectively.

Algorithm 1: Floyd-Warshall

Input: Adjacency matrix $\mathbf{A} \in \mathbb{R}^{|V|\times|V|}$ of graph $G = (V, E)$

Initialize D as empty 2-dimensional array
for $i = 1, \ldots, |V|$ **do**
  for $j = 1, \ldots, |V|$ **do**
    if $\mathbf{A}[i, j] == 1$ *and* $i \neq j$ **then**
      $D[i, j] = \nu((v_i, v_j))$
    else
      if $i == j$ **then**
        $D[i, j] = 0$
      else
        $D[i, j] = \inf$
      end
    end
  end
end
for $k = 1, \ldots, |V|$ **do**
  for $i = 1, \ldots, |V|$ **do**
    for $j = 1, \ldots, |V|$ **do**
      if $D[i, k] + D[k, j] < D[i, j]$ **then**
        $D[i, j] = D[i, k] + D[k, j]$
      end
    end
  end
end

Output: Return D as E_S

Given two graphs G and G' and their corresponding Floyd-transformed graphs S and S', respectively, the shortest-path kernel is then defined by

$$\kappa_{\mathrm{SP}}(G, G') = \sum_{e \in E_S} \sum_{e' \in E'_s} \kappa_{path}(e, e'), \tag{2.9}$$

where κ_{path} is an edge path kernel defined as follows. Let us assume two edges $e = (v_i, v_j) \in E_S$ and $e' = (v'_i, v'_j) \in E'_S$ are given, then $\kappa_{path}(e, e')$ is defined as the product of node and edge kernels

$$\kappa_{path}(e, e') = \kappa_{node}(v_i, v'_i) \cdot \kappa_{node}(v_j, v'_j) \cdot \kappa_{edge}(e, e').$$

Here $\kappa_{node}(\cdot)$ is a kernel for quantifying the node label similarity and $\kappa_{edge}(\cdot)$ is a kernel to measure the similarity of the shortest-path distances assigned to the edges, such that $\kappa_{edge}((v_i, v_j), (v'_i, v'_j)) = 0$ if $d(v_i, v_j) = \infty$ and/or $d(v'_i, v'_j) = \infty$

The complexity of the shortest-path graph kernel depends on the algorithm used to compute the shortest-path between all pairs of nodes in G. A common algorithm for this task is the Floyd-Warshall algorithm (see Alg. 1), which has a time complexity of $\mathcal{O}(|V|^3)$, where $|V|$ is the number of nodes in graph G. This is because in the Floyd-Warshall algorithm the distance matrix d is updated $|V|$ times, and each update requires $\mathcal{O}(|V|^2)$ operations. Another algorithm that can be used to compute the shortest-path graph kernel is Dijkstra's algorithm. Dijkstra's algorithm has a time complexity of $\mathcal{O}(|V| + |E|log(|V|))$ and is typically faster than the Floyd-Warshall algorithm for sparse graphs, but slower for dense graphs. In practice, the shortest-path graph kernel can be computed efficiently for graphs of moderate size.

Weisfeiler-Lehman Kernel [20] is a popular graph kernel with efficient classification performance. This kernel works on top of a well-known graph isomorphism algorithm, namely the Weisfeiler-Lehman graph isomorphism test. This test consists of an iterative method that produces a canonical form for each graph. In each iteration, the current feature label l of a given node v_i is aggregated with the labels of all adjacent nodes and replaced with a new compressed label. The Weisfeiler-Lehman test performs h iterations over the graph. If the canonical forms of the underlying graphs do not match after h iterations, then the two graphs in question can not be isomorphic. However, if the canonical forms do match, it is not yet clear whether or not the two graphs are isomorphic as two non-isomorphic graphs can end up with the same canonical form during the Weisfeiler-Lehman procedure.

The basic idea of the Weisfeiler-Lehman graph kernel is to compute the above-described procedure $h \geq 0$ times. In each iteration i, one computes a feature vector $\phi^i(G)$ for each graph G, which is formally defined as

$$\phi^i(G) = (c_i(G, \sigma_{i1}), \ldots, c_i(G, \sigma_{i|\Sigma_i|})). \tag{2.10}$$

Here, Σ_i is the set of aggregated node features of G and G' and $c_i : \mathcal{G} \times \Sigma_i \to \mathbb{N}$ corresponds to the number of occurrences of the updated label l at iteration i. The h-Weisfeiler-Lehman kernel is then formally defined as

$$\kappa_{h\text{-WL}}(G, G') = \langle \phi_{h\text{-WL}}(G), \phi_{h\text{-WL}}(G') \rangle, \tag{2.11}$$

where $\phi_{h\text{-WL}}(G)$ is the sequence of feature vectors of all h iterations, i.e., $\phi_{h\text{-WL}}(G) = (\phi^0(G), \phi^1(G), \ldots, \phi^h(G))$, and $\langle \cdot, \cdot \rangle$ denotes the standard dot product.

The complexity of the Weisfeiler-Lehman graph kernel can be broken down into three steps, which are repeated h times.

(1) The *sorting* step, where each node v is represented as a list L_v of its neighbors, has a complexity of $\mathcal{O}(|E|)$.
(2) During the *compression* step, each list L_v is compressed into a hash value (also complexity of $\mathcal{O}(|E|)$).
(3) Finally, the *relabeling* step, relabel the node v with the previously computed hash value as its new node label. The complexity of this step is $\mathcal{O}(|V|)$.

Thus, we can observe that for a given pair[5] of graphs the runtime complexity is $\mathcal{O}(|E|h)$.

2.4.3 *Neural Network-based Graph Classifier*

Graph Neural Networks (GNNs) [8; 9; 100] are a type of deep learning method that is specifically designed to work with graph-based data. At a high level, GNNs learn representations for each node in a graph based on their local neighborhood structure. This is done by exchanging information between neighboring nodes in the graph in a way similar to dissemination of information in a social network. This allows GNNs to capture important local patterns and relationships in the graph, while also leveraging the overall structure of the graph to make predictions.

The typical architecture of a GNN consists of several layers of computation, each of which typically contains a message-passing mechanism, aggregation functions, and learnable parameters. In a GNN, the representation of a node v is updated on the basis of information collected from its neighbors. In the message-passing step, each node in the graph sends a message to its neighboring nodes, based on its current feature representation. These messages are then aggregated and transformed into a new representation for each node. This is done using the following equation:

$$h_v^{(t+1)} = \text{UPDATE}\left(h_v^{(t)}, \text{AGGREGATE}\left(\{h_u^{(t)} : u \in \mathcal{N}(v)\}\right)\right) \tag{2.12}$$

[5] Remark that, in general, $|V| \ll |E|$

Here, $h_v^{(t)}$ represents the representation of node v in the t-th layer. $\mathcal{N}(v)$ denotes the set of neighbors of node v. The function AGGREGATE combines the information from the neighbors. The function UPDATE combines the current node representation with the received aggregated messages.

The AGGREGATE function can be any type of function that is independent of the order it receives the information from the neighboring nodes. The AGGREGATE function is typically a sum, a mean, or a max function. However, other methods involving more advanced functions were also proposed, e.g., DiffPool [101] or SortPool [102], to name just two examples.

In the node update step, each node combines its new representation with its old representation, using a neural network to compute a new feature vector. The specific form of the update function depends on the architecture and design of the GNN. It typically involves a learnable transformation that combines the node's current representation with the messages received. This transformation may include parameters such as weights and biases, which are learned during the learning process. This new feature vector is then passed on to the next GNN layer or used to make predictions.

There are different variations of GNNs available (for a thorough review of GNN models we refer to [9]). The *Graph Convolutional Network* (GCN) [21] is a widely used baseline model in GNNs. It incorporates the symmetric-normalized aggregation technique along with the self-loop update strategy. These choices in aggregation and update mechanisms play a critical role in the model's ability to effectively propagate information across the nodes of the graph, which ultimately contributes to its performance in tasks involving graph data.

Graph Attention Network (GAT) is another variant of a GNN [22]. GAT introduces a novel neural architecture for processing graph-structured data. GAT use masked self-attentional layers, overcoming the limitations of previous methods based on graph convolutions. This allows nodes to assign varying weights to their neighbors' features without expensive operations or prior knowledge of the graph structure.

In [103], *Graph WaveNet* is introduced as a novel graph neural network architecture designed for spatial-temporal graph modeling. It incorporates an adaptive dependency matrix, learned through node embedding, to accurately capture hidden spatial dependencies. The model also employs a stacked dilated 1D convolution component with an exponentially expanding receptive field to handle long sequences.

The authors of [104] propose a graph transformer with two specific fea-

tures. First, an attention mechanism based on neighborhood connectivity and second, positional encoding using Laplacian eigenvectors. Additionally, they proposed to substitute the layer normalization with batch normalization for enhanced training speed and generalization, and extend the architecture to incorporate edge feature representation.

In [105], a graph similarity metric is directly computed by means of a GNN. It uses a message-passing neural network to capture the graph structure and employs a siamese network approach to learn the similarity metric.

In the present thesis, we make use of the *Deep Graph Convolutional Neural Network* (DGCNN) [102]. This architecture consists of three consecutive stages. First, graph convolutional layers are used to extract local substructure features of the nodes and establish a consistent node ordering. Second, a SortPooling layer arranges the node features in the established order and standardizes input sizes. Third, traditional convolutional and dense layers are utilized to process the sorted graph representations and generate the final classification. The time complexity of this specific GNN is $\mathcal{O}(|E|)$, so it depends on the sparsity of graphs involved in the training process [9].

2.5 Graph Reduction

The expansion of computing resources has enabled the generation of vast amounts of data. This in turn evoked the need for efficient and reliable methods to summarize and simplify data in order to extract meaningful insights from it. Although data reduction methods on statistical data have been extensively studied, it is only recently that researchers started to focus on structural data reduction [32], i.e., reduction of graphs.

Graph reduction methods offer many benefits. First, by generating smaller summaries through graph reduction methods, the storage space required can be significantly reduced compared to the original graphs. This is especially useful when dealing with large-scale graphs (e.g., graphs representing social media networks with billions of users). Second, graph reduction methods can help filter out this noise and retain only the "essential" information. This is also highly beneficial as the underlying data often contains noise from collection and transcription errors, including erroneous nodes and edges hidden from the human eye. In addition, graph reduction methods can help to speed up graph matching algorithms, thereby

addressing one of the main problems described in Section 2.4, namely high computation cost. That is, graph reduction methods typically generate smaller graphs while preserving relevant information, which supports fast and efficient analysis by means of graph matching algorithms.

The concept of graph reduction lacks a widely accepted definition as it is, in general, context-dependent and can be defined differently depending on the actual goal. One can define graph reduction, for instance, to preserve specific structural patterns or to focus on maintaining given distributions of graph properties (e.g., the degree distribution or the diameter to name two examples).

This lack of a generally applicable definition poses three major challenges. First, the inherent structural complexity of graph data makes it often complex to partition and parallelize operations due to many interactions between nodes. Second, since graph reduction method aim to extract *interesting* information, determining what is considered interesting is subjective and requires expert knowledge and user preferences. Moreover, the distinction between interesting and uninteresting information is difficult to make in practice and often involves a trade-off between time, space, and preserved information in the reduced graph. Third, since graph reduction methods depend on the specifics of the problem, there is no *universal* way to evaluate the effectiveness of graph reduction methods. For instance, in a database context, a reduction method may be considered successful if the data retrieved from queries are highly accurate. In the context of graph matching, however, the effectiveness of the reduction method would be better assessed based on the similarity of graph distances obtained from the reduced graphs compared to those obtained from the original graphs.

This particular lack of a universal benchmark emphasizes the need to carefully consider the context and specific goals when evaluating the effectiveness of graph reduction techniques. In this thesis, our main goal is to reduce the size of the graphs in order to speed up standard graph matching algorithms. Consequently, our goal is to ensure that these algorithms maintain the same level of accuracy whether they are applied on the reduced or original graphs. Hence, we define the classification accuracy obtained using reduced graphs as one of the main criteria of our evaluations (alongside the runtime). That is, the classification task is used as a proxy-task for assessing the effectiveness of the proposed graph reduction methods.

The basic hypothesis underlying this approach is that if graph matching algorithms achieve comparable (or even better) classification accuracies on reduced graphs compared to the one achieved on the original graphs, then

the reduction methods is considered successful and can leverage the "most important" information out of the original graphs.

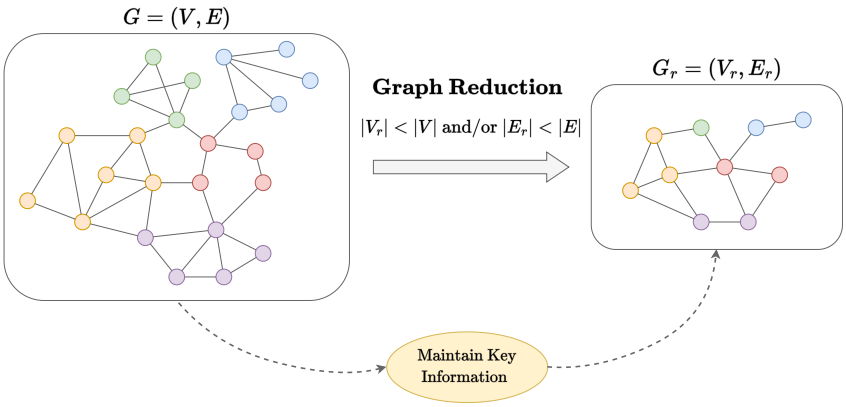


Fig. 2.8: General framework for graph reduction[6]

Despite that no universal definition of graph reduction is available, we choose to base our research on the following generic definition for graph reduction. Graph reduction is defined as the process of finding a graph of smaller size while preserving key information. Formally, given a graph $G = (V, E)$, the goal is to find a smaller graph $G_r = (V_r, E_r)$ with $|V_r|$ nodes and $|E_r|$ edges, where $|V_r| < |V|$ and/or $|E_r| < |E|$, which is a good approximation of G in some sense [33]. In other words, this definition requires a reduction algorithm that takes a graph as input and produces a smaller graph in terms of nodes and/or edges (see Fig. 2.8). The reduced graph domain $\mathcal{G}_r = \{G_r^{(1)}, \ldots, G_r^{(N)}\}$ is obtained from the original graph domain $\mathcal{G}$ by reducing all graphs $G \in \mathcal{G}$ according to the defined reduction method.

The following three subsections describe the three main graph reduction strategies used throughout this thesis, viz. *graph summarization*, *graph coarsening*, and *hierarchical graph reduction*. Graph summarization methods (described in Subsection 2.5.1) reduce the graphs by sampling the most influential nodes. Graph coarsening methods (described in Subsection 2.5.2) reduce the graphs by aggregating nodes together into super-nodes. Hierarchical graph reduction methods (described in Subsec-

[6]Image adapted from [34]

tion 2.5.3) leverage the other two reduction strategies to create a pyramid of reduced graphs at different levels of reduction.

2.5.1 *Graph Summarization*

A prominent graph reduction strategy is *graph summarization* [32]. Graph summarization eases the discovery of complex patterns in structural data and is employed in a wide range of applications, such as *community detection* [35], *classification* [36], and *visualization* [37].

Graph summarization is based on a graph approximation that retains only a subset of the most important nodes and edges based on specific criteria. The selection of the most relevant nodes and/or edges in the graph structure is accomplished by first quantifying the importance or relevance of each node and/or edge within the graph. Eventually, the nodes and/or edges with the least importance or relevance are omitted leading to a reduced graph via a sampling strategy. We present an illustrative example for graph summarization in Fig. 2.9.

Graph summarization methods include, for instance, methods that sample nodes based on their in- or out- degree or spanning tree substructures, as well as methods that sample edges based on their weights or their effective resistance [106]. Additionally, those methods aim to maintain cuts and the graph spectrum up to some multiplicative error [107] or the node reachability [108]. In [109], for instance, the graph spectrum of reduced graphs is approximated to match the spectrum of the original graphs by first extracting sparse subgraphs via a spanning tree algorithm and then iteratively retrieving some edges that do not appear in the tree.

Another approach to graph summarization is based on *bit-compressions* [32]. This approach aims to minimize the number of bits required to represent the input graph through its reduced version. Some bit-compression based methods are lossless, allowing perfect reconstruction of the original graph from its reduced version, For instance, the authors of [110] demonstrate that web graphs are compressible down to almost two bits per edge. Other compression methods are lossy compromising some details in the reconstruction process to save more space. For instance, in [111], a lossy graph compression method is introduced with the aim to preserve the communities in social networks.

Note that, in general graph reduction methods aim at obtaining sparse subgraphs that approximate the characteristics of the original graphs (e.g., the distribution of the connected components size or community structure)

and not on identifying patterns that summarize the input graph to improve user understanding [32]. While sampling nodes and/or edges can approximate certain graph properties with theoretical guarantees [112], it has the major limitation that it cannot detect complex graph structures and often focuses on individual nodes and/or edges rather than on collective patterns.

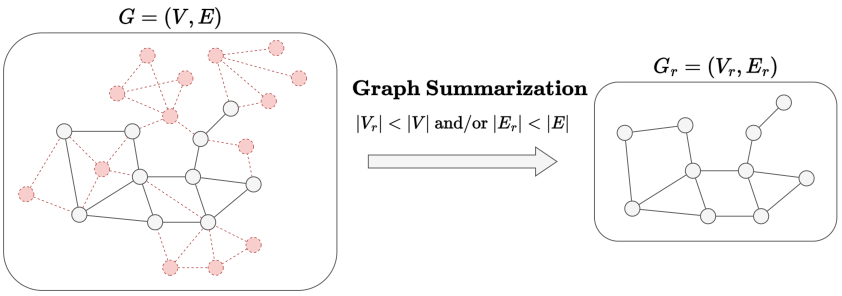


Fig. 2.9: Illustrative example of graph summarization: In the left part, nodes with the least score are highlighted in red. In the right part, nodes with the least score are omitted in the reduced version.

2.5.2 *Graph Coarsening*

The selection of nodes and/or edges that are eventually deleted, as proposed in graph summarization, ultimately leads to a loss of some information. Graph coarsening methods are an alternative to graph summarization that aim at maintaining as much information as possible during the graph reduction process. Graph coarsening involves grouping and aggregating nodes and/or edges into *super-nodes* and/or *super-edges*. Different approaches are possible to group and aggregate nodes and/or edges, namely *node clustering methods* and *node aggregation methods*. An illustrative example of graph coarsening is presented in Fig. 2.10.

For instance, node clustering methods use graph clustering algorithms for aggregation. Graph clustering methods usually aim to find a grouping of the nodes into clusters such that the number of cross-cluster edges is minimized. A reduced version of the input graph can then be obtained by mapping all the nodes that belong to the same cluster into a super-node and connecting them with super-edges. The weight of those super-edges can be determined by the sum of the cross-cluster edges. For instance, in [113], a graph coarsening method is proposed where clusters with high intra-cluster

edge densities and relatively low inter-cluster edge densities are first found and then grouped into super-nodes. The authors of [114] introduce SNAP, a graph coarsening method that is used to analyze social media networks. SNAP produces a reduced graph where every node inside a super-node has the same values for selected attributes and is adjacent with similar selected relation. In [33], spectral coarsening methods are reviewed and evaluated in order to speed up graph neural network training.

Node aggregation methods use a recursive process of aggregating nodes into super-nodes connected with super-edges. The aggregation is typically based on an optimization of the specific problem at hand. In [115], for instance, biological graphs are coarsened into smaller and more comprehensible versions of the graphs, where the nodes represent entire patterns of the original graphs. The authors of [38] use graph coarsening methods in electrical networks to obtain lower dimensional, yet electrically equivalent, circuits. In [116], graphs are reduced by merging nodes with similar relationships to minimize the approximation error of edge weights and to maximize the compression.

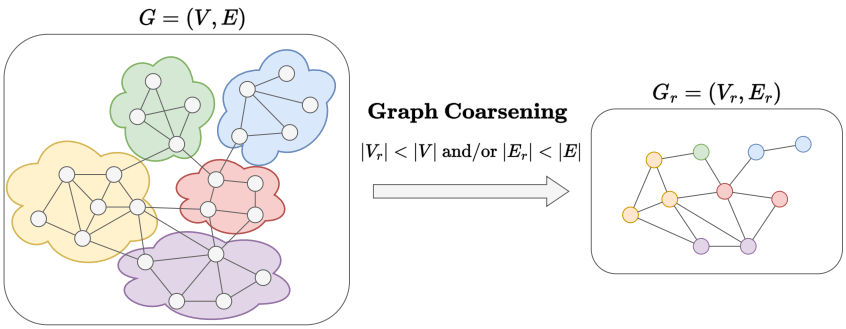


Fig. 2.10: Illustrative example of graph coarsening. In the left part, the different cluster of nodes are highlighted in different colors. In the right part, the nodes belonging to the same cluster are aggregated into super-nodes.

2.5.3 *Hierarchical Graph Reduction*

A third prominent graph reduction approach is *hierarchical graph representation* [39]. This approach can be used with both of the previously reviewed

graph summarization and graph coarsening methods.

The basic idea of hierarchical graph representations is to progressively reduce the original graphs as the number of reduction levels increases. The key concept of hierarchical graph representation methods is to construct a pyramid of subgraphs and then use the subgraphs at the highest level to perform the graph matching. In this way, only the most abstract versions of the graphs remain at the top level, simplifying the processing and analysis of complex structural data. We present an illustrative example of hierarchical graph representation in Fig. 2.11

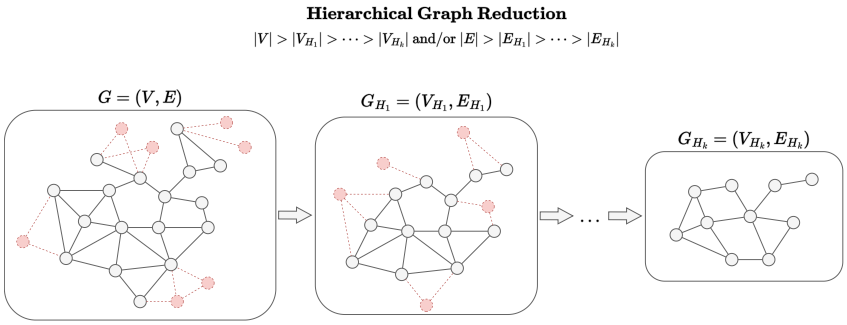


Fig. 2.11: Illustrative example of hierarchical graph representation, where the graph G is progressively reduced as the level of reduction H_i increases $(i = 1, \ldots, k)$.

Originally, hierarchical graph representations are proposed in the field of Computer Vision [117; 118]. The basic idea is to represent an image in a multiresolution pyramid where level 0 is the original image and the upper levels are aggregations of the pixels of the previous levels. In the multilevel representation, each level represents different semantic properties like texture or color [117]. This representation can then be used, for instance, to find boundaries between regions in an image. Following that idea, in [119], the authors use hierarchical representations in conjunction with graph data structures. The idea is to create a hierarchy based on the community compression of the previous levels.

In [120], for instance, the authors use the Fiedler vector to decompose a graph in a stable way. Based on this decomposition, a hierarchical graph simplification process is proposed in which each neighborhood is represented

by a node and connected by edges if there is at least one edge between their common neighborhood. In [40; 41], hierarchical representation for graphs is used in a pattern recognition context. The authors propose to embed graphs into a vector space and use a community detection method to find the nodes to merge. Thereby, they create a representation that encodes the abstract information while preserving the relationship with the initial graph. In [121], a coarse-to-fine graph matching strategy is proposed. The idea is to represent graphs as a pyramid where the lower levels contain the coarse information of the graphs and the upper levels contain more fine information. Then the authors perform the graph comparison in a coarse-to-fine fashion.

Graph Datasets

3

> Quand les choses sont connues, ne dirait-on pas qu'elles ne sont que mieux cachées.
>
> *La commune* (1898), Louise Michel

According to recent studies, the machine learning and pattern recognition community has expressed concerns about several methodological issues in research. These include, for example, the replicability crisis, whereby previously published results cannot be replicated [122]. Other problems include biased results [123] and models that cannot handle real-world scenarios due to too small datasets or a lack of diversity [124].

In summary, poor research practices, such as unclear experimental setups, non-reproducible results, and inappropriate model comparisons, prevent consistent evaluation of machine learning and pattern recognition methods and require concerted efforts to prevent their use.

To address the challenge of inadequate model comparisons, the use of standardized datasets with rigorous experimental designs is a common approach [125]. In the present thesis, we apply this strategy using different graph datasets from various domains for graph classification tasks.

The present chapter consists of two major parts. First, we present a comprehensive list of 28 graph datasets, including both node-labeled and unlabeled datasets. For each dataset, we detail its source along a visual plot of the graph for each class and with some key metrics. These metrics are organized as tables, and they include the number of graphs per dataset, the number of classes, the proportion of each class, the minimum, average, and maximum number of nodes and edges per graph, as well as the average node degree and edge density per graph. The datasets presented are pulled from a diverse range of domains such as chemistry (Section 3.1), bioinformatics (Section 3.2), computer vision (Section 3.3), and social network analysis

(Section 3.4).

The second part of the chapter consists of an analysis of the three graph-based pattern recognition techniques, presented in Section 2.4, which are *Graph Edit Distance*, *Graph Kernel*, and *Graph Neural Network* in conjunction with standard classifiers (such as *k-NN*, *SVM* or *Neural Networks*), which in turn are widely used in academic and applied research.

In particular, the experiments aim to determine which information is most important in the graph structure i.e., structure, labels, or both. For this purpose, we test the three classification paradigms in three different configurations:

- A classifier that has access to the original graphs
- A classifiers that has access to the original graphs without labels
- A classifier that has access to a feature vector that aggregates all labels of the graphs

The third configuration can be seen as a naïve baseline, which should serve as a reference system whenever a novel graph-based method is proposed. For instance, if a graph dataset yields inferior results with a graph-based method compared to the naïve vector-based embedding, it implies that applying any graph reduction method will likely result in suboptimal results.

We are aware that this is not the first attempt to identify when, why and which graph-based methods are most effective. For instance, in [6], the authors provide a thorough review of the most common graph kernels, and in [126] it is shown that the state-of-the-art graph kernels can underperform compared to simpler methods. Furthermore, the authors of [125] present a comprehensive comparison of six graph neural network architectures.

Our analysis differs from previous work in two key points. First, to the best of our knowledge, we are the first to compare each of the three popular graph-based classifiers with two systems that operate in a similar (or identical) way, but with access only to the graph structure or the node labels. Second, we complement our work by explicitly listing graph datasets that can be used to report the results of (approximate) graph-based methods (or, put negatively, we point out which graph datasets should not be used for such research purposes). That is the list of graph datasets that are the most valuable for us to use to assess our graph reduction methods.

3.1 Chemical Compound Graph Datasets

Chemical compounds, the building blocks of matter, consist of atoms bonded together [127]. That is a chemical compound is made up of different types of atoms. Moreover, they have distinct structures held together by chemical bonds, which can be covalent, ionic, or metallic.

The description of the molecular compounds can be effectively achieved through graph representations. In this context, a graph uses nodes to represent atoms and edges to represent the covalent bonds that connect them. Nodes in such graph are typically labeled with the specific chemical symbols corresponding to the atoms they represent. The edges representing the bonds between atoms can be labeled with information about the nature of those bonds, such as their valence. An example of a molecule represented as a graph can be found in Fig. 3.1.

This section introduces 16 molecule graph datasets from the TUDataset repository [128] used in the remainder of this thesis. These datasets are derived from real chemical compounds, and they serve as the foundation for exploring the relationship between molecular structure and biological activity. We present the following 16 graph datasets:

- AIDS (in Subsection 3.1.1)
- BZR and BZR-MD (in Subsection 3.1.2)
- COX2 and COX2-MD (in Subsection 3.1.3)
- DHFR and DHFR-M (in Subsection 3.1.4)
- ER-MD (in Subsection 3.1.5)
- MUTAG (in Subsection 3.1.6)
- MUTAGENICITY (in Subsection. 3.1.7)
- NCI1 and NCI109 (in Subsection 3.1.8)
- PTC-MM, PTC-FM, PTC-MR, and PTC-FR (in Subsection 3.1.9)

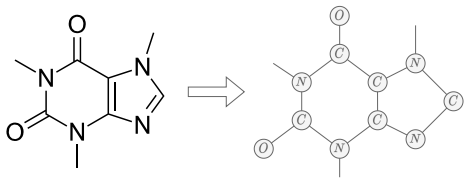


Fig. 3.1: Example of a molecular compound represented as a graph.

3.1.1 *AIDS*

The *AIDS* dataset is sourced from the National Cancer Institute (NCI), specifically from the AIDS Antiviral Screen Database of Active Compounds [129]. It was first proposed in the IAM Graph Repository [130] as a graph dataset and contains 2,000 graphs. The graphs in the AIDS dataset represent the molecular structure of chemical compounds. The nodes in the graphs represent atoms, and the edges represent covalent bonds between atoms. The node labels indicate the *atom type*. Edges are unlabeled.

AIDS	
Num Graphs $\lvert\mathcal{G}\rvert$	2,000
Num Classes $\lvert\Omega\rvert$	2
Class proportion	20.0% - 80.0%
Min. - Avg. - Max. $\lvert V\rvert$	2 - 15.7 - 95
Min. - Avg. - Max. $\lvert E\rvert$	1 - 16.2 - 103
Avg. node degree	2.01
Avg. edge density	0.19

Table 3.1: Summary of some metrics of the AIDS dataset.

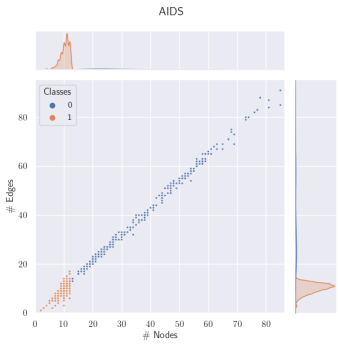
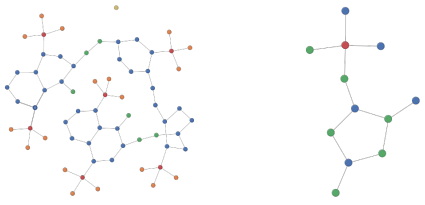


Fig. 3.2: Distribution of graphs by number of nodes and edges

The main purpose of this dataset is a classification task, viz. determine whether a given compound is active or inactive against the HI virus. Additionally, Fig. 3.3 showcases one compound from each class, with distinct colors representing different chemical symbols.



(a) Class '0' (b) Class '1'

Fig. 3.3: Example graph for both classes of the AIDS dataset.

3.1.2 *BZR & BZR-MD*

Benzodiazepine receptor (BZR) [131] is a node labeled graph dataset that contains 405 distinct chemical compounds. Each individual graph in this dataset represent a brain receptor molecule (BZR), where atoms are represented by nodes and covalent bonds are represented by edges. Nodes are labeled by the *atom type*, while edges are unlabeled.

BZR	
Num Graphs $\|\mathcal{G}\|$	405
Num Classes $\|\Omega\|$	2
Class proportion	79.0% - 21.0%
Min. - Avg. - Max. $\|V\|$	13 - 35.8 - 57
Min. - Avg. - Max. $\|E\|$	13 - 38.4 - 60
Avg. node degree	2.15
Avg. edge density	0.06

Table 3.2: Summary of some metrics of the BZR dataset.

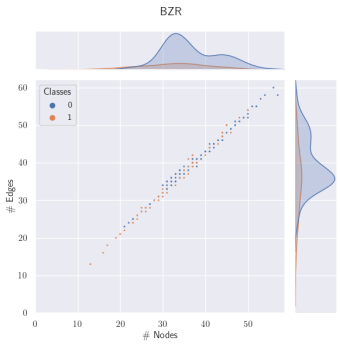
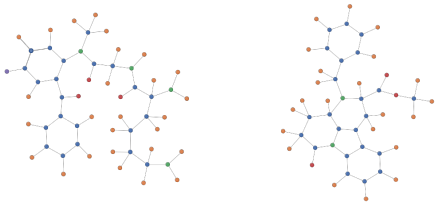


Fig. 3.4: Distribution of graphs by number of nodes and edges

The BZR dataset has been designed as a classification task that consists of distinguishing compounds that are active on the Benzodiazepine receptor from those that are not reactive. Table 3.2 contains important metrics for the BZR dataset, while Fig. 3.4 shows the distribution of graphs based on their number of nodes and edges. Fig. 3.5 represents a compound from each class, with differently colored nodes for different chemical symbols.



(a) Class '0' (b) Class '1'

Fig. 3.5: Example graph for both classes of the BZR dataset.

Benzodiazepine receptor - Modified (BZR-MD) [97; 131] graph dataset is a modified version of the BZR dataset. The number of graphs is reduced to obtain a balanced dataset (i.e., there is the same number of graphs in each class). Moreover the edge structure of all graphs is modified to obtain complete graphs (see, for instance, Fig. 3.7 where we observe that graphs from both classes are complete). A summary of the different graph metrics about BZR-MD as well as a distribution of the graphs by the number of nodes and edges can be found in Table 3.3 and Fig. 3.6, respectively.

BZR-MD	
Num Graphs $\|\mathcal{G}\|$	306
Num Classes $\|\Omega\|$	2
Class proportion	49.0% - 51.0%
Min. - Avg. - Max. $\|V\|$	8 - 21.3 - 33
Min. - Avg. - Max. $\|E\|$	28 - 225.1 - 528
Avg. node degree	20.30
Avg. edge density	1.00

Table 3.3: Summary of some metrics of the BZR-MD dataset.

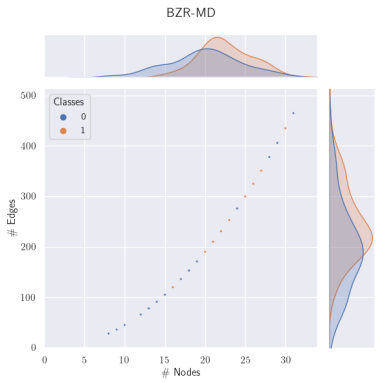


Fig. 3.6: Distribution of graphs by number of nodes and edges

(a) Class '0' (b) Class '1'

Fig. 3.7: Example graph for both classes of the BZR-MD dataset.

3.1.3 *COX2 & COX2-MD*

The *Cyclooxygenase-2* (COX2) dataset is originally proposed in [131] and represents a collection of 467 graphs modelling compounds that inhibit the COX2 enzyme. In this dataset, each individual molecule is represented as graph with nodes representing atoms and edges representing covalent bonds. Nodes are labeled by the *atom type*, while edges are unlabeled.

COX2	
Num Graphs $\|\mathcal{G}\|$	467
Num Classes $\|\Omega\|$	2
Class proportion	78.0% - 22.0%
Min. - Avg. - Max. $\|V\|$	32 - 41.2 - 56
Min. - Avg. - Max. $\|E\|$	34 - 43.4 - 59
Avg. node degree	2.11
Avg. edge density	0.05

Table 3.4: Summary of some metrics of the COX2 dataset.

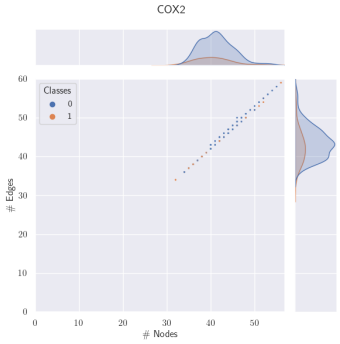
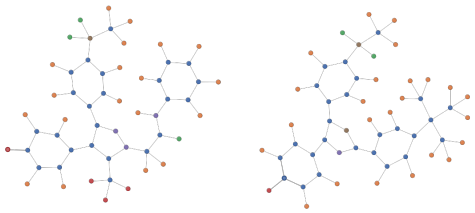


Fig. 3.8: Distribution of graphs by number of nodes and edges

This dataset has been primarily designed as a classification task to recognize COX2 enzymes and COX2 inhibiters. The COX2 enzymes are involved in processes like inflammation, pain, and fever and COX2 inhibitors target this enzyme, and are thus commonly used for anti-inflammatory, and antipyretic effects. Table 3.4 contains important metrics about the COX2 dataset. In addition to this table, Fig. 3.8 shows the distribution of the graphs in the dataset based on the number of nodes and edges. Fig. 3.9 displays two graph examples stemming from the two classes.



(a) Class '0' (b) Class '1'

Fig. 3.9: Example graph for both classes of the COX2 dataset.

The *Cyclooxygenase-2 - Modified* (COX2-MD) [97; 131] dataset is a modified version of the original COX2 dataset. In order to obtain a balanced dataset, the number of graphs in the dominant class has been reduced. Moreover for all graphs, the edge structure has been adapted to obtain complete graphs (see, for instance, Fig. 3.11 where we present a visual representation of two complete graphs stemming from the two classes). Table 3.5 and Fig. 3.10 contain a summary of the various graph metrics and a distribution of the graphs based on the number of nodes and edges, respectively.

COX2-MD	
Num Graphs $\|\mathcal{G}\|$	303
Num Classes $\|\Omega\|$	2
Class proportion	51.0% - 49.0%
Min. - Avg. - Max. $\|V\|$	21 - 26.3 - 36
Min. - Avg. - Max. $\|E\|$	210 - 335.1 - 630
Avg. node degree	25.28
Avg. edge density	1.00

Table 3.5: Summary of some metrics of the COX2-MD dataset.

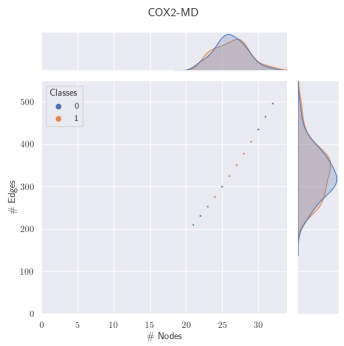


Fig. 3.10: Distribution of graphs by number of nodes and edges

(a) Class '0' (b) Class '1'

Fig. 3.11: Example graph for both classes of the COX2-MD dataset.

3.1.4 *DHFR & DHFR-MD*

The *Dihydrofolate Reductase* (DHFR) data, originally proposed in [131], is a collection of 756 graphs representing chemical compounds that inhibit the dihydrofolate reductase (DHFR) enzyme. In this graph dataset, each atom is represented by a node, and each covalent bond is represented by an edge. The nodes are labeled with the *atom type*, and the edges are unlabeled.

DHFR	
Num Graphs $\|\mathcal{G}\|$	756
Num Classes $\|\Omega\|$	2
Class proportion	39.0% - 61.0%
Min. - Avg. - Max. $\|V\|$	20 - 42.4 - 71
Min. - Avg. - Max. $\|E\|$	21 - 44.5 - 73
Avg. node degree	2.10
Avg. edge density	0.05

Table 3.6: Summary of some metrics of the DHFR dataset.

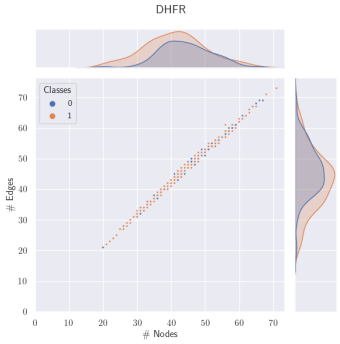
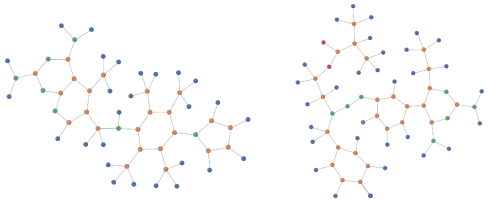


Fig. 3.12: Distribution of graphs by number of nodes and edges

The DHFR dataset has been designed as a classification task that consists of discriminating between molecular compounds that inhibit or not inhibit the DHFR enzyme. DHFR is an enzyme that plays a crucial role in DNA synthesis and is an important target for various pharmaceutical drugs. For a comprehensive overview of the dataset, including important metrics, as well as the distribution the graphs based on the number of nodes and edges refer to Table 3.6 and 3.12, respectively. Additionally, we provide two examples to visually grasp the structure of these graphs in Fig. 3.13.



(a) Class '0' (b) Class '1'

Fig. 3.13: Example graph for both classes of the DHFR dataset.

The *Dihydrofolate reductase - Modified* (DHFR-MD) [97; 131] dataset is a modified version of the original DHFR dataset. The edge structure of all graphs has been adapted to transform them into complete graphs (see Fig. 3.15 where we display two examples stemming from both classes representing complete graphs)[1]. Table 3.7 and Fig. 3.14 give an overview of the different metrics for this modified dataset and the distribution of the graphs based on the number of nodes and edges, respectively.

DHFR-MD	
Num Graphs $\lvert\mathcal{G}\rvert$	393
Num Classes $\lvert\Omega\rvert$	2
Class proportion	32.0% - 68.0%
Min. - Avg. - Max. $\lvert V\rvert$	13 - 23.9 - 39
Min. - Avg. - Max. $\lvert E\rvert$	78 - 283.0 - 741
Avg. node degree	22.87
Avg. edge density	1.00

Table 3.7: Summary of some metrics of the DHFR-MD dataset.

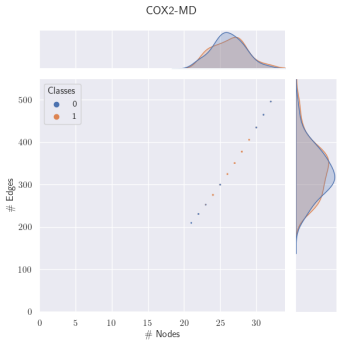


Fig. 3.14: Distribution of graphs by number of nodes and edges

(a) Class '0' (b) Class '1'

Fig. 3.15: Example graph for both classes of the DHFR-MD dataset.

[1]Note that, for this particular dataset and contrary to the other MD versions (i.e., BZR-MD, COX2-MD) the class proportion has not been even out.

3.1.5 *ER-MD*

The *Estrogen Receptor - Modified* (ER-MD) [131; 97] dataset contains 446 graphs representing ligands that interact with the estrogen receptor (ER). In this graph dataset, each atom corresponds to a node, and their bonds are represented by edges. Nodes are labeled with the *atom type* and edges are unlabelled.

ER-MD	
Num Graphs $\|\mathcal{G}\|$	446
Num Classes $\|\Omega\|$	2
Class proportion	59.0% - 41.0%
Min. - Avg. - Max. $\|V\|$	4 - 21.3 - 43
Min. - Avg. - Max. $\|E\|$	6 - 234.8 - 903
Avg. node degree	20.33
Avg. edge density	1.00

Table 3.8: Summary of some metrics of the ER-MD dataset.

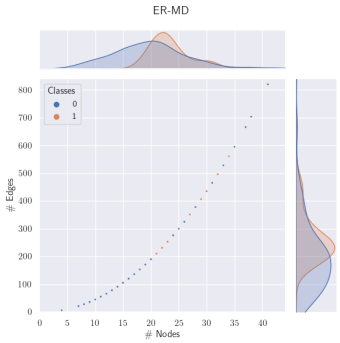


Fig. 3.16: Distribution of graphs by number of nodes and edges

ER-MD dataset is designed as a classification task, focusing on distinguishing compounds that interact with the ER enzyme and those that do not. ER-MD is a modification of the original ER graph dataset[2], with all graphs modified to be complete. Table 3.8 provides important metrics about the dataset and Fig. 3.16 illustrates the distribution of graphs according to their number of nodes and edges. Fig. 3.17 visually shows a chemical compound from each class in ER-MD.

(a) Class '0' (b) Class '1'

Fig. 3.17: Example graph for both classes of the ER-MD dataset.

[2]Note that the original ER dataset is not present in the TUDataset repository.

3.1.6 *MUTAG*

The *MUTAG* [132] dataset was introduced by [133] as a graph dataset in the Inductive Logic Programming (ILP) task and contains 188 graphs. The graphs in the MUTAG dataset represent the molecular structure of molecular compounds, with nodes representing atoms identified by their *atom type* and unlabeled edges representing covalent bounds.

MUTAG	
Num Graphs $\|\mathcal{G}\|$	188
Num Classes $\|\Omega\|$	2
Class proportion	34.0% - 66.0%
Min. - Avg. - Max. $\|V\|$	10 - 17.9 - 28
Min. - Avg. - Max. $\|E\|$	10 - 19.8 - 33
Avg. node degree	2.19
Avg. edge density	0.14

Table 3.9: Summary of some metrics of the MUTAG dataset.

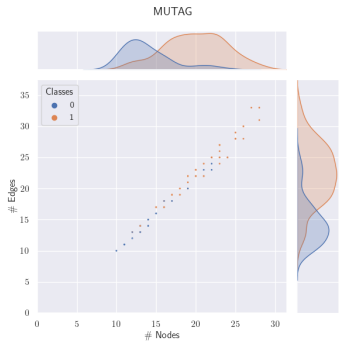
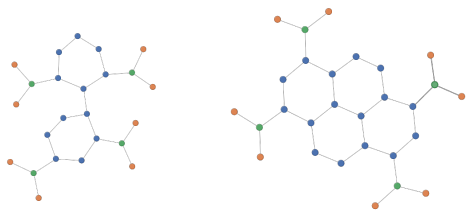


Fig. 3.18: Distribution of graphs by number of nodes and edges

The classification task of the MUTAG dataset is to predict the mutagenicity on Salmonella Typhimurium. That is the dataset is divided into two classes according to their mutagenic effect on bacteria. Table 3.9 contains important metrics for the MUTAG dataset. The distribution of graphs according to their nodes and edges is shown in Fig. 3.18. Fig. 3.19 provides a visual representation of two graphs for each class.



(a) Class '0' (b) Class '1'

Fig. 3.19: Example graph for both classes of the MUTAG dataset.

3.1.7 *MUTAGENICITY*

The *MUTAGENICITY* data originates from the *Chemical Carcinogenicity Research Information System* (CCRIS) database [134] The graph-based dataset is sourced from the IAM graph repository [130] and contains 4,337 graphs. In the MUTAGENICITY dataset, graph nodes represent atoms and are labeled by *atom type*, while edges represent covalent bonds and are unlabeled.

MUTAGENICITY	
Num Graphs $\|\mathcal{G}\|$	4,337
Num Classes $\|\Omega\|$	2
Class proportion	55.0% - 45.0%
Min. - Avg. - Max. $\|\boldsymbol{V}\|$	4 - 30.3 - 417
Min. - Avg. - Max. $\|\boldsymbol{E}\|$	3 - 30.8 - 112
Avg. node degree	2.04
Avg. edge density	0.09

Table 3.10: Summary of some metrics of the MUTAGENICITY dataset.

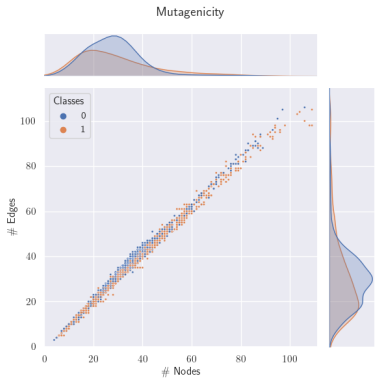
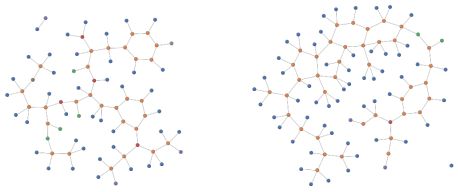


Fig. 3.20: Distribution of graphs by number of nodes and edges

Mutagenicity refers to the ability of a chemical to induce mutations in DNA. This dataset was therefore designed as a classification task, which consists of distinguishing between mutagenic and non-mutagenic molecules. Table 3.10 provides essential metrics about the dataset and Fig. 3.20 shows the distribution of the graphs based on the number of nodes and edges. Fig. 3.21 showcases example graphs from both classes.



(a) Class '0' (b) Class '1'

Fig. 3.21: Example graph for both classes of the MUTAGENICITY dataset.

3.1.8 *NCI1 & NCI109*

The *NCI1* and *NCI109* [135] datasets are both derived from the PubChem website [136] and are selected from the bioassay records for two different types of cancer cell lines. The nodes of the graphs of both datasets correspond to the various atoms, and the edges correspond to the bonds between the atoms. The labels on the nodes correspond to the *atom type* while edges are unlabelled. The NCI1 and NCI109 datasets both contain about 4,100 chemical compounds.

NCI1	
Num Graphs $\|\mathcal{G}\|$	4,110
Num Classes $\|\Omega\|$	2
Class proportion	50.0% - 50.0%
Min. - Avg. - Max. $\|V\|$	3 - 29.9 - 111
Min. - Avg. - Max. $\|E\|$	2 - 32.3 - 119
Avg. node degree	2.16
Avg. edge density	0.09

Table 3.11: Summary of some metrics of the NCI1 dataset.

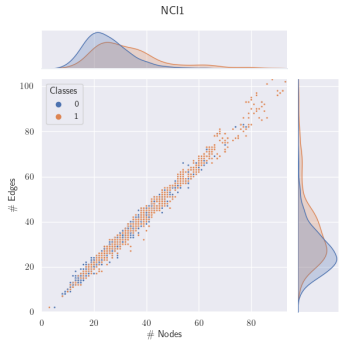
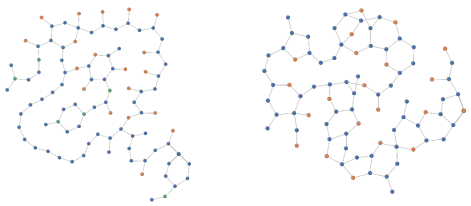


Fig. 3.22: Distribution of graphs by number of nodes and edges



(a) Class '0' (b) Class '1'

Fig. 3.23: Example graph for both classes of the NCI1 dataset.

Each of the NCI anti-cancer screens forms a classification problem. The class labels on these datasets are decided by the *outcome* field of the bioassay which is either active or inactive. The NCI1 dataset pertains to a human tumor cell line known as NCI-H23, specifically focusing on Non-small cell lung cancer. The NCI109 dataset pertains to a human tumor cell

line known as OVCAR-8, which is derived from ovarian cancer cells.

In Table 3.11 and Table 3.12, we present some key metrics about the NCI1 and NCI109 datasets, respectively. The graph distribution based on their number of nodes and edges is displayed in Fig. 3.22 and in Fig. 3.24 for the NCI1 and NCI109 datasets, respectively. Fig. 3.23 and Fig. 3.25 showcase two example graphs for the NCI1 and the NCI109 dataset, respectively.

NCI109	
Num Graphs $\|\mathcal{G}\|$	4,127
Num Classes $\|\Omega\|$	2
Class proportion	50.0% - 50.0%
Min. - Avg. - Max. $\|V\|$	4 - 29.7 - 111
Min. - Avg. - Max. $\|E\|$	3 - 32.1 - 119
Avg. node degree	2.16
Avg. edge density	0.09

Table 3.12: Summary of some metrics of the NCI109 dataset.

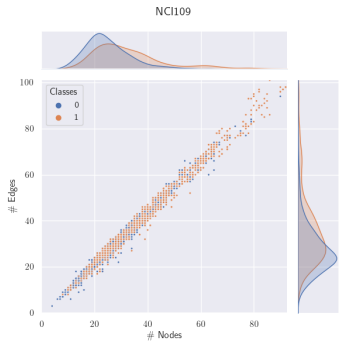
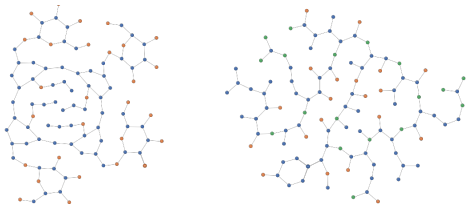


Fig. 3.24: Distribution of graphs by number of nodes and edges



(a) Class '0' (b) Class '1'

Fig. 3.25: Example graph for both classes of the NCI109 dataset.

3.1.9 *PTC*

The *PTC* dataset comes from the Predictive Toxicology Challenge [137]. The original PTC dataset is divided into four different subsets [97]: *PTC-MM*, *PTC-FM*, *PTC-MR*, and *PTC-FR*, each corresponding to a different type of laboratory animal. Each of the PTC datasets contains about 350 individual compounds. In these datasets, molecules are naturally represented as graphs, where nodes symbolize atoms labeled by the *atom type* and unlabeled edges denote chemical bonds.

PTC-MM	
Num Graphs $\|\mathcal{G}\|$	336
Num Classes $\|\Omega\|$	2
Class proportion	62.0% - 38.0%
Min. - Avg. - Max. $\|V\|$	2 - 14.0 - 64
Min. - Avg. - Max. $\|E\|$	1 - 14.3 - 71
Avg. node degree	1.97
Avg. edge density	0.22

Table 3.13: Summary of some metrics of the PTC-MM dataset.

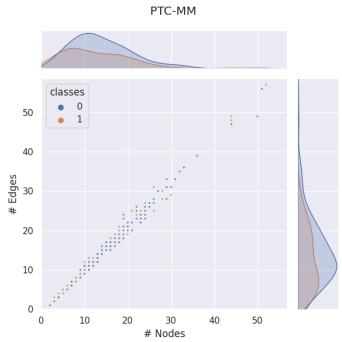
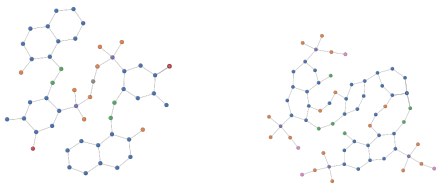


Fig. 3.26: Distribution of graphs by number of nodes and edges



(a) Class '0' (b) Class '1'

Fig. 3.27: Example graph for both classes of the PTC-MM dataset.

The PTC datasets serve as a resource for various toxicological prediction tasks related to chemical compounds. The task consist of determining whether or not a given chemical compound has as a specific toxicity on different types of laboratory animals.

In Tables 3.13, 3.14, 3.15, and 3.16, some key metrics concerning the PTC-MM, PTC-FM, PTC-MR and PTC-FR datasets, are presented. In addition, we display the graph distribution according to their number of nodes and edges for all PTC datasets in Figs. 3.26, 3.28, 3.29, and 3.30. In Fig. 3.27, we present two example graphs from the PTC-MM dataset (similar plots are obtained for the other datasets, PTC-FM, PTC-MR and PTC-FR).

PTC-FM	
Num Graphs $\lvert\mathcal{G}\rvert$	349
Num Classes $\lvert\Omega\rvert$	2
Class proportion	59.0% - 41.0%
Min. - Avg. - Max. $\lvert V\rvert$	2 - 14.1 - 64
Min. - Avg. - Max. $\lvert E\rvert$	1 - 14.5 - 71
Avg. node degree	1.98
Avg. edge density	0.22

Table 3.14: Summary of some metrics of the PTC-FM dataset.

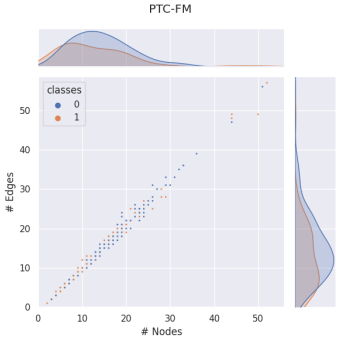


Fig. 3.28: Distribution of graphs by number of nodes and edges

PTC-MR	
Num Graphs $\lvert\mathcal{G}\rvert$	344
Num Classes $\lvert\Omega\rvert$	2
Class proportion	56.0% - 44.0%
Min. - Avg. - Max. $\lvert V\rvert$	2 - 14.3 - 64
Min. - Avg. - Max. $\lvert E\rvert$	1 - 14.7 - 71
Avg. node degree	1.98
Avg. edge density	0.21

Table 3.15: Summary of some metrics of the PTC-MR dataset.

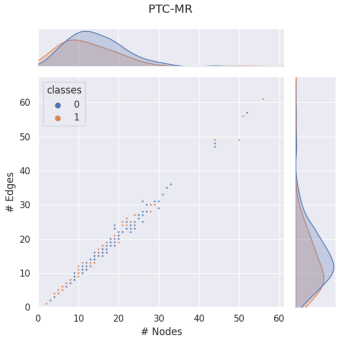


Fig. 3.29: Distribution of graphs by number of nodes and edges

PTC-FR	
Num Graphs $\|\mathcal{G}\|$	351
Num Classes $\|\Omega\|$	2
Class proportion	66.0% - 34.0%
Min. - Avg. - Max. $\|V\|$	2 - 14.6 - 64
Min. - Avg. - Max. $\|E\|$	1 - 15.0 - 71
Avg. node degree	1.99
Avg. edge density	0.21

Table 3.16: Summary of some metrics of the PTC-FR dataset.

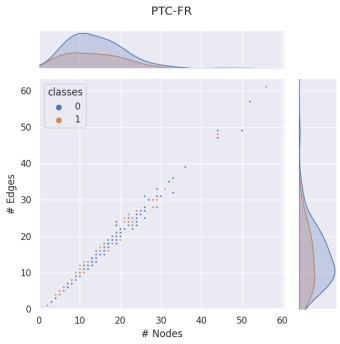


Fig. 3.30: Distribution of graphs by number of nodes and edges

3.2 Bioinformatic Graph Datasets

Bioinformatics is an interdisciplinary field that combines elements of biology, computer science, and statistics to analyze biological data [138]. It plays a pivotal role in modern biology, providing tools and methods for managing, analyzing, and interpreting large quantities of biological data. In the present thesis, we use graphs that represent two types of biological data, viz. proteins and brain images.

Proteins are complex molecules composed of amino acids linked into chains. Proteins play a crucial role in the structure and regulation of the body's tissues and organs [139]. The specific arrangement of the amino acids that make up a protein determines its unique function and structure. In the following sections, we discuss three graph datasets representing proteins

- DD (in Subsection 3.2.1)
- ENZYMES (in Subsection 3.2.2)
- PROTEINS (in Subsection 3.2.3)

A brain scan is a non-invasive procedure that uses a variety of imaging techniques to create pictures of the brain. Brain scans can be used to diagnose a variety of conditions, e.g., brain tumors, or neurological disorders. Brain scans can also be segmented into parts and can be used to find different relations between those parts. A simplified example of a segmented brain represented as a graph can be found in Fig. 3.31. We present the following graph datasets representing brain networks.

- KKI (in Subsection 3.2.4)
- OHSU (in Subsection 3.2.5)
- Peking-1 (in Subsection 3.2.6)

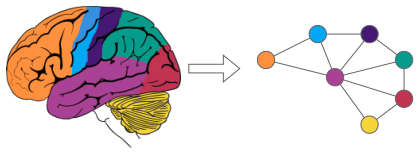


Fig. 3.31: Example of a brain segmented by region of interest represented as a graph[3].

3.2.1 *DD*

The DD dataset, introduced in [140], comprises a collection of 1,178 distinct protein structures. In this dataset, each individual protein is represented as a graph, with nodes representing amino acids. An edge connects two nodes if the corresponding amino acids are located within 6 angstroms of each other, indicating spatial proximity.

DD	
Num Graphs $\|\mathcal{G}\|$	1,178
Num Classes $\|\Omega\|$	2
Class proportion	59.0% - 41.0%
Min. - Avg. - Max. $\|V\|$	30 - 284.3 - 5748
Min. - Avg. - Max. $\|E\|$	63 - 715.7 - 14267
Avg. node degree	4.98
Avg. edge density	0.03

Table 3.17: Summary of some metrics of the DD dataset.

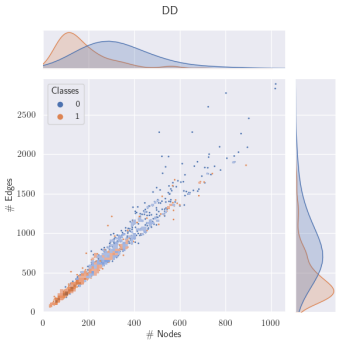
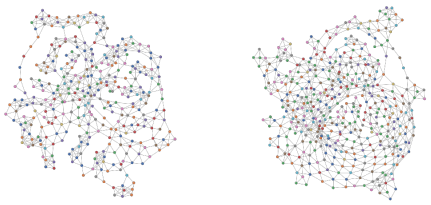


Fig. 3.32: Distribution of graphs by number of nodes and edges

The main objective of this dataset is a classification task that consists of distinguishing proteins that function as enzymes from those that have no enzymatic activity. Table 3.17 lists key metrics for the DD dataset, and Fig. 3.32 displays the distribution of graphs by nodes and edges. Fig. 3.33 shows a compound from each class, using distinct colors for amino acids (node labels).



(a) Class '0' (b) Class '1'

Fig. 3.33: Example graph for both classes of the DD dataset.

[3]Image adapted from `https://www.ninds.nih.gov/health-information/public-education/brain-basics/brain-basics-know-your-brain`

3.2.2 *ENZYMES*

The *ENZYMES* graph dataset is a collection of 600 protein tertiary structures obtained from the BRENDA enzyme database [141]. The ENZYMES dataset is formally proposed a graph dataset in [142] and graphs represent the structural and functional relationships between amino acids in proteins. The nodes serve as representations of secondary structure elements (SSE), and each node is labeled according to its specific type, whether it is a helix, sheet, or turn. The unlabeled edges represent non-covalent interactions between amino acids.

ENZYMES	
Num Graphs $\lvert\mathcal{G}\rvert$	600
Num Classes $\lvert\Omega\rvert$	6
Class proportion	17.0% - 17.0% 17.0% - 17.0% 17.0% - 17.0%
Min. - Avg. - Max. $\lvert V\rvert$	2 - 32.6 - 126
Min. - Avg. - Max. $\lvert E\rvert$	1 - 62.1 - 149
Avg. node degree	3.86
Avg. edge density	0.16

Table 3.18: Summary of some metrics of the ENZYMES dataset.

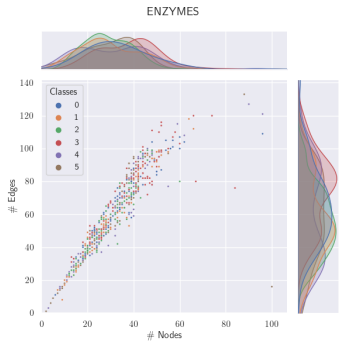
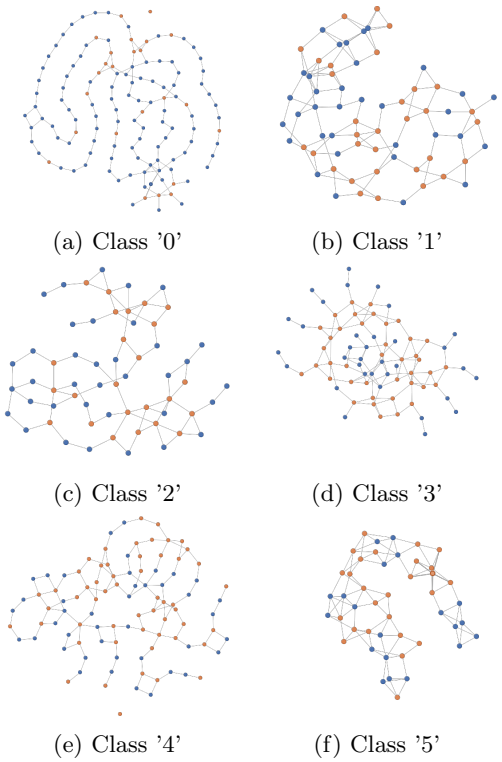


Fig. 3.34: Distribution of graphs by number of nodes and edges

The ENZYMES dataset is associated with the task of classifying 600 enzymes to one of the 6 EC top-level classes. Table 3.18 provides important statistical measures for the ENZYMES dataset, offering a comprehensive overview of its key metrics. Meanwhile, Fig. 3.34 visually presents the distribution of graphs based on their nodes and edges. Furthermore, Fig. 3.35 offers a visual representation of an enzyme from each class within the ENZYMES dataset. Different colors are employed to distinguish between various amino acids.



(a) Class '0' (b) Class '1'

(c) Class '2' (d) Class '3'

(e) Class '4' (f) Class '5'

Fig. 3.35: Example graph for all classes of the ENZYMES dataset.

3.2.3 *PROTEINS*

The *PROTEINS* dataset originates from the Protein Data Bank (PDB) [143] and was created and transformed to a graph dataset in [140; 142]. The 1,113 graphs in the PROTEINS dataset represent structural and functional relationships among amino acids in proteins. PROTEINS uses secondary structure elements as nodes, depicting structural and functional connections between amino acids in proteins. Unlabeled edges connect two nodes which are neighbors along the amino acid sequence.

PROTEINS	
Num Graphs $\|\mathcal{G}\|$	1,113
Num Classes $\|\Omega\|$	2
Class proportion	60.0% - 40.0%
Min. - Avg. - Max. $\|\boldsymbol{V}\|$	4 - 39.1 - 620
Min. - Avg. - Max. $\|\boldsymbol{E}\|$	5 - 72.8 - 1049
Avg. node degree	3.73
Avg. edge density	0.21

Table 3.19: Summary of some metrics of the PROTEINS dataset.

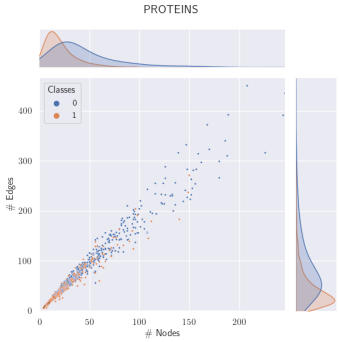
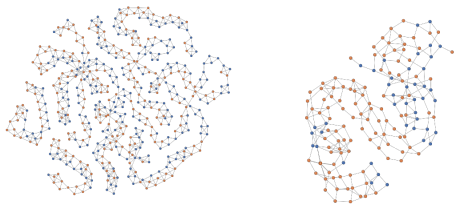


Fig. 3.36: Distribution of graphs by number of nodes and edges

The PROTEINS dataset differs from ENZYMES, as it comes with the task of classifying into enzymes and non-enzymes. Table 3.19 presents relevant metrics for the PROTEINS dataset. Alongside, Fig. 3.36 visually represents the distribution of graphs, giving an overview of their structural diversity. Furthermore, Fig. 3.37 serves as a visual representation, presenting sample graphs from both classes.



(a) Class '0' (b) Class '1'

Fig. 3.37: Example graph for both classes of the PROTEINS dataset.

3.2.4 *KKI*

The *KKI* [144] dataset is a brain network derived from the complete atlas of functional brain resonance images (fMRI) [145]. The graphs in the KKI graph dataset represent the functional relationships between different brain regions. In these graphs, each node represents a specific *Region Of Interest* (ROI) in the brain, while edges indicate correlations between pairs of these ROIs.

KKI	
Num Graphs $\|\mathcal{G}\|$	83
Num Classes $\|\Omega\|$	2
Class proportion	45.0% - 55.0%
Min. - Avg. - Max. $\|V\|$	5 - 27.0 - 90
Min. - Avg. - Max. $\|E\|$	4 - 48.4 - 237
Avg. node degree	3.19
Avg. edge density	0.18

Table 3.20: Summary of some metrics of the KKI dataset.

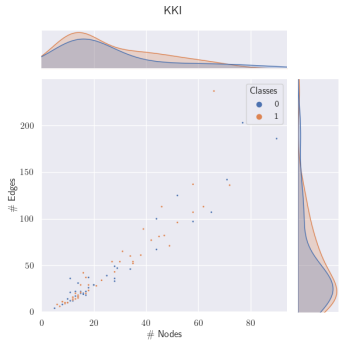
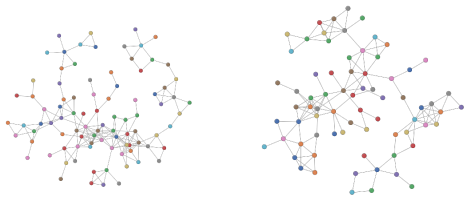


Fig. 3.38: Distribution of graphs by number of nodes and edges

The construction of the ROI network has been specifically designed for the classification of *Attention Deficit Hyperactivity Disorder* (ADHD), a neurodevelopmental disorder characterized by symptoms such as inattention, hyperactivity, and impulsivity. Table 3.20 provides key metrics for the dataset and Fig. 3.38 visually presents how graphs are distributed in terms of nodes and edges. Fig. 3.39 showcases ROI networks from each class in the KKI dataset.



(a) Class '0' (b) Class '1'

Fig. 3.39: Example graph for both classes of the KKI dataset.

3.2.5 *OHSU*

The *OHSU* dataset introduced in [144], is – similar as KKI – a brain network derived from the comprehensive atlas of functional brain resonance images (fMRI) [145]. Functional connectivity graphs are constructed from this fMRI data. Each node in these graphs signifies a distinct region of interest (ROI) within the brain, while edges denote correlations between pairs of these ROIs.

OHSU	
Num Graphs $\|\mathcal{G}\|$	79
Num Classes $\|\Omega\|$	2
Class proportion	44.0% - 56.0%
Min. - Avg. - Max. $\|V\|$	9 - 82.0 - 171
Min. - Avg. - Max. $\|E\|$	9 - 199.7 - 823
Avg. node degree	4.33
Avg. edge density	0.08

Table 3.21: Summary of some metrics of the OHSU dataset.

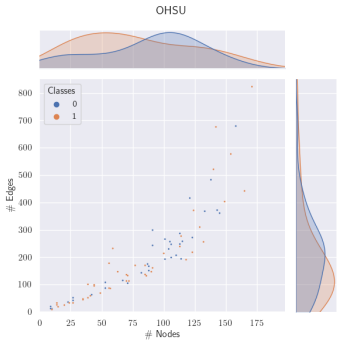
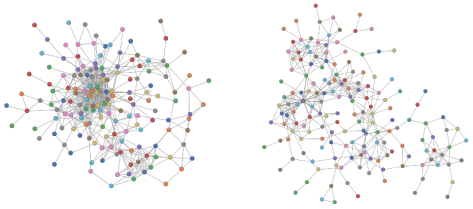


Fig. 3.40: Distribution of graphs by number of nodes and edges

The construction of the ROI network is specifically designed for the classification of Attention Deficit Hyperactivity Disorder (ADHD). That is, the OHSU dataset is designed for the task of hyperactive-impulsive (HI) classification. In Table 3.21, we give important metrics on the OHSU dataset, while in Fig. 3.40, we display the distribution of the graphs on the basis of their number of nodes and edges. Furthermore, Fig. 3.41 serves as a visual showcase, presenting ROI networks from both classes in the OHSU dataset.



(a) Class '0' (b) Class '1'

Fig. 3.41: Example graph for both classes of the OHSU dataset.

3.2.6 *Peking-1*

The *Peking-1* [144] dataset is a brain graph derived from the complete atlas of functional brain resonance images (fMRI) [145]. This dataset comprises resting-state fMRI. To construct functional connectivity graphs, the fMRI data is combined with the CC200 functional parcellation algorithm [145]. In these graphs, each node represents a specific region of interest (ROI) within the brain, while edges signify correlations between pairs of these ROIs.

Peking-1	
Num Graphs $\|\mathcal{G}\|$	85
Num Classes $\|\Omega\|$	2
Class proportion	58% - 42.0%
Min. - Avg. - Max. $\|V\|$	7 - 39.3 - 134
Min. - Avg. - Max. $\|E\|$	7 - 77.4 - 535
Avg. node degree	3.55
Avg. edge density	0.13

Table 3.22: Summary of some metrics of the Peking-1 dataset.

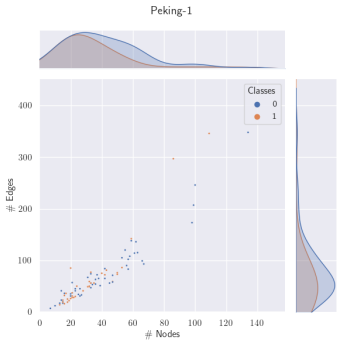
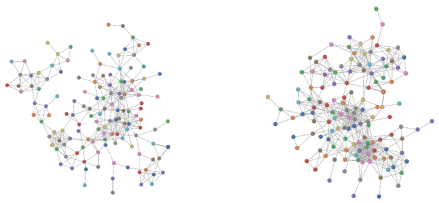


Fig. 3.42: Distribution of graphs by number of nodes and edges

The Peking-1 dataset is constructed for gender classification. Table 3.22 provides key metrics about the Peking-1 dataset. Meanwhile, Fig. 3.42 illustrates the distribution of graphs based on their nodes and edges. Fig. 3.43 showcases ROI networks from both classes within the Peking-1 dataset.



(a) Class '0' (b) Class '1'

Fig. 3.43: Example graph for both classes of the Peking-1 dataset.

3.3 Computer Vision Graph Datasets

Graph-based computer vision is a branch of computer vision that uses graph theory to represent and analyze visual data [24; 146]. In this approach, an image or video is represented as a graph, where each pixel, superpixel, or region of interest (ROI) is a node in the graph. The edges between nodes are then often used to represent the relationships between the corresponding nodes, capturing their similarity, spatial proximity, or functional connectivity. These relationships can be quantified using various features, such as intensity, color, texture, or motion vectors. Graph representations make it possible to apply graph algorithms and techniques to extract meaningful information from visual data (e.g., *Multi-level Graph Learning Network* (MGLN) for *Hyperspectral Image* (HSI) classification [147]).

Graphs with regular structures, like grid graphs, can be used to represent images. In the context of this special type of graph, nodes (representing individual pixels) may contain discrete or continuous vector-valued data, representing a range of information about the pixel. For instance, if an image contains rich 3-channel color information, this can be represented by continuous node attributes.

Images often come with additional data and semantic annotations (known as metadata). However, due to the resource-intensive nature of data collection, this information may only be partially available.

A rather simplified example of a graph representing an image using the image segmentation annotation is presented in Fig. 3.44.

In the following subsection, we present two graph datasets representing semantic images:

- MSRC9 and MSRC-21 (in Subsection 3.3.1)

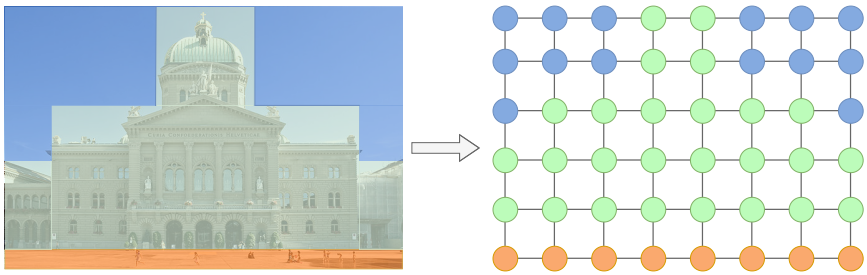


Fig. 3.44: Example of an image represented as a grid-graph[4].

3.3.1 *MSRC-9 & MSRC-21*

The MSRC 9-class and MSRC 21-class datasets are used in semantic image processing, as introduced in [148], and transformed as a graph dataset in [149]. Each image is represented as a conditional Markov random field graph, derived from image over-segmentation into superpixels. Nodes are connected if the superpixels are adjacent, and each node can further be annotated with a semantic label. Node labels are derived by considering the most common label among all pixels in the corresponding superpixel.

MSRC-9	
Num Graphs $\|\mathcal{G}\|$	221
Num Classes $\|\Omega\|$	8
Class proportion	9.0% - 14.0% 14.0% - 14.0% 10.0% - 14.0% 13.0% - 14.0%
Min. - Avg. - Max. $\|V\|$	25 - 40.6 - 55
Min. - Avg. - Max. $\|E\|$	53 - 97.9 - 145
Avg. node degree	4.82
Avg. edge density	0.12

Table 3.23: Summary of some metrics of the MSRC-9 dataset.

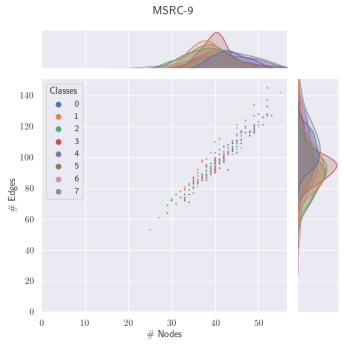
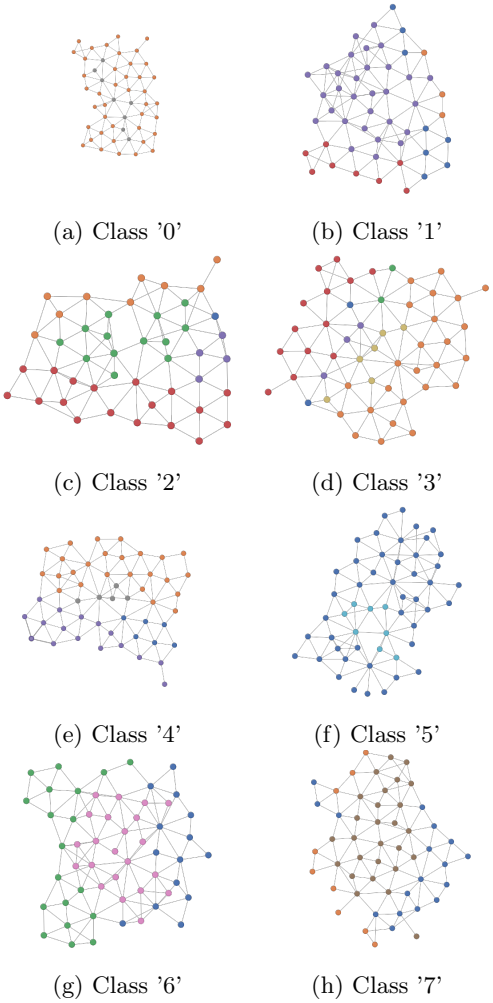


Fig. 3.45: Distribution of graphs by number of nodes and edges

Semantic labels include various classes like *building*, *grass*, *tree*, *cow*, and others, along with a *void* label for unclassified objects. Images with *void* labels were excluded, resulting in a classification task with eight classes for MSRC-9 and 20 classes for MSRC-21.

Table 3.23 and Table 3.24 provides important metrics for the MSRC-9 and MSRC-21 dataset, respectively. Fig. 3.45 and Fig. 3.47 show the distribution of graphs by the number of nodes and edges for the MSRC-9 and MSRC-21 datasets, respectively. Finally, in Fig. 3.46 and Fig. 3.48, a visual example is presented for each class for both datasets.

[4]Image adapted from `https://commons.wikimedia.org/wiki/File:Bundeshaus_Bern_2009,_Flooffy.jpg`



(a) Class '0' (b) Class '1'

(c) Class '2' (d) Class '3'

(e) Class '4' (f) Class '5'

(g) Class '6' (h) Class '7'

Fig. 3.46: Example graph for all classes of the MSRC-9 dataset.

MSRC-21	
Num Graphs $\lvert\mathcal{G}\rvert$	563
Num Classes $\lvert\Omega\rvert$	20
Class proportion	4.0% - 5.0% 5.0% - 5.0% 5.0% - 5.0% 5.0% - 5.0% 5.0% - 6.0% 5.0% - 6.0% 5.0% - 5.0% 4.0% - 5.0% 5.0% - 4.0% 5.0% - 2.0%
Min. - Avg. - Max. $\lvert V\rvert$	51 - 77.5 - 141
Min. - Avg. - Max. $\lvert E\rvert$	121 - 198.3 - 405
Avg. node degree	5.10
Avg. edge density	0.07

Table 3.24: Summary of some metrics of the MSRC-21 dataset.

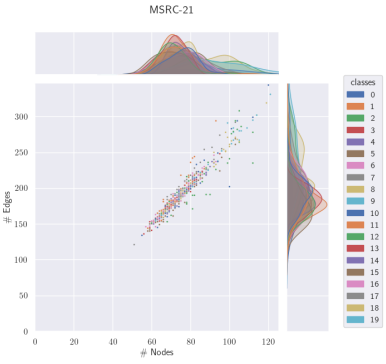
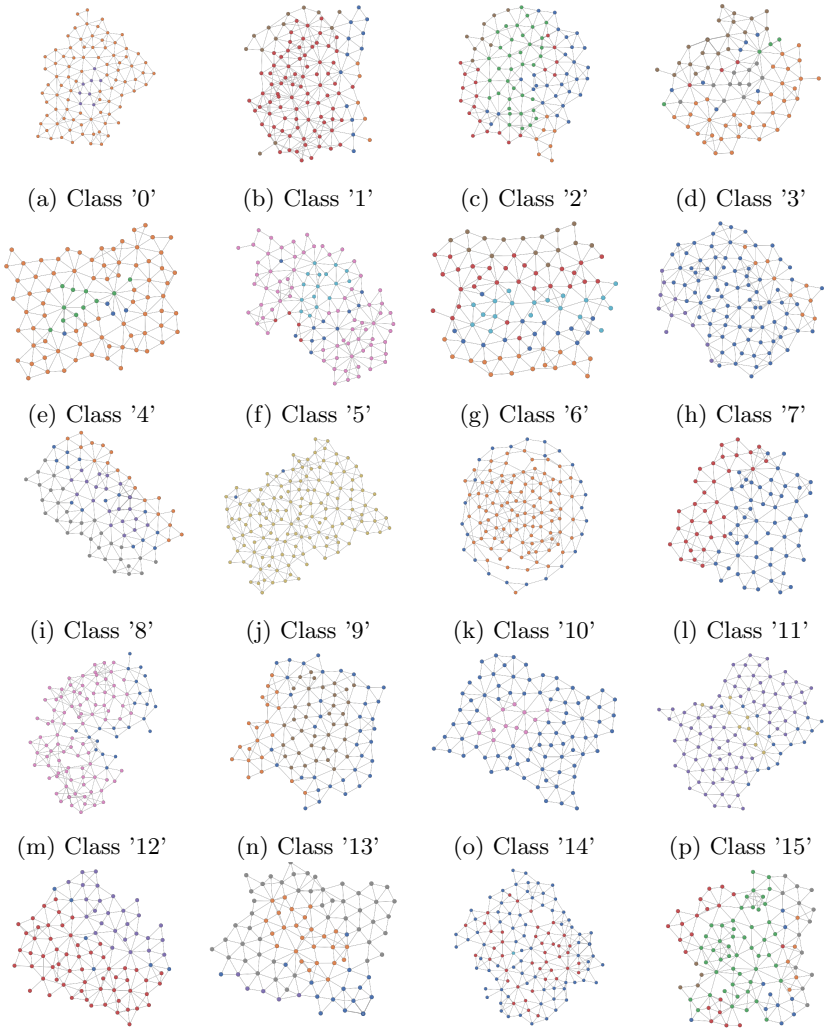


Fig. 3.47: Distribution of graphs by number of nodes and edges



(a) Class ’0’ (b) Class ’1’ (c) Class ’2’ (d) Class ’3’

(e) Class ’4’ (f) Class ’5’ (g) Class ’6’ (h) Class ’7’

(i) Class ’8’ (j) Class ’9’ (k) Class ’10’ (l) Class ’11’

(m) Class ’12’ (n) Class ’13’ (o) Class ’14’ (p) Class ’15’

(q) Class ’16’ (r) Class ’17’ (s) Class ’18’ (t) Class ’19’

Fig. 3.48: Example graph for all classes of the MSRC-21 dataset.

3.4 Social Networks Graph Datasets

Social networks are a kind of graph in which the nodes typically represent individual persons and the edges the social connections between them [150]. In this way, it is possible to model a wide range of social relationships, such as friendships, family ties, and professional relationships.

Two main types of graphs are commonly used to represent social networks, viz. undirected and directed graphs. Undirected graphs are suitable for modeling relationships such as friendships, in which the connection is mutual. A directed graph is better suited to modeling relationships such as following or mentoring, in which one person follows or guides another.

Graphs can be used to analyze social networks in a variety of ways. For example, we can use graphs to identify key individuals in a network. To this end, one can identify individuals who have a large number of connections or who are close to many other important individuals. A second task consists of identifying communities in a network. The objective is to identify groups of individuals who are more closely connected than they are to individuals in other groups. An example of a simplified social network graph can be found in Fig. 3.49.

The following sections present the following four graph datasets sampled from the social network domain.

- COLLAB (in Subsection 3.4.1)
- IMDB-BINARY (in Subsection 3.4.2)
- REDDIT-MULTI-5K and -MULTI-12K (in Subsection 3.4.3).

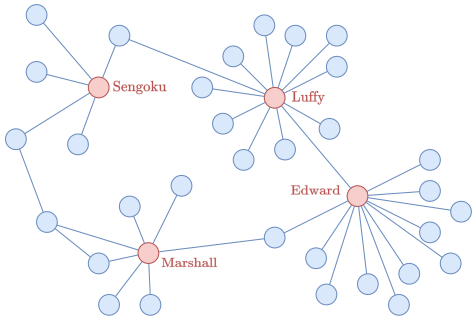


Fig. 3.49: Example of a social network graph

3.4.1 *COLLAB*

The *COLLAB* [151] dataset is a compilation of scientific collaboration graphs. It is derived from three public collaboration datasets [152]: *High Energy Physics*, *Condensed Matter Physics*, and *Astro Physics*. Each graph represents the ego-network of a researcher within a specific research field. That is the nodes represent researchers and an edge between two nodes exists if the corresponding researchers have collaborated with each others.

COLLAB	
Num Graphs $\|\mathcal{G}\|$	5,000
Num Classes $\|\Omega\|$	3
Class proportion	52.0% - 15.0% 33.0%
Min. - Avg. - Max. $\|V\|$	32 - 74.5 - 492
Min. - Avg. - Max. $\|E\|$	60 - 2457.2 - 40119
Avg. node degree	37.37
Avg. edge density	0.51

Table 3.25: Summary of some metrics of the COLLAB dataset.

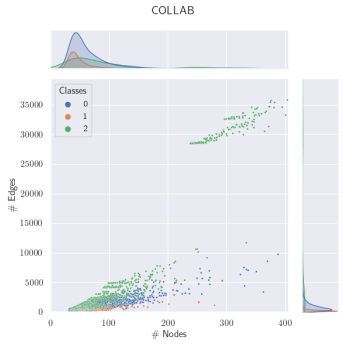


Fig. 3.50: Distribution of graphs by number of nodes and edges

The COLLAB dataset is designed as a classification task, where the graph labels indicate the corresponding research area. Table 3.25 provides important metrics about the COLLAB dataset. Fig. 3.50 illustrates the distribution of graphs based on their nodes and edges. Additionally, Fig. 3.51 shows ego-networks of different classes within the COLLAB dataset. It is noteworthy that in these graphs, the color of a node corresponds to the number of its neighbors.

(a) Class '0' (b) Class '1' (c) Class '2'

Fig. 3.51: Example graph for all classes of the COLLAB dataset.

3.4.2 *IMDB-BINARY*

The *IMDB-BINARY* [151] is a collection of ego-networks derived for each actor/actress from movie data on IMDB[5]. In each graph, nodes represent actors/actresses, and an edge connects them if they appear in the same movie.

IMDB-BINARY	
Num Graphs $\|\mathcal{G}\|$	1,000
Num Classes $\|\Omega\|$	2
Class proportion	50.0% - 50.0%
Min. - Avg. - Max. $\|V\|$	12 - 19.8 - 136
Min. - Avg. - Max. $\|E\|$	26 - 96.5 - 1249
Avg. node degree	8.89
Avg. edge density	0.52

Table 3.26: Summary of some metrics of the IMDB-BINARY dataset.

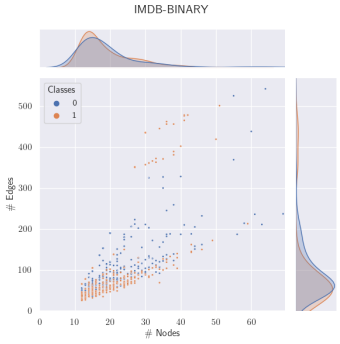
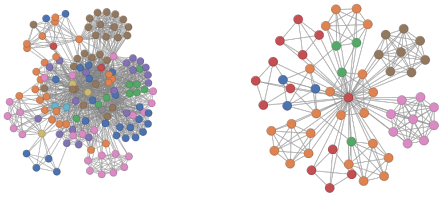


Fig. 3.52: Distribution of graphs by number of nodes and edges

These collaboration graphs are specifically generated for *Action* and *Romance* genres and each ego-network is labeled with the corresponding genre the graph belongs to. The objective is to predict the genre of an ego-network graph. Table 3.26 presents key information about the IMDB-BINARY dataset. Fig. 3.52 visually presents how graphs are distributed based on the number of nodes and edges. Fig. 3.53 displays ego-networks from both classes within the IMDB-BINARY dataset. Note that in these graphs, the node color corresponds to the number of its neighbors.



(a) Class '0' (b) Class '1'

Fig. 3.53: Example graph for all classes of the IMDB-BINARY dataset.

[5] `https://www.imdb.com/`

3.4.3 *REDDIT-MULTI-5K & REDDIT-MULTI-12K*

The *REDDIT-MULTI-5K* and *REDDIT-MULTI-12K* [151] datasets are collections of online subreddits[6] discussion represented as graphs. In both REDDIT-MULTI-5K and REDDIT-MULTI-12K, each graph corresponds to a discussion thread on the Reddit platform, where nodes represent users and there is an edge between two nodes if at least one of them responds to another's comment.

REDDIT-MULTI-5K	
Num Graphs $\|\mathcal{G}\|$	4,999
Num Classes $\|\Omega\|$	5
Class proportion	20.0% - 20.0% 20.0% - 20.0% 20.0%
Min. - Avg. - Max. $\|V\|$	22 - 508.5 - 3648
Min. - Avg. - Max. $\|E\|$	21 - 594.9 - 4783
Avg. node degree	2.25
Avg. edge density	0.01

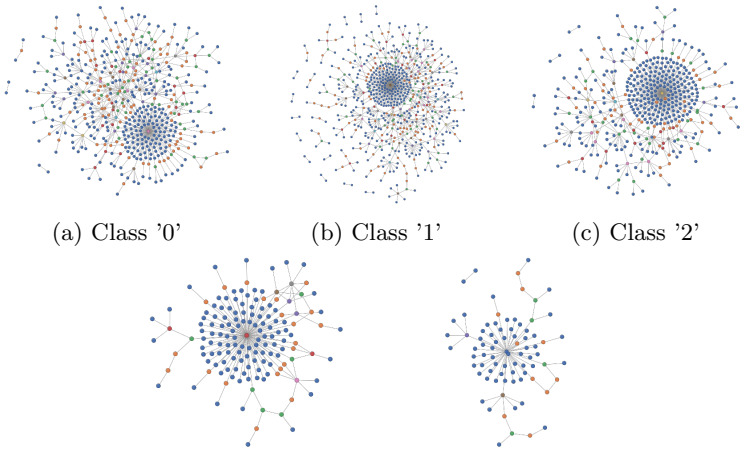
Table 3.27: Summary of some metrics of the REDDIT-MULTI-5K dataset.

Fig. 3.54: Distribution of graphs by number of nodes and edges

In REDDIT-MULTI-5K, the data is sourced from five different subreddits: *worldnews*, *videos*, *AdviceAnimals*, *aww*, and *mildlyinteresting*. *REDDIT-MULTI-12K* includes all the subreddits found in *REDDIT-MULTI-5K* along with a broader selection of subreddits, viz. *AskReddit*, *atheism*, *IAmA*, *Showerthoughts*, *todayilearned*, *TrollXChromosomes*. Each graph is labeled according to its originating subreddit. The objective of the classification task is to categorize each graph into the corresponding subreddit.

Table 3.27 and Table 3.28 provide important metrics for the REDDIT-MULTI-5K and REDDIT-MULTI-12K dataset, respectively. Fig. 3.54 and Fig. 3.56 illustrate the distribution of graphs based on their nodes and edges for REDDIT-MULTI-5K and REDDIT-MULTI-12K, respectively. Finally, Fig. 3.55 and Fig. 3.57 provide visual examples for each class in both datasets. Note that for those graph representations the color of a node corresponds to the number of its neighbors.

[6] `https://www.reddit.com/`



(a) Class '0' (b) Class '1' (c) Class '2'

(d) Class '3' (e) Class '4'

Fig. 3.55: Example graph for all classes of the REDDIT-MULTI-5K dataset.

REDDIT-MULTI-12K	
Num Graphs $\|\mathcal{G}\|$	11,929
Num Classes $\|\Omega\|$	11
Class proportion	6.0% - 9.0% 8.0% - 10.0% 4.0% - 8.0% 10.0% - 9.0% 4.0% - 22.0% 8.0%
Min. - Avg. - Max. $\|V\|$	2 - 391.4 - 3782
Min. - Avg. - Max. $\|E\|$	1 - 456.9 - 5171
Avg. node degree	2.28
Avg. edge density	0.02

Table 3.28: Summary of some metrics of the REDDIT-MULTI-12K dataset.

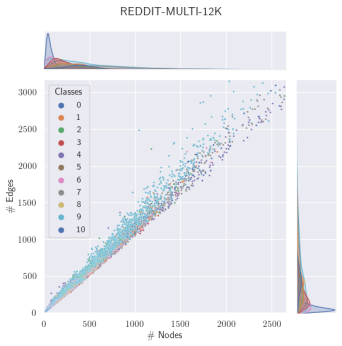
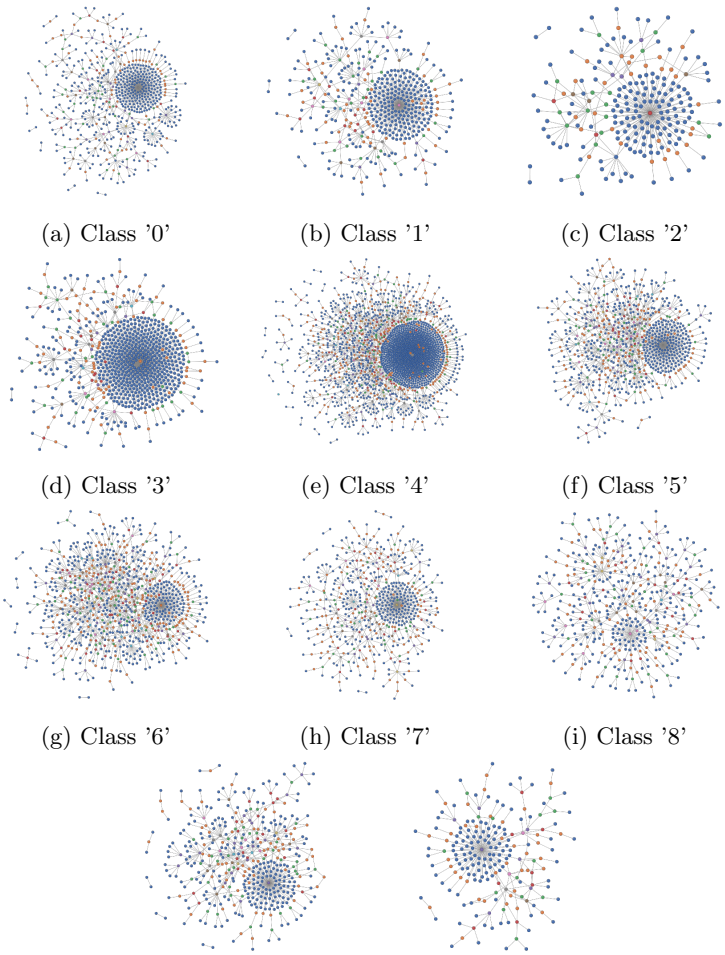


Fig. 3.56: Distribution of graphs by number of nodes and edges



(a) Class ’0’ (b) Class ’1’ (c) Class ’2’

(d) Class ’3’ (e) Class ’4’ (f) Class ’5’

(g) Class ’6’ (h) Class ’7’ (i) Class ’8’

(j) Class ’9’ (k) Class ’10’

Fig. 3.57: Example graph for all classes of the REDDIT-MULTI-12K dataset.

3.5 Dataset Filtering

In the present section, we present a thorough experiment (presented in a conference paper [46]) to filter out graph datasets unsuited for evaluating the proposed graph reduction methods (detailed in the following Chapters 4, 5, and 6).

First, in Subsection 3.5.1, we detail the configurations tested for the three classifier paradigms (described in Chapter 2). Following this, in Subsection 3.5.2, we provide a detailed description of the technical parameters employed in the experiments, including the evaluation scheme, metrics, and hyperparameter ranges. The actual experimental evaluation is then divided into two parts. First, in Subsection 3.5.3, we compare classification accuracies across the three classifier paradigms in the three configurations for both the labeled and unlabeled graph datasets. Second, in Subsection 3.5.4, we analyze in which datasets the graph-based approaches are actually significantly better than the vector-based counterparts. In the same section, we ultimately define the different graph datasets that are used for the various evaluations in the remainder of the present thesis.

3.5.1 *Classification Methods Comparison*

The overall information of a graph consists of two parts, namely the structure and the labels. By omitting one or the other, or using both pieces of information at the same time, we thus obtain three different configurations. To determine the most beneficial piece of information in a graph in the context of graph classification, we use three classification paradigms that are graph edit distance, graph kernel, and graph neural network coupled with the three configurations presented in the present paragraph.

In Table 3.29, the three configurations are summarised for all the different classifiers (see Fig. 3.58).

- **(I)** The initial configuration (shown in column 1) consists of ***L****abeled Graphs*, which involves conducting graph classification on the original graphs, including both its structure and node labels.
- **(II)** The second configuration (shown in column 2) aims to examine the significance of the graph structure itself by excluding the node labels to obtain ***U****nlabeled Graphs*.
- **(III)** The final configuration (shown in the third column) is the ***A****ggregated Labels* setup, in which only the node information is

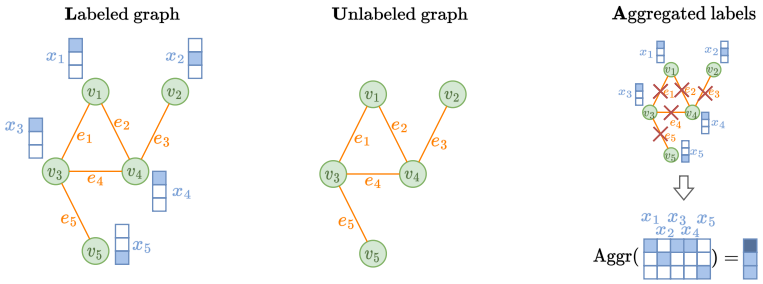


Fig. 3.58: A visual summary of the three tested configurations.

Table 3.29: Three configurations (L, U, A) for the three classifier paradigms (k-NN, SVM, NN). **(I) L**abeled Graphs: Original graphs with node labels and graph structure, **(II) U**nlabeled Graphs: Graphs without labels (graph structure only), and **(III) A**ggregated Labels: Vector representation of the graphs based on the node labels.

		Graph-Based		**Vector-Based**
		Labeled Graphs	**U**nlabeled Graphs	**A**ggregated Labels
Classifier	***k*-NN**	GED(L)	GED(U)	L_2(A)
	SVM	WL(L)	WL(U)	RBF(A)
	NN	GNN(L)	GNN(U)	MLP(A)

retained. We use global sum pooling on the graphs' nodes to obtain a basic vector representation of the graphs. This feature vector is then fed into three statistical classifiers (a k-NN classifier using the Euclidean distance (L_2), an SVM with a radial basis function (RBF), and a multilayer perceptron (MLP)). The rationale of this procedure is to use statistical classifiers that are conceptually closely related to the three graph-based classifiers used for the two configurations.

3.5.2 *Experimental Setup*

In order to ensure the reliability of the empirical results and reduce the impact of random data partitioning, we use a 10-fold cross-validation strategy that is repeated 10 times in a stratified manner for each dataset and configuration. The performance of the models is then estimated using each partition, where hyperparameters are chosen through an internal model selection process that only uses the training data. Note that the model selection is conducted independently for each training and test split, thus the optimal hyperparameter configurations may differ from one split to another.

A common metric to assess the quality of a classifier is the classification accuracy. It measures the relative proportion of correctly classified instances out of the total number of instances. However, the classification accuracy can be misleading in imbalanced datasets, where the number of instances in one class is much larger than the number of instances in another class (this may cause the classifier to predict the majority class more often and thus report over-optimistic results). Therefore, we report the balanced accuracy [153] in this chapter since not all datasets used are class-balanced.

Concerning the computation of BP-GED, we use unit costs for both node and edge insertions/deletions. For the node substitution cost, we use the Euclidean distance between the corresponding node labels. For edge substitution, we use a zero cost (since the edges of all graphs are unlabeled). Parameter $\alpha \in]0, 1[$ represents the relative importance of node and edge edit operation costs and is varied from 0.1 to 0.9 in increments of 0.1 in our evaluation. For the k-NN classifier, we optimize the number of neighbors k within the range $k \in \{3, 5, 7\}$.

In the kernel scenario, we use a 4-Weisfeiler-Lehman kernel which means we perform four refinement iterations. To optimize the SVM parameter C, which balances the trade-off between large margins and minimizing misclassification, we explore values in the range $10^{-2.0,-1.5,\ldots,2.0}$. We use the same range for the regularization parameter when optimizing the SVM with radial basis function for the aggregated node labels.

For the training process of the GNN experiments, we use the hyperparameters as proposed in [125]. For the fully connected network, the following parameters are optimized. The depth of the standard fully connected layers $d \in \{1, 3, 5, 10\}$, the number of neurons per layer $n \in \{5, 10, 20, 30\}$, the dropout value $\delta \in \{0, 0.25, 0.5\}$ and the learning rate $l \in \{0.005, 0.01, 0.05, 0.1\}$.

3.5.3 *Graph Classification*

Fig. 3.59 shows the balanced classification accuracy results obtained by the classifiers in the tested settings on labeled graph datasets. For the three graph-based classifiers (GED, WL, GNN) two different setups are evaluated, viz. labeled graphs (L) and unlabeled graphs (U). The three vector-based classifiers (L_2, RBF, MLP) are tested on the aggregated node labels (A).

Four main trends can be observed from Fig. 3.59 on the node-labeled graph datasets. First, we observe that on the datasets MUTAGENICITY, NCI1, and NCI109 the three different types of classifiers somehow achieve the expected results. That is the statistical classifiers using the aggregated vectors achieve the lowest accuracies, the second-best performance is achieved with graph-based methods using unlabeled graphs, and the best classification results are obtained on the original graphs.

Second, we notice that on the six datasets BZR-MD, COX2-MD, MSRC-9, MSRC-21, PROTEINS, and DD, the classifiers face difficulties when the node labels are removed from the graphs. That is, on those datasets, the accuracies obtained by the classifiers operating on unlabeled graphs is almost consistently lower than those of the classifiers that operate on the aggregated feature vectors. That indicates that the structure in those datasets is complicated to distinguish from one another, and moreover, that the labels on the nodes play a pivotal role in those applications.

Third, we observe that all classifiers have difficulties performing well on the OHSU, Peking-1, and KKI datasets. The achieved results are only slightly better than random predictions, indicating that these tasks are extremely challenging and currently unsolved by the tested classifiers.

Finally, we observe that on the datasets BZR-MD, COX2, COX2-MD, ER-MD, DHFR-MD, MSRC-9, MSRC-21, DD, PROTEINS, and the four PTCs datasets, the performance of the naïve vector-based approach is comparable to that of the more advanced graph-based techniques. We have two possible explanations for this phenomenon. First, for the MD versions of these datasets, the graphs are heavily modified and fully connected, rendering graph-based classification less effective. Second, by visualizing the aggregated feature vectors with T-SNE in Fig. 3.60 [7], we find that the different classes are easily separable on these datasets, which in turn explains the good results of the vector-based classifiers.

Fig. 3.61 (a) and (b) illustrate the balanced classification accuracy ob-

[7] Fig. 3.60 displays the T-SNE visualization for only four datasets the visualization of the remaining datasets can be found in Appendix A.1.

tained by the classifiers on unlabeled graph datasets, both without and with the inclusion of the node degree information, respectively. For the three graph-based classifiers (GED, WL, GNN), only the unlabeled graphs (U) configuration is evaluated. The three vector-based classifiers (L_2, RBF, MLP) are tested with aggregated "dummy"-node labels (A).

Note that the computation of the GED(U) for the REDDIT-MULTI-5K and the REDDIT-MULTI-12K datasets are computationally too expensive, so we cannot report those results. When analyzing the results of the balanced classification accuracy on the unlabeled graph datasets in Fig. 3.61 (a), we observe that the advanced graph classification methods consistently outperform the naïve baseline across all datasets and all classification methods. A second observation can be made from the use of node degree (see Fig. 3.61 (b)). Specifically, we note that incorporating node degree potentially enhances the classification accuracies of graph-based classifiers, though not consistently across all datasets. That is, we can see an improvement in the results compared to the naïve baseline embedding.

The T-SNE visualization for the unlabeled graph datasets is presented in Fig. 3.62. We observe that across all datasets, the classes are not easily separable.

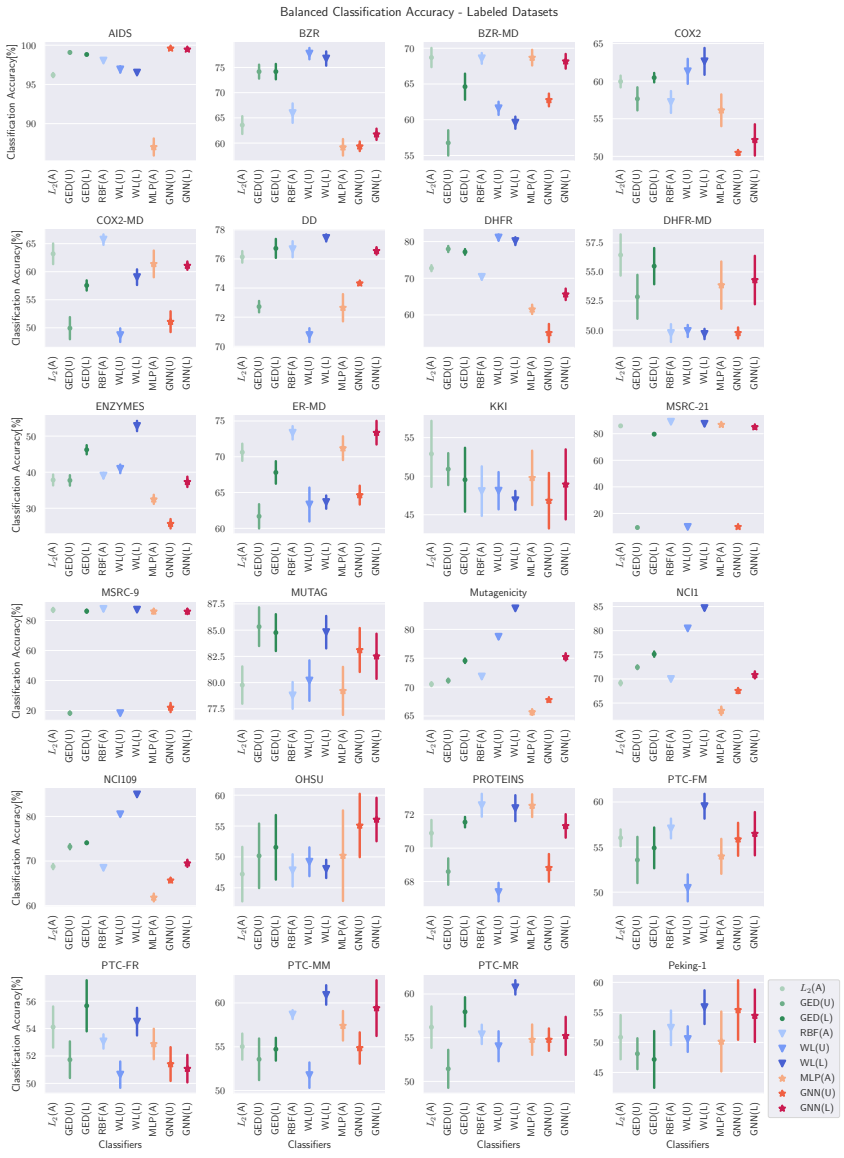


Fig. 3.59: Balanced classification accuracy of the three types of classifiers (k-NN, SVM, NN) achieved in the three tested configurations (L, U, A) across all node-labeled graph datasets.

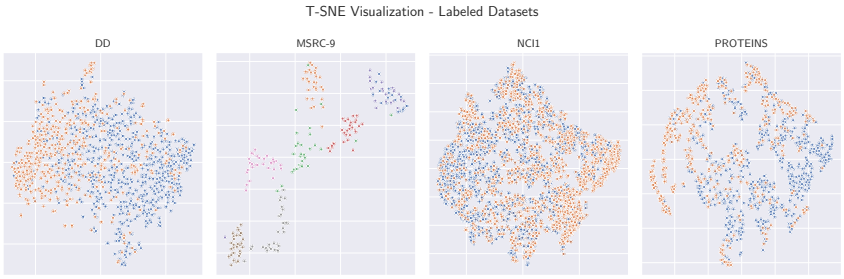
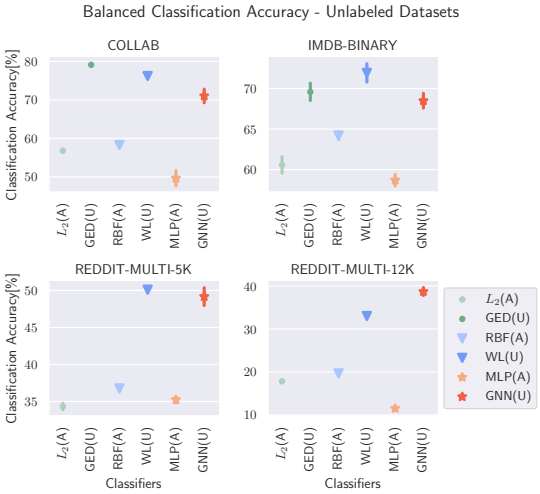


Fig. 3.60: T-SNE visualization of the vector representation of the node-labeled graph datasets.

(a) Balanced classification accuracy of unlabeled graph datasets.

Fig. 3.61: Balanced classification accuracy (with and without node degree information) of the three types of classifiers (k-NN, SVM, NN) achieved in the three tested configurations (L, U, A) across all unlabeled graph datasets.



(b) Balanced classification accuracy of unlabeled graph datasets with node degree information.

Fig. 3.61: Balanced classification accuracy (with and without node degree information) of the three types of classifiers (k-NN, SVM, NN) achieved in the three tested configurations (L, U, A) across all unlabeled graph datasets.

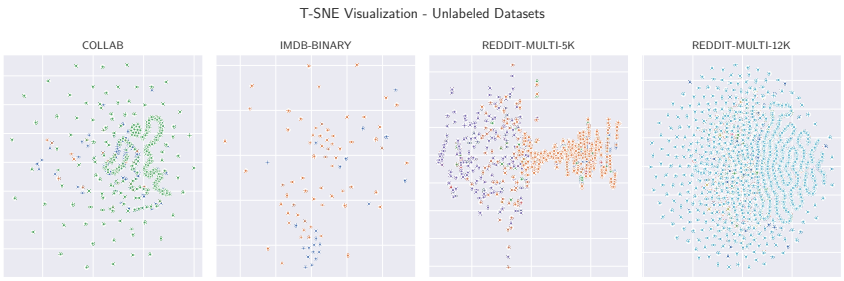


Fig. 3.62: T-SNE visualization of the vector representation of unlabeled graph datasets.

3.5.4 *Dataset Selection*

Comparing two classification algorithms is not a trivial task due to the risk of committing type I or type II errors. Type I errors occur when the null hypothesis is wrongly rejected even though it is true, whereas type II errors occur when the null hypothesis is not rejected even though it is false. To address this issue, the authors of [154] conduct an empirical evaluation of multiple statistical tests and conclude that the 10-time repeated 10-fold cross-validation test is the most effective. This test involves all 100 individual systems to estimate the mean and variance of the accuracy with 10 degrees of freedom (making it conceptually simple to use).

We apply this statistical test on the node-labeled graph datasets, to conduct a comparison between the statistical classifiers that use the aggregated node labels only and their graph-based counterparts (i.e., the classifiers that use both node labels and graph structure). In particular, this analysis counts how often the graph-based methods outperform their vector-based counterpart.

Regarding the results in Fig. 3.63, we observe that on the following six datasets, all of the three graph-based approaches outperform the corresponding vector-based approaches.

- DHFR
- ENZYMES
- MUTAG
- MUTAGENICITY
- NCI1
- NCI109

On the following four datasets, two of the three graph-based methods outperform the vector-based methods.

- AIDS
- BZR
- DD
- OHSU

Finally, in five cases, only one of the graph-based methods outperforms the vector-based method (this is most frequently observed when comparing the two kernel approaches).

- COX2

- ER-MD
- PTC-FM
- PTC-MM
- PTC-MR

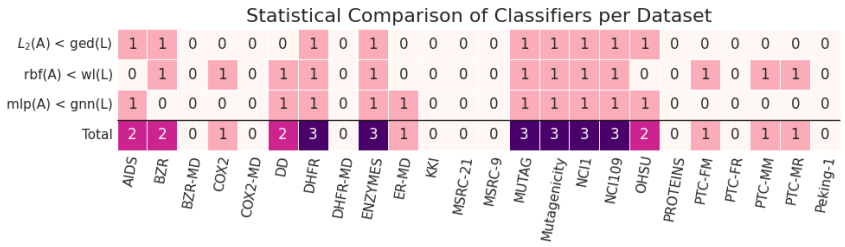


Fig. 3.63: Comparison of the statistical classifiers that use the aggregated labels (i.e., L_2(A), RBF(A), and MLP(A) and the graph-based approaches that operate on the original graphs with both structure and node labels (i.e., GED(L), WL(L), GNN(L)).

On the remaining nine node-labeled datasets, the graph-based methods show no significant advantage over the corresponding vector-based methods. On the total of 24 datasets, the GED and GNN methods show superiority over the corresponding vector-based methods in 9 cases, while the WL-graph kernel performs better than the RBF-kernel in 14 cases.

Concerning the unlabeled datasets, the graph-based method consistently outperform the naïve embedding across all datasets. Hence, we opt not to explicitly present the classifier comparison for the unlabeled datasets.

Based on the results shown in Fig. 3.63, we can make the following assumption. First, the new graph-based approximation algorithms that we propose and that aim to reduce computation time should, most of the time, be tested on datasets where the graph-based classification methods outperform the vector-based method in at least two, ideally three, cases. This recommendation is based on the fact that even the fastest approximation method is likely to be slower than the baseline feature vector method presented here.

Based on this analysis, we now outline the dataset selection used throughout the rest of the present thesis. Table 3.30 and 3.31 provide an overview of the labeled and unlabeled datasets used to evaluate the proposed graph reduction methods (described in Chapter 4, 5, and 6), respectively.

Note that out of the 28 datasets initially presented, nine datasets are excluded from further consideration in this thesis. Specifically, the majority of these datasets exhibited poor performance, as indicated in Fig. 3.63 – for instance ER-MD, KKI, MSRC-9, MSRC-21, the four PTC datasets, and PEKING-1.

Table 3.30: Selection of the labeled datasets per chapter.

Dataset	**Used in Chapter**
AIDS	4
BZR	5, 6
BZR-MD	6
COX2	6
COX2-MD	6
DD	6
DHFR	5, 6
DHFR-MD	6
ENZYMES	4, 5, 6
ER-MD	-
KKI	-
MSRC-9	-
MSRC-21	-
MUTAG	6
MUTAGENICITY	4, 5, 6
NCI1	4, 5, 6
NCI109	5
OHSU	6
PROTEINS	4, 5, 6
PTC-FM	-
PTC-FR	-
PTC-MM	-
PTC-MR	-
PEKING-1	-

Table 3.31: Selection of the unlabeled datasets per chapter.

Dataset	**Used in Chapter**
COLLAB	5
IMDB-BINARY	4, 6
REDDIT-MULTI-5K	5
REDDIT-MULTI-12K	5

Graph Reduction by means of Centrality Measures

4

> Toute connaissance est une réponse à une question. S'il n'y a eu de question, il ne peut y avoir de connaissance scientifique. Rien ne va de soi. Rien n'est donné. Tout est construit.
>
> *La formation de l'esprit scientifique* (1938), Gaston Bachelard

4.1 Introduction

As thoroughly described in Chapter 2, graph matching [4; 5] is one of the most fundamental problems in graph-based pattern recognition. Considering the influence of graph matching in miscellaneous pattern recognition applications, it is crucial to develop and research efficient procedures for this task. The *Graph Edit Distance* (GED) [86] (also described in Chapter 2), introduced approximately 40 years ago, remains one of the most versatile and robust graph matching models available. Unfortunately, it is well known that the computation of GED is an $\mathcal{NP}$-complete problem in general [27]. Therefore, comparing large graphs is often not possible (or at least computationally expensive) and thus not feasible for real-time pattern analysis.

Over the last decade, however, different approximations to reduce the runtime of GED have been proposed [11; 155; 156; 157; 158]. These approximation span a range of techniques, from using a beam search procedure instead of A* to narrow down the search space [156; 157], to employing linear programming methods for computing the upper and lower bounds of GED [158], and ultimately reducing the GED computation to a linear sum

assignment problem (LSAP) [11; 155]. All of these approximations substantially reduce the computational complexity of GED. Yet, even with these polynomial time algorithms, the application of GED remains problematic – in particular for large graphs with many nodes and/or many edges.

The present chapter aims to further improve the computation cost of GED. However, we do not suggest a new algorithmic approximation but introduce a novel approximation on the data side, i.e., we propose and investigate novel graph reduction method. That is, besides using approximations, we reduce the matching time by working with reduced versions of the graphs rather than the original graphs.

In the present chapter, the proposed graph-based pattern recognition framework operates directly in the graph domain. For this reason, our graph classification method is based on the *BP-GED* [11] approximation (termed GED from now on). This GED approximation is joined with a distance-based classifier such as the *k-Nearest Neighbors classifier* (k-NN) as it has shown reasonable classification accuracies on diverse classification tasks (e.g., [159; 160]).

Note that the present chapter summarizes three preliminary conference papers [42; 43; 44] and one journal paper [45] and is organized as follows. In Section 4.2, we explain in detail our graph reduction process. That is, we show how to map graphs from the original graph domain to a graph subspace where the graph matching is eventually conducted. In Section 4.3, we demonstrate the effectiveness of our proposed method. That is, we compare both matching time and classification accuracy obtained on the reduced graphs (with different sizes) with the corresponding results obtained in the original graph domain. In Section 4.4, we propose and research an extension of the graph reduction strategy to address the problem of the loss of information when nodes are discarded. That is, we propose and evaluate two distinct modifications to the classification procedure to mitigate this effect. The goal of Section 4.5 is to improve the classification performance of a k-NN classifier coupled with GED. To this end, we define and evaluate a novel method that extracts extra information out of different reduced versions of the original graphs and combines this information in a multiple classifier system. Finally, in Section 4.6, we summarize the key findings of the proposed approaches and conclude this chapter.

4.2 Graph Reduction Using Centrality Measures

The reduction method proposed in this chapter is based on network's node *centrality measures* [150] and employs the strategy of graph summarization. In Subsection 4.2.1, we present centrality measures – borrowed from the field of network analysis [150] – to decide which nodes to omit from the original graphs. In Subsection 4.2.2, we present our graph reduction procedure. Finally, in Subsection 4.2.3, we present a qualitative analysis of the reduced graphs obtained after applying our proposed graph reduction method.

4.2.1 *Centrality Measures*

Centrality measures indicate how important a node in a graph is by quantifying the contribution of each node to the graph connection. The idea is imported from *social network analysis*, where it helps to comprehend the social network interactions [150]. There is no universal definition for the influence of a node, and thus there exists a wide variety of centrality measures that depend on the problem to be solved [150].

Roughly speaking there are two categories of centrality measures available, viz. *Degree-based* and *Shortest-path based* measures. The degree-based methods use the degree property of the nodes, i.e., how many edges are connected to a node, to derive their centrality score. Meanwhile, the shortest-path based algorithms compute a node's centrality score by counting the number of paths between any two pairs of nodes that pass by it.

In the present chapter, we focus on two popular centrality measures stemming from both categories, namely *PageRank* (degree-based) [161] and *Betweenness* (shortest-path based) [162]. Note that our graph reduction framework is not only limited to those two measures. That is, any other centrality measures could be used as well.

PageRank [161] is a centrality measure that is originally used to rank search engine results and has the advantage to be content-independent. The basic concept relies on the fact that a node's importance increases by having connections with other nodes that are themselves important. That is, the importance of a node is thus proportional to the sum of the scores of the nodes in its neighborhood.

This naive formulation has, however, an undesirable property; if an important node points to many other nodes then its high-centrality will be spread among all its neighbors. That means they are all going to be consid-

ered as important by association (which is often not a desirable behavior). Hence, the authors of [161] propose to dilute the influence of an influential node proportionally to the number of its neighbors. Formally, PageRank is defined as

$$\boldsymbol{x} = \alpha \boldsymbol{A}\boldsymbol{D}^{-1}\boldsymbol{x} + \beta \mathbf{1}\,, \tag{4.1}$$

where vector $\boldsymbol{x}$ contains the n PageRank scores for all nodes $\{v_1, \ldots, v_n\}$ of a given graph $G = (V, E)$. Parameter α plays the role of a damping factor (there is no clear theory to choose its values, it was primarily set to 0.85). The matrix $\boldsymbol{A}$ is the adjacency matrix of graph $G = (V, E)$ and $\boldsymbol{D}$ is the diagonal matrix with elements $\boldsymbol{D}_{ii} = \max(d_i^{out}, 1)$, where value d_i^{out} corresponds to the outdegree of the i-th node ($\max(\cdot, \cdot)$ is used in case a node has zero neighbors). In the second part of Eq. 4.1, β is an additive constant (conventionally set to 1).

Betweenness [162] is a centrality measure that counts the number of times a node lies on the shortest paths connecting pairs of nodes. In a graph with flowing information, it expresses, on average, the number of time messages passes between each pair of nodes. Nodes with a high Betweenness score have significant importance within a graph by the control of the information flowing between other nodes. Formally, the Betweenness score x_i for the i-th node $v_i \in V$ of a given graph $G = (V, E)$ is defined by

$$x_i = \sum_{u,v \in V} n_{uv}^i, \tag{4.2}$$

where

$$n_{uv}^i = \begin{cases} 1, & \text{if node } i \text{ is on the shortest-path} \\ & \text{between node } u \text{ and } v \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

In its primal formulation (see Eq. 4.2), Betweenness scales with the number of pairs of nodes. One may be interested to re-scale the Betweenness score to be between 0 and 1. This can readily be done by dividing the score by $(|V| - 1)(|V| - 2)$, where $|V|$ is the number of nodes in graph G.

4.2.2 *Creation of Reduced Graphs*

Once the node centrality scores are computed for each node (either with PageRank or with Betweenness), one can sort them according to their cen-

trality from the least to the most important one. The higher the score for a node, the more significant it is and thus should remain in the graph. Hence, we use the centrality score to sort the nodes from the least to the most influential ones. With a reduction factor $\lambda \in]0, 1]$ we are then readily able to select the most influential $\sigma = \lfloor \lambda |V| \rfloor$ nodes in the graph, while the other nodes and their incident edges can be removed from the graph. The reduction factor λ is a user-defined parameter and can be seen as the percentage of remaining nodes of the original graph.

That is, if we set $\lambda = 0.8$, then around 80% of the nodes remain in the reduced versions of the graphs and 20% of the nodes are omitted. We can now arbitrarily vary the reduction factor λ from 1.0 to 0.0 by different step-sizes to obtain differently sized graphs out of one source graph. By using either PageRank or Betweenness as selection criterion with $\lambda \in \{1.0, 0.8, 0.6, 0.4, 0.2\}$, we thus build for both centrality measures four different graph subspaces from the original graph domain. Note that with $\lambda = 1.0$ all the nodes remain in the reduced graph which obviously corresponds to the original graph.

In practice, we stop removing nodes from G whenever the number of nodes falls below a fixed threshold (in our experiments we set this threshold to 5 nodes). Note, moreover, that if $c > 1$ nodes have the same score and $\rho = \lfloor (1 - \lambda)|V| \rfloor < c$ nodes should be actually removed in the next step, the ρ omitted nodes are randomly selected from the c candidates.

For each node, the centrality score is initially computed once on the original graphs. Based on these scores we build all graph subspaces at once. That is, we do not recompute the centrality measures on a specific reduction level λ before continuing to the next lower level. Actually, preliminary experiments show no significant differences in the reduced graphs whether or not the centrality scores are recomputed after each graph reduction.

In Fig. 4.1, we show an original graph $G = (V, E)$ with $|V| = 14$ and two reduced versions of the graph with $\lambda = 0.8$ and $\lambda = 0.4$. The number of nodes that remain in the graph with $\lambda = 0.8$ and $\lambda = 0.4$ is $\lfloor 0.8 \cdot 14 \rfloor = 11$ and $\lfloor 0.4 \cdot 14 \rfloor = 5$, respectively.

In the remainder of the chapter, we term a reduced graph (with reduction factor λ) as $G_\lambda = (V_\lambda, E_\lambda)$. When reducing all graphs in a given dataset of size N, we obtain a reduced graph subspace $\mathcal{G}_\lambda = \{G_\lambda^{(1)}, \ldots, G_\lambda^{(N)}\}$. Repeating this process with different reduction factors $\lambda_1, \ldots, \lambda_r$ we obtain r reduced graph subspaces $\mathcal{G}_{\lambda_1}, \ldots, \mathcal{G}_{\lambda_r}$.

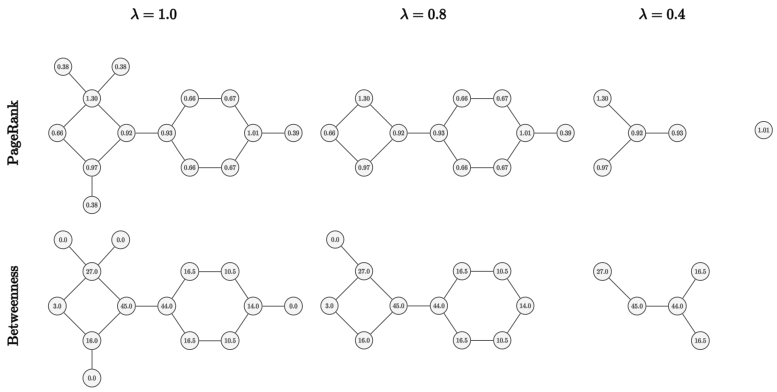


Fig. 4.1: An example of our reduction approach on a synthetic graph with both node selection criteria (i.e., PageRank and Betweenness) and $\lambda \in \{0.8, 0.4\}$. (For better visibility, the PageRank scores are scaled up by factor 10.)

4.2.3 *Qualitative Results*

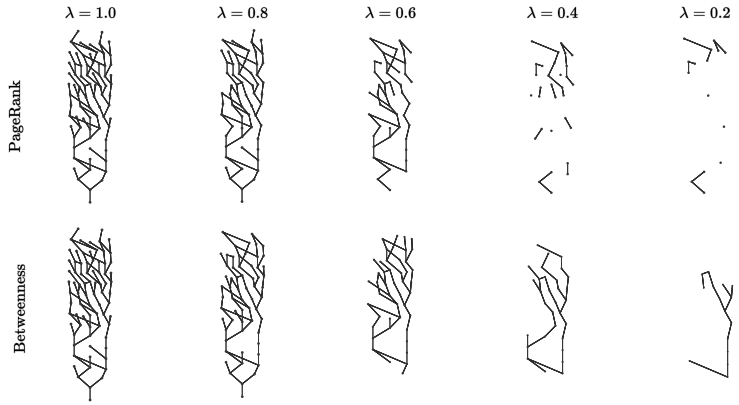
We start our evaluation with a qualitative investigation on the AIDS dataset[1] (see Chapter 3). Fig. 4.2 shows original and reduced graphs of molecular compounds from the AIDS dataset. We vary the parameter λ to all reduction levels (including $\lambda = 1.0$ which corresponds to the original graph domain).

We notice variations between the two centrality measures. By reducing the graph size based on the PageRank selection method, we tend to preserve the intra-community nodes. The communities are kept together while being separated from one another. Roughly speaking, PageRank tends to produce reduced graphs with large numbers of connected components. This effect is particularly apparent in Fig. 4.2 (a), with $\lambda = 0.6$.

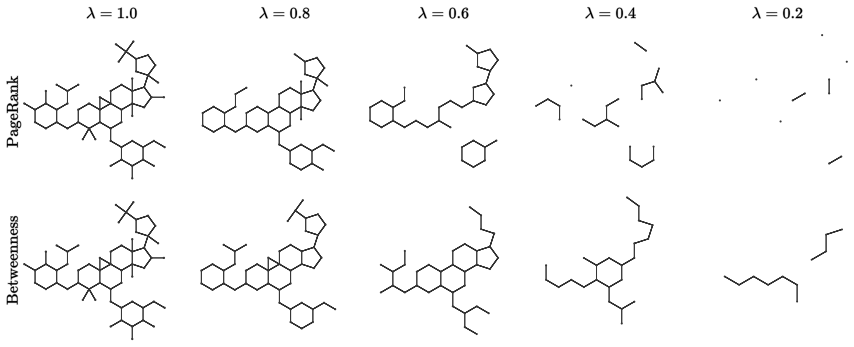
On the other hand, by deleting nodes upon Betweenness values, we keep the backbone structure of the graph. That is, we observe that the main paths in the graph form communities and are kept when discarding nodes from the graph. For instance, in Fig. 4.2 (a) and (c) with $\lambda = 0.2$ we observe that the main paths in the graph form communities and are kept in the graph. Simultaneously, the external nodes from the communities

[1] On the other datasets similar observations can be made (see Appendix B.1).

tend to be omitted using this specific centrality measure.

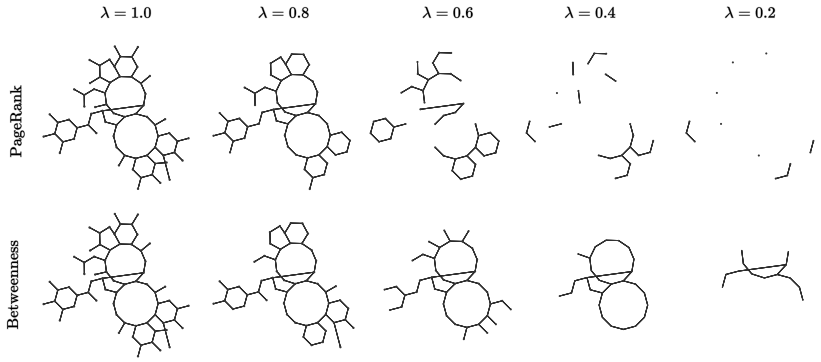


(a) Reduced molecular compound from AIDS



(b) Reduced molecular compound from AIDS

Fig. 4.2: Example graphs from AIDS with both reduction techniques (PageRank, Betweenness) and all reduction levels λ.



(c) Reduced molecular compound from AIDS

Fig. 4.2: Example graphs from AIDS with both reduction techniques (PageRank, Betweenness) and all reduction levels λ.

4.3 Graph Matching on Reduced Graphs

In the present section, the goal is to research how GED and its computation are affected when working on reduced graphs. Our experimental evaluation consists of two different parts. First, in Section 4.3.1 we perform GED computations on the original graphs and all graph subspaces and compare the run times with each other. Moreover, we compare the classification accuracies achieved on different reduction levels with both the accuracy achieved on the original graphs and on randomly reduced graphs. Second, in Section 4.3.2, we show and discuss scatter plots that illustrate the correlation between the original distances and the corresponding distances in the graph subspaces.

We use six graph datasets presented in Table 4.1 to conduct empirical evaluations. We split all datasets into three disjoint subsets for training, validation, and testing as follows. We split the graphs from the IAM graph repository (i.e, AIDS and MUTAGENICITY) according to the proposed splitting of the benchmark. The NCI1 dataset is split to match the size of the three sets of the MUTAGENICITY dataset. The other datasets are divided w.r.t. the 60-20-20% split rule for training, validation, and test sets, respectively. In this chapter, we also report the standard classification accuracy that allows us to measure the performance of the different proposed methods on the classification task.

In Table 4.1, we summarize the number of graphs per split used for training, validation and testing and present some other statistical properties for each graph dataset, viz. the number of graphs, the number of classes, and the average number of nodes and edges (for further information about those datasets we refer to Chapter 3).

Table 4.1: Properties of the graph datasets. We show the size of the graph datasets ($|\mathcal{G}|$) with the number of graphs in the training, validation, and test set ($|\mathcal{G}_{tr}|$, $|\mathcal{G}_{va}|$, $|\mathcal{G}_{te}|$), the number of classes ($|\Omega|$) and the average number of nodes and edges per graph ($\emptyset|V|$, $\emptyset|E|$).

	Graph Dataset Property				
Dataset	$\boldsymbol{\lvert\mathcal{G}\rvert}$	($\boldsymbol{\lvert\mathcal{G}_{tr}\rvert}$, $\boldsymbol{\lvert\mathcal{G}_{va}\rvert}$, $\boldsymbol{\lvert\mathcal{G}_{te}\rvert}$)	$\boldsymbol{\lvert\Omega\rvert}$	$\boldsymbol{\emptyset\lvert V\rvert}$	$\boldsymbol{\emptyset\lvert E\rvert}$
AIDS	2,000	(250, 250, 1,500)	2	9.5	10.0
ENZYMES	600	(360, 120, 120)	6	32.6	62.1
IMDB-BINARY	1,000	(600, 200, 200)	2	19.8	96.5
MUTAGENICITY	4,337	(1,500, 500, 2,337)	2	30.3	30.8
NCI1	4,110	(1,500, 500, 2,110)	2	29.9	32.3
PROTEINS	1,113	(660, 220, 223)	2	39.1	72.8

In all of the experiments of the present section, we first reduce all graphs (stemming from training, validation, and test sets) to have $\sigma = \lfloor \lambda |V| \rfloor$ remaining nodes for all reduction levels λ. For the second and third experiment, we then compute the required distance matrices between the condensed graphs using GED at each level λ. For GED computation, we apply unit costs for node and edge insertions/deletions. The cost for node substitutions with unequal node symbols is set to a constant cost of 2.

On all datasets and reduction levels λ, we individually optimize parameter $\alpha \in [0, 1]$ that weights the relative importance of node and edge edit operation costs.

4.3.1 *Computation Time and Classification Accuracy*

The goal of this evaluation is twofold. First, we aim to verify whether or not our reduction process deteriorates the classification accuracy when compared to the one obtained with the original graphs. We choose a k-

Nearest Neighbor (k-NN) as our classification method[2]. The second goal of this evaluation is concerned with the runtimes. That is, we aim at verifying the expected substantial speed-up of the computation time due to the smaller number of nodes in the reduced graphs.

In order to verify the benefit of the proposed graph reduction using elaborated centrality measures, we also compare the novel method with a random subsampling of the graphs. For this experiment, we run our algorithm 10 times with random node selection for each reduction level in order to alleviate the variance of accuracies of the random node selection.

In Fig. 4.3 we display the classification accuracy per reduced graph level for all datasets (including $\lambda = 1.0$). For the control experiment (random selection), we plot the 95% confidence interval from the 10 runs for each reduced graph level. All computations have been conducted in a parallel configuration on an *AMD Ryzen 9 5900X* with 12 cores.

As expected, the classification accuracy of the k-NN generally decreases when reducing the size of the graphs. This is, most probably, due to the loss of information when discarding nodes. Note, however, that the classification accuracy remains relatively stable on the AIDS dataset for all reduction levels and for MUTAGENICITY up to level $\lambda = 0.6$. We can observe that the classification accuracy obtained with our reduction methods (both PageRank and Betweenness) are clearly better than the random selection on all molecule graphs (AIDS, MUTAGENICITY, and NCI1).

For the PROTEINS and ENZYMES graphs, the reduction based on Betweenness gives better classification accuracies than the random deletion for all reduction levels. Yet, the reduced graphs based on PageRank achieve quite similar accuracies as offered with the randomly reduced graphs. On the IMDB graphs, we do not observe a clear benefit of our reduction scheme compared to a random node selection.

Despite the general drop in the classification accuracy using the reduced graphs rather than the original graphs, we can conclude that the accuracies achieved with reduced graphs using our method remains comparable with the accuracies achieved with the original graphs. Moreover, we can state that using elaborated reduction methods rather than using random subsampling is clearly beneficial.

The great benefit of the novel method is, of course, the computation time that steadily decreases on the reduced graphs. In Table 4.2, we observe an evident speed up of the runtimes on the reduced graphs. Note that the

[2]Parameter $k \in \{1, 3, 5, 7\}$, that defines the number of neighbors considered in the k-NN classifier, is optimized on all datasets and reduction levels individually.

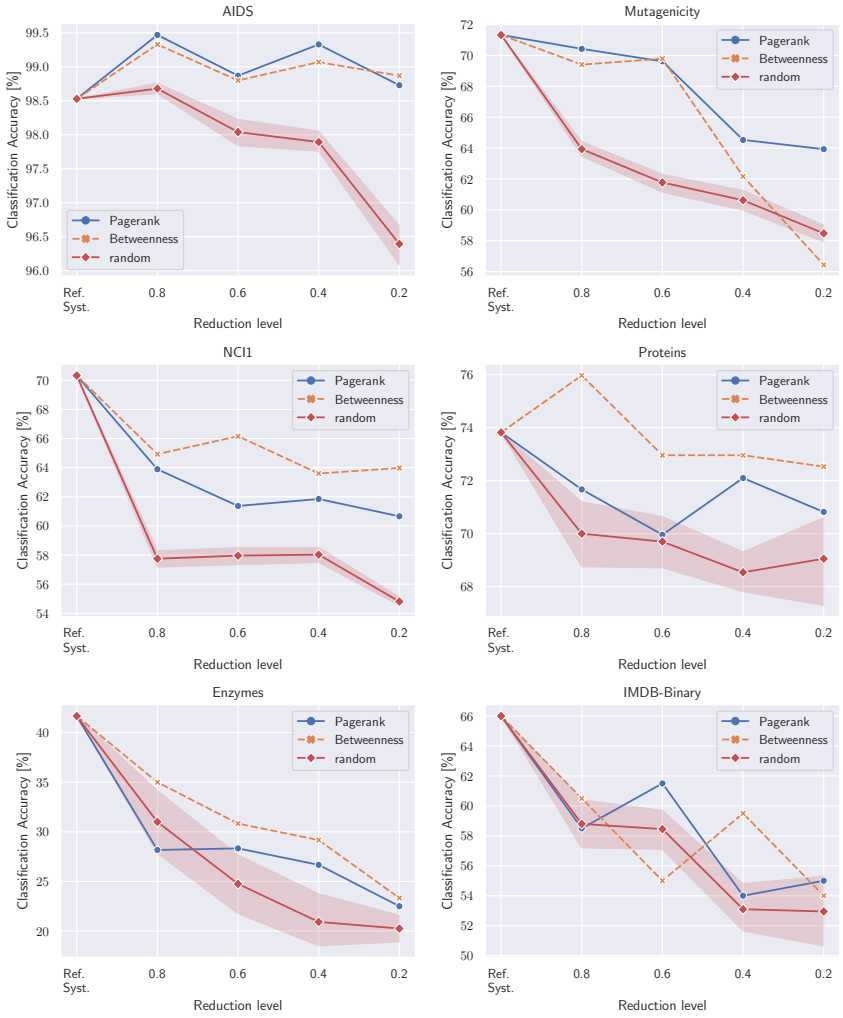


Fig. 4.3: Classification accuracies for all datasets and all reduction levels (including $\lambda = 1.0$) for our reduction methods (PageRank and Betweenness) and a random subsampling.

runtimes are averaged from 10 runs with a 95% confidence interval.

Table 4.2, shows runtime improvements up to a factor of four (with $\lambda = 0.4$ on MUTAGENICITY, for instance). Using reduction level $\lambda = 0.6$, we can report that the runtimes on all datasets are approximately halved

Table 4.2: Computation time in seconds on the test set using the original graphs (i.e., $\lambda = 1.0$) and all evaluated reduction levels λ.

Dataset	Reduction Level λ				
	$\lambda = 1.0$	$\lambda = 0.8$	$\lambda = 0.6$	$\lambda = 0.4$	$\lambda = 0.2$
AIDS	9.93 ± 0.25	7.25 ± 0.30	5.59 ± 0.13	4.00 ± 0.13	2.78 ± 0.21
MUTAGENICITY	63.27 ± 1.62	46.08 ± 1.55	33.23 ± 0.95	21.91 ± 0.55	15.18 ± 0.72
NCI1	51.41 ± 1.10	36.72 ± 0.95	25.85 ± 0.71	17.89 ± 0.68	12.62 ± 0.88
PROTEINS	7.09 ± 0.37	4.67 ± 0.25	3.09 ± 0.16	2.02 ± 0.21	1.18 ± 0.24
ENZYMES	2.18 ± 0.10	1.41 ± 0.10	1.11 ± 0.14	0.61 ± 0.04	0.35 ± 0.05
IMDB-BINARY	4.32 ± 0.15	2.91 ± 0.23	1.94 ± 0.21	1.08 ± 0.13	0.61 ± 0.11

when compared with the runtimes on the original graphs (with $\lambda = 1.0$).

Note that the matching times for PageRank, Betweenness and randomly reduced graphs are the same. Once the graphs are reduced, they have the same size no matter which reduction method is used and thus the matching time is equivalent.

4.3.2 *GED Quality Measure*

GED is often employed in conjunction with distance-based classifiers. Hence, we aim at verifying whether the distances obtained with the reduced graphs make sense. In order to verify this, we show scatter plots that indicate the correlation between the original graph distance and the distance obtained in the reduced graph space with PageRank (equivalent plots are obtained with Betweenness are shown in Appendix B.2). In particular, for each pair of graphs a dot is plotted in Fig. 4.4. Each dot represents the distance in the original graph space on the x-axis and the corresponding distance on the reduced graphs on the y-axis.

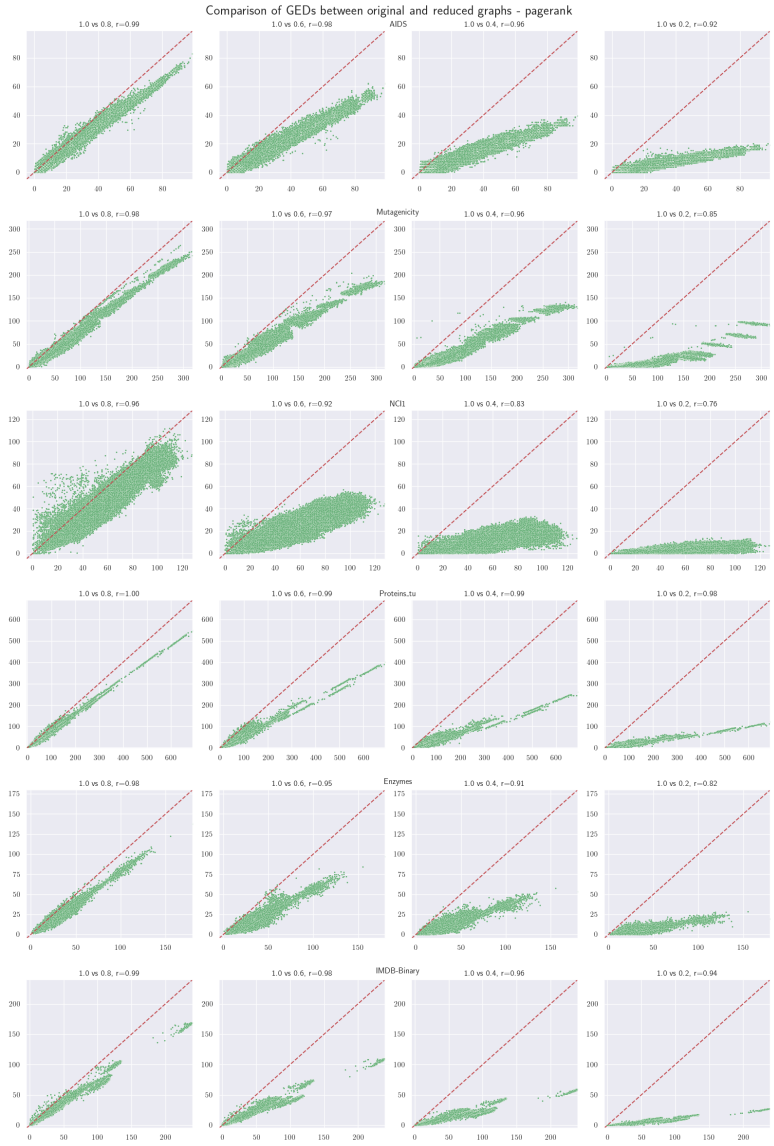


Fig. 4.4: Pairwise distances between graphs in the original graph space and their resulting counterpart in the reduced graph domain using PageRank on all datasets.

Two main observation can be drawn from these plots. First, the distances in the original and reduced graph space correlate quite well. In general, if the distance is large between two graphs in the original graph space then the distance is also relatively large in the reduced graph space. Vice versa, if the distance is small in the original graph space, the corresponding distance is small as well in the reduced graph space. However, the more the graphs are reduced the more the overall distances decrease, reducing the correlation which makes it more complicated for a distance-based classifier to accurately predict the correct class of the graphs.

The second observation is that there are some points above the main diagonal. That means, the distances in the reduced graph space (especially for $\lambda = 0.8$) overestimate the GED compared to its corresponding counterpart in the original graph domain. At first glance this seems to be counter-intuitive. However, an in-depth examination of those pairwise graphs shows that this GED overestimation is a "natural" effect. Due to the reduction, the approximate GED finds other assignments of local substructures for the nodes, and that might lead – for some pairs of graphs – to greater distances than in the original graph space. An illustration of this behavior is given in Fig. 4.5.

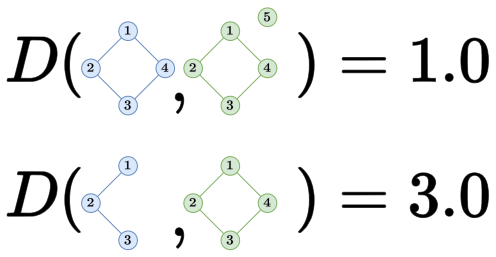


Fig. 4.5: Example of the GED overestimation after graph reduction.

On the upper part of Fig 4.5 the original GED is shown, where just deleting one node allows to match both graphs. On the lower part of Fig 4.5 GED is displayed after graph reduction. In this setup, the overall GED rises from 1.0 to 3.0 in order to incorporate the updated structural assignment (inserting one node and two edges).

4.4 Two-Step Graph Classification

As demonstrated in Section 4.3, some limitation appears when working with systematically reduced graphs – specifically, it has a disadvantageous effect on the classification accuracy. To counter this negative effect, one could modify the standard classification process into a two-step classification scheme. For the sake of conciseness, we use the reduced graphs described in Section 4.2 using the PageRank centrality measure only.

In the proposed framework the size-reduced and original graphs are now employed in two separate steps. The basic idea of the first step is to conduct all matchings on the strongly reduced graphs. Then, in the second step, we conduct as few matchings as possible on the original graphs. Hence, our method can also be interpreted as a coarse-to-fine approach that starts on rather coarse representations and eventually continues on the more fine-grained graph representations – similar to the approach presented in [121].

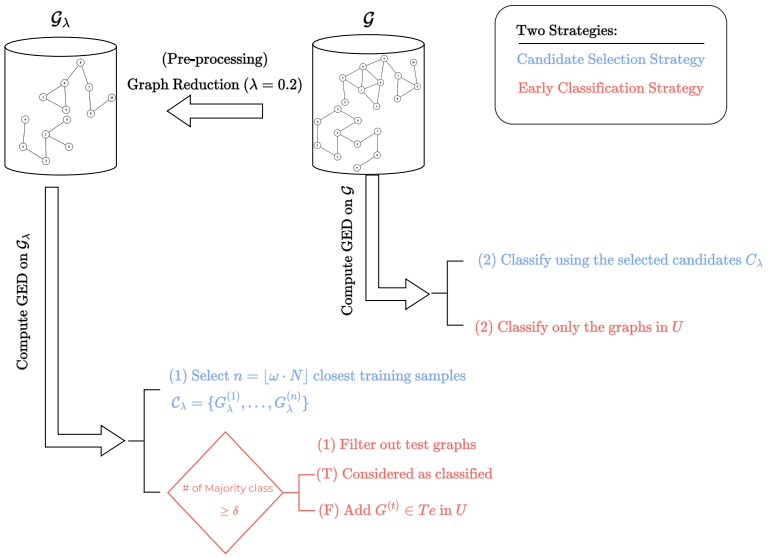


Fig. 4.6: Illustration of the two steps of two different classification strategies (shown in blue and red color.

However, our approach differs in three major parts with [121]. First, we use node centrality measures (rather than node clustering) for graph reductions. Second, we employ only two levels of hierarchy (since we pri-

marily aim at speeding up the classification process). Third, we propose two different strategies for the crucial decision on how to proceed after step 1 (a visual summary of both strategies is presented in Fig. 4.6).

- The first strategy is to select appropriate candidates in the reduced graph space for further processing (described in blue in Fig. 4.6).
- The second strategy is to accept the classification obtained in the reduced graph space if the classifier is confident enough (described in red in Fig. 4.6).

Both strategies are described in greater detail in the next two subsections.

Note that, regardless of the strategy actually employed, the proposed two-step classification scheme allows us to trade-off between runtime and classification accuracy. That is, pruning numerous graphs in the first step allows a faster computation time but possibly deteriorates the classifier's accuracy. Contrariwise, keeping many graphs for the second step possibly allows better classification accuracy but might increase the computation time.

4.4.1 *Candidate Selection Strategy*

The aim of this strategy is as follows. Based on the matching information obtained on the reduced graphs, we keep in a first step the nearest training graphs for each test graph. We term these nearest neighbors – actually used in step 2 – as *candidates*. We define a parameter ω that defines the relative amount of training graphs that are selected as candidates. With $\omega = 0.1$, for instance, we select 10% of the nearest training graphs as candidates for each test graph.

The second step of the classification process is based on the original graphs. That is, the test graphs are matched with the original graphs that correspond to the selected candidates from step 1. Intuitively, step 1 acts as a filter that pre-selects plausible candidates from the training set for further and more precise investigations during step 2.

The complete process – termed *CandSel* from now on – is formalized in Alg. 2. The algorithm takes as parameters a test graph in two representations $G^{(t)}$ and $G_\lambda^{(t)}$ (the original and the size-reduced graph), as well as N original training graphs $\mathcal{G} = \{G^{(1)}, \ldots, G^{(N)}\}$ and their corresponding reduced versions $\mathcal{G}_\lambda = \{G_\lambda^{(1)}, \ldots, G_\lambda^{(N)}\}$. Note that we use a k-NN classifier on line 8 of the algorithm. However, any other distance or similarity-based

classification could be employed as well.

Algorithm 2: CandSel

Input: Parameter ω, training graphs $\mathcal{G} = \{G^{(1)}, \ldots, G^{(N)}\}$ and their reduced counterparts $\mathcal{G}_\lambda = \{G_\lambda^{(1)}, \ldots, G_\lambda^{(N)}\}$, graph to classify from test set $G^{(t)}$ and its reduced version $G_\lambda^{(t)}$

STEP 1
$n = \lfloor \omega \cdot N \rfloor$ `// number of candidates`
for $i \in \{1, \ldots, N\}$ **do**
  compute $\mathrm{GED}(G_\lambda^{(i)}, G_\lambda^{(t)})$
$\mathcal{C}_\lambda = \{G_\lambda^{(1)}, \ldots, G_\lambda^{(n)}\}$ `//` n `training graphs in` $\mathcal{G}_\lambda$ `closest to` $G_\lambda^{(t)}$
$\mathcal{C} = \{G^{(1)}, \ldots, G^{(n)}\}$ `// corresponding graphs in` $\mathcal{G}$
STEP 2
Classify $G^{(t)}$ with the aid of the selected candidates $\mathcal{C} = \{G^{(1)}, \ldots, G^{(n)}\}$

4.4.2 *Early Classification Strategy*

In this second strategy, we first apply a k-NN classification on the reduced graphs. All graphs for which the classification is — more or less — confident, are directly classified without further processing in step 2. Formally, we measure the confidence of each decision by means of the number of neighbors k' among the k-nearest neighbors that stem from the same class (with $k' \leq k$). If k' is greater than, or equal to, a certain threshold δ we consider the class prediction as confident enough to be accepted.

Considering a 5-NN, for instance, and we set $\delta = 4$, then at least four of the nearest neighbors have to stem from the same class so that the graph is classified in the first step[3]. Note that this strategy directly depends on the classifier actually employed. That is, when this strategy is used in conjunction with another distance-based classifier, another metric for the confidence has to be defined first.

For any reduced graph $G_\lambda^{(t)}$ that is not classified in the first step (due to too-low classification confidence), the classification is conducted on its

[3]Note that for a binary classification task with a 5-NN, it makes no sense to set $\delta \leq 3$ because all samples would be immediately classified in the first step.

original counterpart $G^{(t)}$. That is, we have to compute all distances from $G^{(t)}$ to all original training graphs to apply a final k-NN classification. We formalize this procedure – termed *EarlyClass* from now on – in Alg. 3.

Algorithm 3: EarlyClass

Input: Parameter δ, training graphs $\mathcal{G} = \{G^{(1)}, \ldots, G^{(N)}\}$ and their reduced counterparts $\mathcal{G}_\lambda = \{G_\lambda^{(1)}, \ldots, G_\lambda^{(N)}\}$, graph to classify from test set $G^{(t)}$ and its reduced version $G_\lambda^{(t)}$

STEP 1
for $i \in \{1, \ldots, N\}$ **do**
  compute $\mathrm{GED}(G_\lambda^{(i)}, G_\lambda^{(t)})$
$\mathcal{C}_\lambda = \{G_\lambda^{(1)}, \ldots, G_\lambda^{(k)}\}$ // k `closest graphs to` $G_\lambda^{(t)}$ `in` $\mathcal{G}_\lambda$
if *number of graphs of majority class in* $\mathcal{C}_\lambda \geq \delta$ **then**
  Classify $G^{(t)}$ as the accepted classification of $G_\lambda^{(t)}$
else
  Add $G^{(t)}$ in U
STEP 2
Classify the remaining $G^{(t)} \in U$ with the aid of the original training graphs $\mathcal{G} = \{G^{(1)}, \ldots, G^{(N)}\}$

4.4.3 *Experimental Evaluation*

The purpose of our experimental evaluation is twofold. First, we aim at investigating the reduction of the computation time actually possible with the proposed framework. Second, we want to evaluate whether or not the classification accuracy can be maintained when applying the two strategies in our two-step classification procedure. Hence, we compare both the runtime and the classification accuracy obtained by our novel methods with a standard classification method, viz. a k-NN classifier that operates on the original graphs only.

4.4.3.1 *Candidate Selection*

Table 4.3 shows both the classification accuracy and the runtime for the reference system (i.e., a standard k-NN classifier operating in the original graph space) and the novel method *CandSel* with $\omega \in \{0.20, 0.10, 0.05\}$.

On all datasets, we observe that the classification accuracy generally

Table 4.3: Classification accuracy and runtime (in seconds) obtained with a k-NN classifier on the original graphs (Reference System) and the results obtained with our novel method *CandSel* with $\omega \in \{0.20, 0.10, 0.05\}$. ($\circ$/$\bullet$: statistically significantly better/worse than the reference system.)

Dataset		Ref. System	CandSel $\omega = 0.20$	CandSel $\omega = 0.10$	CandSel $\omega = 0.05$
AIDS	Acc [%]	98.53	97.53$^\bullet$	98.80	96.53$^\bullet$
	Time [s]	9.93	7.56	5.62	4.99
ENZYMES	Acc [%]	41.67	36.66$^\bullet$	29.17$^\bullet$	24.17$^\bullet$
	Time [s]	2.18	1.21	0.86	0.66
IMDB-BINARY	Acc [%]	66.00	59.50$^\bullet$	58.50$^\bullet$	55.50$^\bullet$
	Time [s]	4.32	3.67	2.03	2.26
MUTAGENICITY	Acc [%]	71.33	72.95°	72.02	70.60
	Time [s]	63.27	48.70	41.99	36.90
NCI1	Acc [%]	70.33	69.24$^\bullet$	68.96$^\bullet$	68.15$^\bullet$
	Time [s]	51.41	43.10	34.05	29.38
PROTEINS	Acc [%]	73.82	70.82	70.82	71.24
	Time [s]	7.09	3.88	2.88	1.97

decreases as the parameter ω is reduced (as expected). Simultaneously, we observe substantial reductions in the runtimes with increasing values of ω. The runtime of our novel system with $\omega = 0.10$, for instance, is reduced to about 50% of the runtimes of the reference system on all datasets.

On the AIDS, NCI1, ENZYMES, and IMDB-BINARY, we observe that the classification accuracy of our novel approach is in general worse than the accuracy obtained by the reference system. We also see that most of these deteriorations are statistically significant (with the exception of the result obtained on the AIDS dataset with $\omega = 0.10$). However, we can also report that at least on AIDS and NCI1 the classification accuracies obtained by means of our novel system are in a fairly similar range as those of the reference system. Moreover, on the other two datasets, viz. MUTAGENICITY and PROTEINS, not a single statistically significant deterioration compared to the reference system can be seen – on the contrary, we observe one statistically significant improvement (on the MUTAGENICITY dataset with $\omega = 0.20$).

Overall these classification results of *CandSel* are convincing and encouraging especially when considering the substantial decrease in the runtimes of our framework compared with the reference system.

Table 4.4: Classification accuracy and runtime (in seconds) obtained with a k-NN classifier on the original graphs (Reference System) and the results obtained with our novel method *EarlyClass* with $\delta \in \{5, 4\}$. ($\circ$/$\bullet$: statistically significantly better/worse than the reference system.)

Dataset		Ref. System	EarlyClass	
			$\delta = 5$	$\delta = 4$
AIDS	Acc [%]	98.93	98.60	98.73
	Time [s]	9.93	3.48	3.31
ENZYMES	Acc [%]	41.67	36.67$^\bullet$	37.50
	Time [s]	2.18	3.96	3.76
IMDB-BINARY	Acc [%]	66.00	64.00	62.00$^\bullet$
	Time [s]	4.32	8.14	5.63
MUTAGENICITY	Acc [%]	71.63	71.07	68.12$^\bullet$
	Time [s]	63.27	60.07	39.00
NCI1	Acc [%]	70.52	70.56	68.15$^\bullet$
	Time [s]	51.41	54.22	33.78
PROTEINS	Acc [%]	73.82	75.10	75.54°
	Time [s]	7.09	7.32	4.40

4.4.3.2 *Early Classification*

Table 4.4 shows the classification accuracy and the runtime achieved with the reference system and our novel method *EarlyClass* that uses a 5-NN classifier in conjunction with two thresholds $\delta \in \{4, 5\}$[4].

Likewise to the method *CandSel*, the accuracies achieved with *EarlyClass* are – more or less – comparable to the results of the reference system. For instance, with $\delta = 5$, the classification accuracy remains statistically equivalent to the results obtained with the reference system on all datasets. When the threshold is reduced to $\delta = 4$, the classification accuracy achieved on MUTAGENICITY, NCI1, and IMDB-BINARY is statistically worse than the accuracy of the reference system. Note, however, that our novel system performs in a fairly similar range as the reference system in all cases. Moreover, with $\delta = 4$ we even observe a statistically significant improvement in the classification accuracy on the PROTEINS dataset.

Regarding the runtimes we also observe substantial speed-ups of our

[4]During the validation of the meta-parameters on the AIDS dataset, we observe that in step 2 the graphs stem from one class only. Hence, rather than performing a second matching on these graphs, we decide to directly classify the few graphs for which the second step is actually necessary. Note that this applies on the AIDS dataset only.

Table 4.5: Statistics drawn from the method *EarlyClass* with $\delta = 4$. $|G_{te}|$ is the number of test graphs per dataset. Num class. and Acc refer to the number of graphs classified and the classification accuracy obtained, respectively (during that step). Final Acc is the global classification accuracy obtained at the end of the classification process.

Dataset		Step 1		Step 2		Final Acc [%]
	$\lvert\mathcal{G}_{te}\rvert$	Num class.	Acc [%]	Num class.	Acc [%]	
AIDS	1,500	1,479	98.78	21	95.24	98.73
ENZYMES	120	14	14.29	106	40.57	37.50
IMDB-BINARY	200	56	55.36	144	64.58	62.00
MUTAGENICITY	2,337	1,182	70.73	1,155	65.45	68.12
NCI1	2,100	1,004	67.43	1,096	68.81	68.15
PROTEINS	233	133	78.95	100	71.00	75.54

method compared to the reference system. Using threshold $\delta = 4$, for instance, we observe a substantial decrease of the runtime of about 40% on the datasets MUTAGENICITY, NCI1, and PROTEINS. On the AIDS dataset, the runtime of our method is even about five times faster than the runtimes of the reference system (mainly due to the omitted second step on this dataset). Interestingly, we observe an increase in the runtime of our method on both the ENZYMES and IMDB-BINARY datasets. This result has encouraged us to do some further research and investigations on the behavior of the early classification strategy.

In Table 4.5, we show the number of actually classified graphs with *EarlyClass* (we focus on $\delta = 4$) and the corresponding classification accuracy in both steps (step 1 and step 2).

We observe that, in general, a large amount of the graphs are classified during the first step. For instance, on the AIDS dataset about 98% of the test graphs are classified during the first step. For MUTAGENICITY, NCI1, and PROTEINS, about 50% of the graphs are instantly classified without any further computation. For ENZYMES and IMDB-BINARY, however, we observe that the majority of the graphs are classified during the second step of the algorithm, which might explain why the runtime for this dataset increases. That is, on these particular datasets, the computational overhead of our novel two-step classification method cannot be compensated by many early classifications.

When comparing the classification accuracies achieved in step 1 and step 2 separately, we observe that the classification accuracy decreases in general in the second step (see, for instance, on AIDS, MUTAGENICITY and PROTEINS). A possible explanation might be that only the "difficult" graphs remain to be classified during the second step.

4.5 Multiple Classifier System Based On Reduced Graphs

In the present section, we propose a novel framework that makes use of the reduction process presented in Section 4.2. Roughly speaking, the novel method is based on three basic steps (as illustrated in Fig. 4.7).

- First, we create various reduced graph subspaces, that contain graphs that are in turn reduced to the nodes that contribute the most to the original graph structure. That is, we map the graphs into various reduced graph spaces.
- The second step consists of computing a graph dissimilarity between the graphs in each of the reduced graph subspaces.
- The third and last step of our procedure consists in linearly combining either the distances or the predictions obtained in the different graph subspaces. The meta-parameters for combination, viz. the weight coefficients, are either optimized via grid-search or by means of a genetic algorithm.
  The combined distances or predictions are used as a basis for the final classification. Any classification method that makes use of GED in some way can be used for this purpose (e.g., distance-based graph kernels or distance-based classifiers such as the k-Nearest Neighbor classifier).

From a broad perspective, the proposed framework, as shown in Fig. 4.7, is somehow related to a recently introduced hierarchical graph matching framework [121][5]. In this framework, the nodes are aggregated into supernodes during a graph compression process. A hierarchy of compressed graphs is constructed by means of a community detection algorithm. Then, the matching is performed, starting at the most compressed graphs and potentially going up level by level if a certain similarity threshold is exceeded. In our method, however, we make use of reduced graphs, i.e., nodes are

[5] Other hierarchical graph representations are presented in [40; 41], for instance.

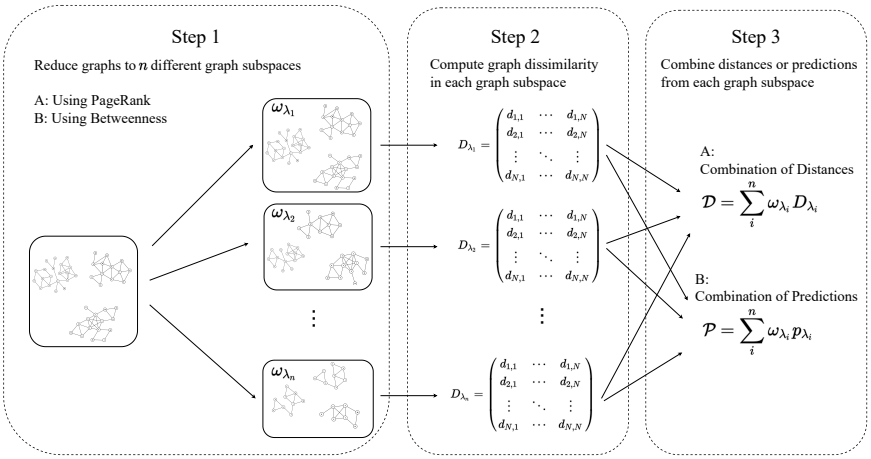


Fig. 4.7: The three basic steps of our novel framework: (1) graph reduction to obtain the reduced graph subspaces, (2) graph matching in reduced graph subspaces and (3) building a multiple classifier system

omitted/deleted during the graph reduction procedure (rather than combined via compression). To this end, we quantify the structural information of each node via centrality measures adapted from social network analysis (formally introduced in Subsection 4.2.1). Next, we remove the nodes that contribute the least to the structure of the graph (according to the centrality measure actually applied). Moreover, we use the extra information gained from the reduced graph subspaces in a multiple classifier scenario and do not perform a coarse-to-fine classification.

4.5.1 *Building a Multiple Classifier System*

In this subsection, we build a multiple classifier system for graphs. To this end, we apply the graph reduction method presented in Section 4.2 on all graphs in $\mathcal{G}$ to obtain graph subspace $\mathcal{G}_\lambda$, where λ corresponds to a given reduction level. We define $N := |\mathcal{G}| = |\mathcal{G}_\lambda|$. Given r different graph subspaces $\mathcal{G}_{\lambda_1}, \ldots, \mathcal{G}_{\lambda_r}$, we can now compute pairwise graph dissimilarities in each graph subspace $\mathcal{G}_{\lambda_i}$. In the present chapter, we employ the GED approximation BP as outlined in Section 4.1.

In detail, for each graph $G_{\lambda_i}^{(j)} \in \mathcal{G}_{\lambda_i}$ we create a distance vector

$$d_j = [d_{j,1}, d_{j,2}, ..., d_{j,N}]$$

representing the distances between itself and the N other graphs in $\mathcal{G}_{\lambda_i}$ and merge the obtained vectors to produce a distance matrix $\boldsymbol{D}_{\lambda_i}$.

$$\boldsymbol{D}_{\lambda_i} = \begin{bmatrix} d_{1,1} & \cdots & d_{1,N} \\ d_{2,1} & \cdots & d_{2,N} \\ \vdots & \ddots & \vdots \\ d_{N,1} & \cdots & d_{N,N} \end{bmatrix} \tag{4.4}$$

Once the distance matrices $\boldsymbol{D}_{\lambda_i}$ for each reduction level λ_i are obtained, we employ two different procedures to combining them and getting a final classification result. In both of these combination scenarios, we employ a distance-based classifier, viz. a k-Nearest Neighbor classifier (k-NN). The k-NN is clearly advantageous in the proposed framework because it directly operates on the resulting distances and can also be used as an indicator of the underlying quality of the distances. Both combination procedures are described in detail in the following paragraphs.

The first combination procedure consists of linearly combining the multiple distance matrices $\boldsymbol{D}_{\lambda_1}, \ldots, \boldsymbol{D}_{\lambda_r}$ at different levels $\lambda_1, \ldots, \lambda_r$ into one *meta-distance matrix* defined by

$$\mathcal{D} = \sum_{i}^{r} \omega_{\lambda_i} \mathbf{D}_{\lambda_i}, \tag{4.5}$$

where parameter $\omega_{\lambda_i} \in [0, 1]$ weights the influence of each reduced graph subspace $\mathcal{G}_{\lambda_i}$. Matrix $\mathcal{D}$ is eventually fed into a k-NN to perform the final classification.

The second idea for condensing the r different graph subspaces $\mathcal{G}_{\lambda_1}, \ldots, \mathcal{G}_{\lambda_r}$ consists of combining the predictions obtained from the k-NN at each reduced graph subspace. Formally, we obtain a prediction vector

$$\boldsymbol{p}_{\lambda_i} = [p_1, p_2, ..., p_N]^T \tag{4.6}$$

for each graph subspace $\mathcal{G}_{\lambda_i}$ where p_j with $j = 1, \ldots, N$ corresponds to the prediction of the j-th graph in the graph subspace $\mathcal{G}_{\lambda_i}$. The prediction vectors are finally linearly combined by

$$\mathcal{P} = \sum_{i}^{r} \omega_{\lambda_i} \mathbf{p}_{\lambda_i} \tag{4.7}$$

in order to obtain the final classification result. That is, we conduct a weighted majority voting [163].

To weight the influence of each reduced graph subspace $\mathcal{G}_{\lambda_i}$ both combination methods introduced above make use of a vector $\boldsymbol{\omega} = [\omega_{\lambda_1}, \ldots, \omega_{\lambda_n}]$, that incorporates the n weighting factors ω_{λ_i} for all graph subspaces.

Our goal is to linearly combine the n reduced graph subspaces, and thus we apply further constraints on $\boldsymbol{\omega}$ such that each entry $\omega_{\lambda_i} \in \boldsymbol{\omega}$ is comprised in a range between 0 and 1 and the sum of all weights equals 1. Formally,

$$\begin{gathered} \sum_{i=1}^{r} \omega_{\lambda_i} = 1 \\ \text{and} \\ \omega_{\lambda_i} \in]0,1[\quad \forall i = 1, \ldots, n \end{gathered} \tag{4.8}$$

We aim to find the linear coefficient vector $\boldsymbol{\omega}^*$ such that the combined distance matrix $\mathcal{D}$ or the combined predictions $\mathcal{P}$ lead to the best possible classification accuracy. We use two different strategies to find the optimal coefficient vector $\boldsymbol{\omega}^*$.

The first optimization method consists of a search over the parameter space in a grid-search fashion. Unfortunately, grid-search is not an efficient technique and scales poorly when the search space is large. In our specific case the search space has a size of $\mathcal{O}(\mathrm{D}^r)$, where D is the total number of values that a weight $\omega_{\lambda_i} \in \boldsymbol{\omega}$ can take and r is the number of graph subspaces that are potentially combined.

As a second optimization technique we use a *Genetic Algorithm* (GA) [164]. GAs are more efficient and scalable search procedures over large search spaces than grid search approaches. Therefore, by means of GAs, we are able to explore more subtle combinations of the weights and thus potentially leading to improved classification accuracies. However, it is important to note that the optimality of the solution found is not guaranteed, and GAs may suffer from overfitting without any well-defined regularization procedures to prevent it.

The GA procedure begins by randomly initiating a set of chromosomes. These chromosomes are then evaluated using the fitness function, and the best-performing ones are selected for the crossover operation, generating the next generation of chromosomes. The crossover step is uniformly performed among the genes of the parents. We introduce a mutation chance with a probability of p_m for each newly created chromosome. This procedure

is repeated until a specified criterion is met, with the criterion set to be reached after a fixed number of optimization iterations in practice.

In our approach, we define $\boldsymbol{\omega}$ as the so-called *chromosome* where each entry $\omega_{\lambda_i} \in \boldsymbol{\omega}$ represents a *gene*. We set the *fitness function* of a chromosome to be the classification accuracy of the k-NN and allow both operations *mutations* and *cross-overs*.

4.5.2 *Experimental Setup and Validation Process*

The main purpose of the following experiments is to empirically verify whether or not the information extracted out of the reduced graphs can help to improve the overall classification performance. In order to test this hypothesis, we first build a baseline for our evaluation by running a k-NN classifier on the original graphs.

The individual hyperparameters of the k-NN are optimized with the graphs contained in the validation set. To alleviate overfitting during the optimization process, we apply a 5-fold cross-validation. The parameters to optimize consist of $\alpha \in]0, 1[$ that weights the relative influence of node and edge edit operation costs and $k \in \{1, 3, 5\}$ that corresponds to the number of neighbors used by the k-NN. We show the optimal parameters α and k found for each dataset in Table 4.6.

For our novel framework, we use the optimized hyperparameters α and k computed during the optimization phase on all graph subspaces $\mathcal{G}_{\lambda_1}, \ldots, \mathcal{G}_{\lambda_r}$. We set the reduction factors to $\lambda \in \{1.0, 0.8, 0.6, 0.4, 0.2\}$ to obtain the original graph space and four reduced graph subspaces with both PageRank and Betweenness. In our evaluation the five graph (sub)spaces, reduced with PageRank and Betweenness, are either used individually or combined with each other. In the combined case, we obtain a total of nine graph (sub)spaces (the original graph space and four subspaces per centrality measure).

When optimizing the weighting parameters $\boldsymbol{\omega}$ with grid search we use the five reduction levels presented above in conjunction with 11 possible weighting factors , i.e. $\omega_{\lambda_i} \in \{1.0, 0.9, \ldots, 0.1, 0.0\}$. Only with those reduction factors the search space is already quite large (having $11^5 = 161,051$ different possibilities). Because of the exponential growth of the search space we cannot apply the grid-search procedure in the scenario where we combine PageRank and Betweenness graph subspaces (in this case the search space would have a size of $11^9 \approx 2.9$ billion possibilities which is no longer feasible).

Table 4.6: Optimal values for α and k obtained during the hyperparameter optimization on the validation sets.

Dataset	Hyperparameter	
	$\boldsymbol{\alpha}$	$\boldsymbol{k}$
AIDS	0.7	1
ENZYMES	0.9	1
IMDB-BINARY	0.9	5
MUTAGENICITY	0.6	5
NCI1	0.7	5
PROTEINS	0.9	3

Table 4.7: The different reduction, combination, and optimization methods to create ten experimental setups.

Reduction Methods	**Combination Methods**	**Optimization Methods**
PageRank (PR)	Combination of Distances (CoD)	Grid Search (GS)
Betweenness (BW)	Combination of Predictions (CoP)	Genetic Algorithm (GA)
PageRank + Betweenness (PR + BW)[6]		

For the GA optimization, we use a random initial population of 30 individual chromosomes, where the random weights of each chromosome (i.e., the genes) are defined such that they sum up to one to match the weighting constraints. Furthermore, the crossover sites in each iteration of the GA are randomly chosen, the mutation probability p_m is set to 0.1, and we run the GA for 100 iterations.

In Table 4.7 we summarize all the reduction, combination, and optimization methods discussed above. We are now able to combine all the presented methods with each other. For instance, we can create a system termed *PR-CoD-GS* that associates PageRank with the combination of distances and a grid search optimization. As stated above, the grid search optimization is not applicable to the combined reduction, leading to a total of ten different experimental setups (see Fig. 4.8).

[6]Due to computational reasons this combined reduction is only optimized via GA.

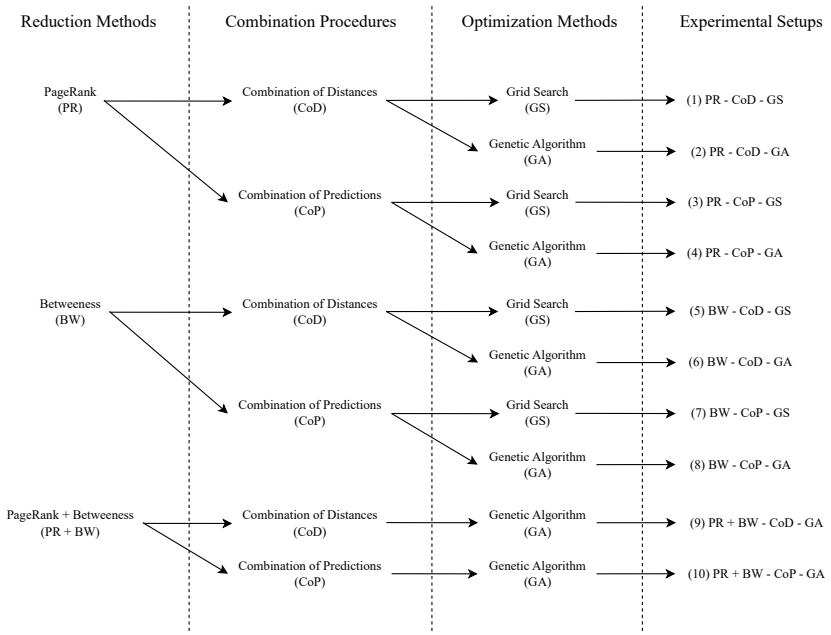


Fig. 4.8: Combinations of reduction, combination, and optimization methods to create ten experimental setups

In Fig. 4.9, we show a bar plot that displays the individual weights $\boldsymbol{\omega}^*$ obtained after the optimization procedure. The figure exhibits the influence of the individually reduced subspaces for all conducted experiments.

In some cases we observe that the original graph subspace $\mathcal{G}_{\lambda_{1.0}}$ dominates the other subspaces. This trend is particularly apparent, for instance, on the ENZYMES and NCI1 datasets with *BW-CoD-GS* and *BW-CoP-GS*, respectively. On the other hand, we can observe in some cases that the reduced graph subspaces substantially contribute to the combined distances and/or predictions. For instance, on the MUTAGENICITY or PROTEINS datasets with *BW-CoP-GS*, the original graph space is completely omitted or does not substantially influence the final classification.

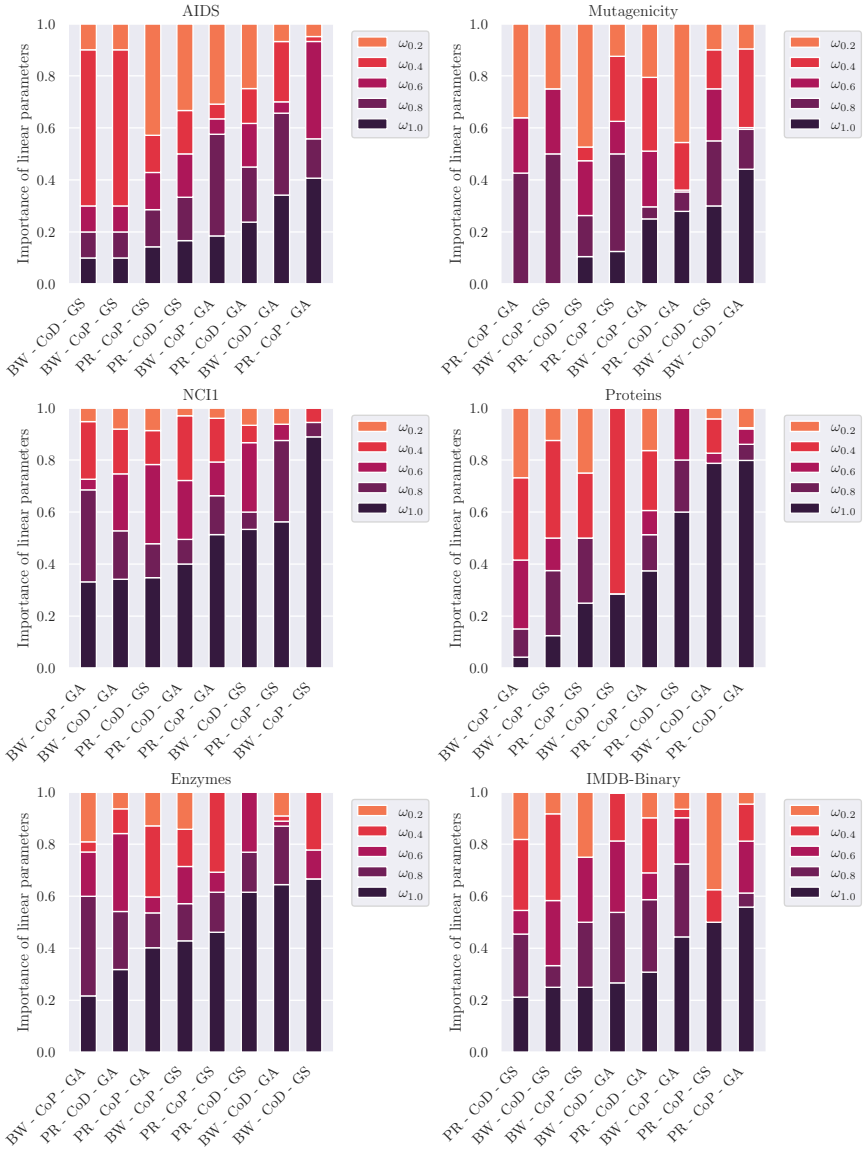


Fig. 4.9: Importance of the individual graph subspaces in the linear combination for all datasets as well as all reduction, combination and optimization methods

Yet, in tendency, no clear pattern in the weighting factors is visible that could favor any of the graph subspaces. Thus the optimal weighting parameters have to be found in an empirical fashion. This observation may indicate that all the reduced subspaces are somehow important and that the optimal weighting depends on the actual application[7].

4.5.3 *Accuracy of the Multiple Classifier System*

In Table 4.8, we present the classification accuracies obtained on all test sets by our method that combines either the distances, termed *Combination of Distance Matrices (CoD)*, or the predictions, termed *Combination of Predictions (CoP)*. Both combinations are either applied on PageRank (PR), Betweenness (BW), or PageRank and Betweenness (PR + BW) reduced graphs. Additionally, we present individual results for both optimization strategies, viz. grid search (GS) and genetic algorithm (GA) (note that for PR + BW only the GA optimization is applied due to computational reasons).

We start our discussion with a focus on the PageRank reduced graphs. We observe that at least one of the proposed combinations of distances or predictions of the reduced subspaces improves the classification accuracy compared to the baseline on all datasets. In 21 out of 24 comparisons our novel approach achieves better results than the reference system (6 of these improvements are statistically significant). These significant improvements are observed on four different data sets. On the other hand we observe only three deteriorations of which only one is significant. Last but not least, we observe that on four datasets the combination of PageRank reduced graphs achieves the overall best results (shown in bold face). Two of these overall best results are achieved with distance based and two with prediction based combinations.

In the case of combinations of Betweenness reduced graphs, comparable, yet slightly worse, results as with the PageRank reduction are obtained. That is, with Betweenness we observe only in 13 out ouf 24 comparisons an improvement over the reference system. On the ENZYMES dataset, however, the classification accuracy is substantially improved by about 8 percentage points when compared to the baseline (from 41.67% to 49.18%).

When combining both PageRank and Betweenness graph subspaces, we

[7] We also study the weighting factors of each graph subspace when combining PageRank and Betweenness graph subspaces. Yet, no clear trend appears in the visualization and thus we do not display those results here.

Table 4.8: Classification accuracy [%] obtained on the test set with linear combinations of reduced graphs. We present the results obtained by a k-NN for the baseline and our two combination methods that are Combination of Distances (CoD) and Combination of Predictions (CoP). The best result per dataset is shown in boldface. (○/●: statistically significantly better/worse than the baseline on a 5% level using a Z-test.)

Method Combination	Dataset					
	AIDS	**ENZYMES**	**IMDB-BINARY**	**MUTAGENICITY**	**NCI1**	**PROTEINS**
Baseline k-NN	98.53	41.67	66.00	71.33	70.33	73.82
PR - CoD - GS	99.13○	45.83	64.50	71.84	72.09	73.39
PR - CoD - GA	99.13○	48.33○	66.00	**72.66**	**73.22**○	75.54
PR - CoP - GS	**99.33**○	41.67	**70.00**	72.32	70.52	73.82
PR - CoP - GA	99.13○	37.50●	**70.00**	71.84	70.52	76.39○
BW - CoD - GS	98.07	46.67	64.00	71.25	71.28	69.52
BW - CoD - GA	99.20○	**49.18**○	64.00	72.53	71.56	75.53
BW - CoP - GS	98.07	43.33	65.50	71.29	70.52	76.82○
BW - CoP - GA	99.20○	41.67	65.50	71.59	70.24	73.82
PR + BW - CoD - GA	99.13○	48.33○	65.50	**72.66**	71.89	75.54
PR + BW - CoP - GA	99.27○	40.83	65.00	71.72	69.38	**77.25**○

Table 4.9: The average rank for each classification method, with the top three highlighted.

Method Combination	Average Rank
PR - CoD - GS	4.1
PR - CoD - GA	**2.0**
PR - CoP - GS	3.5
PR - CoP - GA	3.9
BW - CoD - GS	5.8
BW - CoD - GA	**3.3**
BW - CoP - GS	4.4
BW - CoP - GA	4.8
PR + BW - CoD - GA	**2.3**
PR + BW - CoP - GA	4.5

observe eight improvements in total when compared to the baseline. Three of these improvements are statistically significant. Moreover, with this particular combination we obtain overall best results on the MUTAGENICITY and PROTEINS datasets.

In order to assess which reduction method (PR or BW) together with which combination method (CoD or CoP), coupled with which optimization procedure (GS or GA) performs the best, we rank all methods per dataset and average up the ranks per method (see Table 4.9). We report the top three that achieve the smallest average of rank points, viz. *PR-CoD-GA*, *PR+BW-CoD-GA*, and *BW - CoD - GA*. It is noteworthy that PageRank plays at least a role in the top two and that the top three methods are based on the distance combination that is optimized via genetic algorithm. At the opposite end of the ranking, we have *BW-CoD-GS* and *BW-CoP-GA*.

In summary, we can report that the GA optimization method achieves better results than the grid search, the combination of distances performs better than the combination of predictions, and PageRank works better than Betweenness for building the reduced graph subspaces.

4.5.4 *Time Analysis*

The main drawback of our novel three-step method is the extra computation time used to compute GED in the different graph subspaces. The computation of GED is actually the bottleneck of our framework in terms of

time complexity (although using an $\mathcal{O}(n^3)$ approximation algorithm where $n = |V|$). Hence, we focus our runtime analysis on the second step of our framework.

Based on the fact that the reduced graphs have by definition fewer nodes, the run time of GED is supposed to decrease the smaller the graph subspaces are. In Fig. 4.10, we show the runtime of GED for each graph subspace per dataset. We observe that in 4 out of 6 datasets (i.e., AIDS, PROTEINS, ENZYMES, and IMDB-BINARY) the execution time is only about twice slower compared to the runtime of GED computation on the original graphs. On the other two datasets (MUTAGENICITY and NCI1) the run time is about three times slower than the original system. Considering that our combined systems are superior to the reference system, one can certainly argue that the higher runtime is worth it in any case.

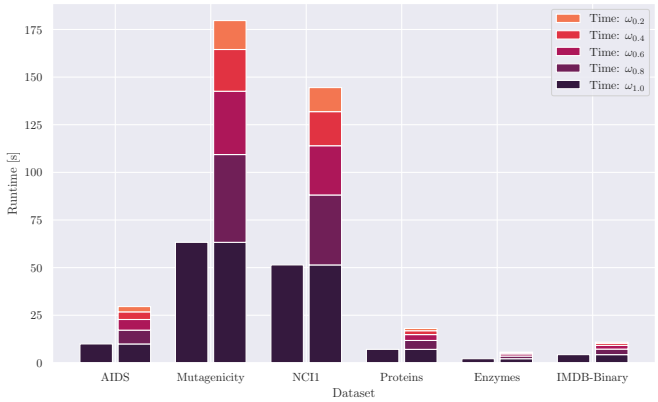


Fig. 4.10: Runtime of GED computation for each graph subspace per dataset.

4.6 Conclusion

In the present chapter, we propose to use centrality measures for nodes in a graph in order to iteratively discard nodes with low centrality scores. We apply two different centrality measures, viz. PageRank and Betweenness. The main motivation for this reduction is a potential gain in the runtime

when the reduced rather than the original graphs are matched with each other.

We demonstrate the benefits and limitations of our reduction technique in a comprehensive experimental evaluation on six real-world datasets. We observe substantial reductions of the graph matching time on all datasets. That is, the runtime of GED computation can be more than halved in general. Comparing the classification accuracies achieved on reduced graphs with the classification accuracy achieved on the original graphs, we can report that in general the accuracies drop with increasing reduction levels. However, at least up to Level $\lambda = 0.6$ the accuracies remain comparable with the original classification accuracies. Moreover, our evaluation clearly shows that using elaborated methods for graph reduction rather than random subsampling is clearly beneficial for most reduction levels and data sets.

Following this, we investigate the use of a graph reduction method in a two-step classification scheme. In particular, we propose two strategies (candidate selection and early classification) that are applied on reduced graphs in order to speed up the complete classification procedure. Both modifications allow us to control the trade-off between classification accuracy and computation time.

With an empirical evaluation on the same six graph datasets, we verify the computational advantages of our novel two-step classification technique. That is our pruning strategies substantially reduce the runtime on all datasets. Moreover, we demonstrate that by using strongly reduced graphs in a two-step procedure, it is possible to maintain reasonable classification accuracy in general. Note that our approach is in some cases and on some datasets even capable to improve the classification accuracy of the reference system.

Finally, we propose a novel framework for graph-based pattern recognition that combines extra information gained from reduced graph subspaces. Roughly speaking, the proposed method works in three subsequent steps. In the first step, multiple reduced graph subspaces using graph reduction methods are produced. During the second step, we use GED to compute the distances between the graphs in their corresponding reduced graph subspaces. In the last step, we linearly combine either the distances or the predictions obtained in the differently reduced graph subspaces. The linear coefficients for the combination are either optimized by means of a grid search or a genetic algorithm.

We empirically validate the advantage of this novel ensemble method.

In particular, we show that a k-NN classifier clearly benefits from the combination of the distances or predictions of reduced graphs. That is, on all datasets the proposed algorithmic framework outperforms the reference system by several percentage points. Comparing the different subsystems with each other, we conclude that the PageRank reduction in conjunction with the combination of distances optimized via genetic algorithm is a good choice in general.

Regarding the importance of each graph subspace we can conclude that all of them are somehow important. That is, the actual importance seems to depend on both the dataset and optimization process.

Clearly, the increase in computation time is the major drawback of the proposed ensemble system. The runtime of our novel framework is actually higher than that of the reference system, but not five times higher, as one might have expected at first glance. The reason for this is, of course, the dramatic decrease of the runtime in strongly reduced graph subspaces. Overall, we observe runtimes that are twice or at most three times as high as those of the reference system. Considering the significantly improved classification accuracy, this slowdown seems acceptable.

Graph Reduction by means of Spectral Clustering

5

> Nous sentons que même si toutes les possibles questions scientifiques ont trouvé leur réponse, nos problèmes de vie n'ont pas même été effleurés. Assurément il ne subsiste plus alors de question ; et cela même constitue la réponse.
>
> *Tractatus logico-philosophicus* (1922), Ludwig Wittgenstein

5.1 Introduction

As previously explained in Chapter 2, several methods have been proposed in the literature to perform graph matching [4; 5]. In this chapter, we focus on four popular graph matching paradigms (formally defined in Section 2.4), namely *Graph Edit Distance* (GED) [86], *Graph Kernels* [6] with the *Shortest-Path kernel* (SP) [19] and the *Weisfeiler-Lehman kernel* (WL) [20], as well as a *Graph Neural Network* (GNN) [9].

As discussed before, a major limitation of graph-based pattern recognition is its high computational cost. To address the computational problems of graph matching, diverse approximation techniques have been proposed. Another approach for improving the efficiency of graph-based pattern recognition is to work with reduced versions of the original graphs [32] (as discussed in Chapter 4).

In the present chapter, we propose a novel two-step approach (originally published in [46]) with the common goal of the previous chapter, viz. to substantially reduce the time required for graph classification.

- In the first step, the original graphs are reduced to a given level

using spectral clustering.
- In the second step, graph matching and classification are performed on the reduced graphs (using both GED in conjunction with a k-NN, two Graph Kernels (i.e., SP and WL) with an SVM, and a GNN).

The main contribution of this chapter is to investigate the effects of this graph reduction process in a typical graph-based pattern recognition scenario. In particular, we evaluate the effectiveness of the proposed method by comparing the matching time and classification accuracy achieved on the reduced graphs with the corresponding metrics observed in the original graph domain. This experimental setup allows us to study the impact of the proposed spectral graph reduction on graph-based pattern recognition computation.

The remainder of the present chapter is structured as follows. In Section 5.2, we describe the spectral graph clustering used to determine graph partitioning, and then present our novel graph reduction method. In Section 5.3, we describe the experimental setup and present the main results of the conducted empirical study. In the last section, Section 5.4, we summarize our findings and suggest directions for future work.

5.2 Graph Reduction Method

The major objective of the present section is to introduce and research a novel method for substantially reducing the size of graphs while preserving their essential properties. These properties can vary depending on the specific problem at hand. In this work, we aim to maintain the classification accuracy achieved on the reduced graphs as close as possible to the one obtained on the original graphs.

Specifically, given a graph $G = (V, E)$ with n nodes and m edges, we create a reduced graph $G_\rho = (V_\rho, E_\rho)$ with $n_\rho < n$ nodes and $m_\rho < m$ edges such that G_ρ is a good approximation of G in some sense [33]. Parameter $\rho \in \mathbb{N}$ is a user-defined reduction factor[1] that controls the extent to which

[1] In the present chapter, the reduction factor ρ is defined slightly differently to the reduction factor λ introduced in Chapter 4. Here, the parameter ρ quantifies by how much the size of a graph is reduced (see further explanation in Subsection 5.2.2), while the parameter λ specifies the percentage of nodes remaining in the reduced graphs. This distinction in ρ is made for ease of notation when using node clustering in graph reduction.

the original graph is reduced. For example, a value of $\rho = 2$ results in a reduction of the graph size (i.e., the number of nodes) by approximately 50%. The reduced graph domain $\mathcal{G}_\rho = \{G_\rho^{(1)}, \ldots, G_\rho^{(N)}\}$ is obtained from the original graph domain $\mathcal{G}$ by reducing all graphs $G \in \mathcal{G}$ according to the defined reduction procedure.

The aim of the following two subsections is twofold. First, in Subsection 5.2.1, we elaborate on the principles of spectral graph clustering, which builds the basis of our reduction method. Second, in Subsection 5.2.2, we demonstrate how one can employ the node partitioning resulting from spectral clustering to achieve a substantial and meaningful reduction of the underlying graphs.

5.2.1 *Graph Clustering*

Graph Clustering [165] is a technique for dividing the nodes of a graph into groups, called *clusters*, such that the nodes within each cluster are closely related in some pre-defined sense.

Formally, given a graph $G = (V, E)$, the *c-way clustering* of the node set V is defined as a set of non-empty clusters $P^c = \{C_1, \ldots, C_c\}$ such that each $C_i \cap C_j = \emptyset$ for $i \neq j$, and $\cup_{i=1}^{c} C_i = V$. The clustering of the nodes is typically based on the underlying structure of the graph so that nodes belonging to the same cluster must exhibit "similar behavior". One possible way to formally define this behavior is to quantify the pairwise node similarity by their connectivity in terms of the underlying edge structure.

In the present chapter, we use spectral graph clustering [166; 167], which relies on a few important properties of the graph Laplacian $\mathbf{L}$ (formally defined in Subsection 2.2.3). The graph Laplacian matrix $\mathbf{L}$ is symmetric and positive semi-definite with $\mathbf{L1} = 0$, where $\mathbf{1}$ refers to the n-dimensional all-ones vector $\mathbf{1} = (1, \ldots, 1)$. This implies that the smallest eigenvalue λ_1 of $\mathbf{L}$ is 0, and moreover, the corresponding eigenvector $\mathbf{u}_1$ is equal to $\mathbf{1}$. The n non-negative, real-valued and ascendingly sorted eigenvalues of $\mathbf{L}$ (i.e., $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$) associated with their corresponding eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_n$ are known as the *graph spectrum* of $\mathbf{L}$ (thus the name *spectral clustering*).

The graph Laplacian and its spectrum have useful properties for analyzing the topology of the corresponding graph. In particular, many theorems from spectral analysis of the graph Laplacian $\mathbf{L}$ show that the combinatorial properties of a graph can be captured by its spectrum [168]. For instance, one can show that the m multiplicity of the eigenvalue 0 of $\mathbf{L}$ equals the

number of connected components in G. For a thorough review of spectral properties of the graph Laplacian, we refer to [166; 168].

The basic idea behind spectral graph clustering is to compute a node embedding based on the c smallest eigenvectors of the graph Laplacian matrix (where c is the number of clusters). This embedding encodes information about the connections between the nodes and can thus be used to identify clusters of densely connected nodes within the graph. Once the embedding has been computed, standard clustering methods (such as k-means [169] or others) can be applied to find the node partitioning of the graph. The spectral graph clustering algorithm is formally described in Alg. 4.

For our specific purpose of fast graph reduction, we apply a slight modification to the standard spectral clustering algorithm to improve the computational efficiency of the eigendecomposition (line 2 of Alg. 4). Instead of embedding the nodes in a c-dimensional space, where c is the number of clusters to be identified in the graph, we embed the nodes in a d-dimensional space with $d < c$. This reduction in the number of dimensions speeds up the computation of the eigenvalues and eigenvectors, and therefore the overall clustering method. The value of d is treated as a free parameter and is optimized for each dataset individually.

Algorithm 4: (Unnormalized) Spectral Clustering

Input: Adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ of graph $G = (V, E)$, number c of clusters.

1 Compute the graph laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$

2 Compute the first c eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_c$ of $\mathbf{L}$

3 Let $\mathbf{U} \in \mathbb{R}^{n \times c}$ be the matrix containing the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_c$ as columns

4 For $i = 1, \ldots n$, let $\mathbf{y}_i \in \mathbb{R}^c$ be the i-th row of $\mathbf{U}$ which corresponds to the embedding vector of the i-th node of G

5 Cluster the vectors $\{\mathbf{y}_i\}_{i=1,\ldots,n}$ in $\mathbb{R}^k$ with any standard clustering algorithm into clusters $C_1, \ldots, C_c$

Output: Clusters $C_1, \ldots, C_c$

5.2.2 *Graph Reduction*

Based on the spectral graph clustering described above, we propose to coarsen the underlying graphs in a novel graph reduction approach, which

is potentially able to preserve important structural features of the graphs. We are aware that graph reduction has been largely investigated in the literature under different formalisms and different names, such as *graph summarization* and *graph coarsening* [32; 33][2]. Moreover, connections between graph reduction and spectral algorithms were also explored in [170; 171]. The major contribution of the present chapter is that we thoroughly evaluate the effects of reduced graphs on a wide range of graph classifiers.

Before applying the spectral graph clustering algorithm, we first conduct a pre-processing step to all of the graphs in the underlying dataset. In this pre-processing step, we consider all connected components of the graphs as distinct graphs. In other words, in case a certain graph $G \in \mathcal{G}$ consists of more than one connected component, we individually apply the spectral graph clustering to each connected component. This pre-processing step turns out to be necessary because the spectral clustering algorithm may not produce meaningful results when applied to graphs that consist of more than one connected component (due to the multiplicity of zero eigenvalues, which can result in problematic node embeddings).

The basic idea of the proposed graph reduction approach is to condense the nodes of one cluster into supernodes and simplify the edge structure between these supernodes. To create the supernodes, we first obtain the graph partitioning by means of the spectral clustering algorithm and then condense all the nodes in a given partition into a single entity without considering the intra-cluster edges. The computation of the feature vector attached to the supernode is achieved by summing up the n feature vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of the n nodes that belong to the same cluster[3].

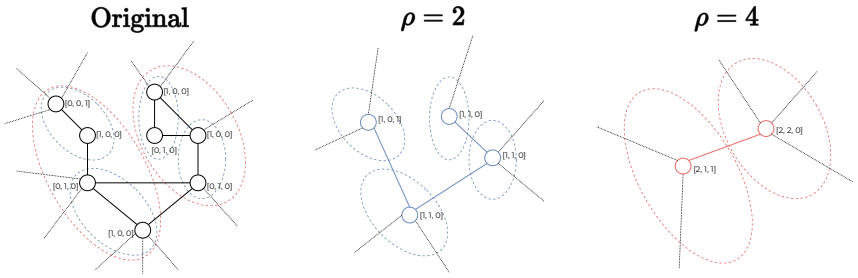
This process effectively reduces the size of the graph by replacing multiple nodes with a single supernode. Additionally, we condense all inter-cluster edges into a single edge between two corresponding supernode in order to simplify the edge structure while preserving the connections between the clusters.

In Fig. 5.1, we illustrate how our supernode creation process operates. In this example, we show two levels of reduction, with $\rho = 2$ and $\rho = 4$, and how nodes are merged at each level once the node clustering is determined. The clusterings obtained with $\rho = 2$ and $\rho = 4$ are represented by blue and red circles, respectively. In the reduced graphs $G_{\rho=2}$ and $G_{\rho=4}$, the node feature vectors correspond to the sum of the feature vectors of the

[2] In the present chapter, the graph coarsening strategy is employed.

[3] Note that other, in particular more elaborated, methods for the labeling of supernodes could be defined – see future work in Section 5.4.

nodes belonging to the blue and red clusters, respectively. Note also how the inter-cluster edges are simplified during the reduction process.



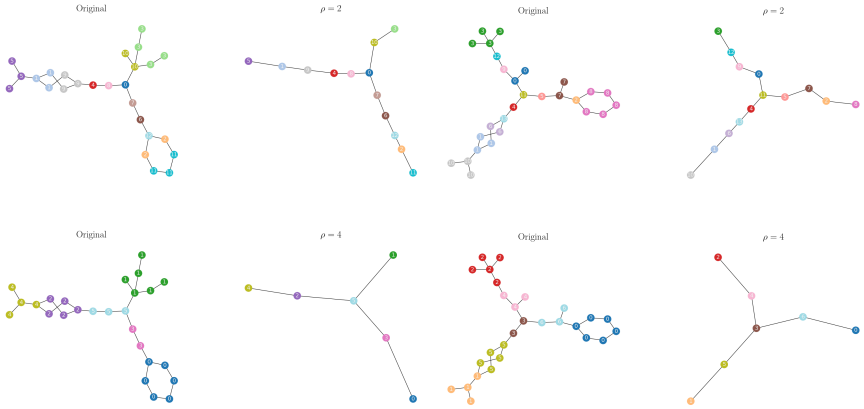
(a) (b) (c)

Fig. 5.1: Example of the supernode creation process. We show parts of the original graph G in (a) and two corresponding graph reductions in (b) and (c) with $\rho = 2$ and $\rho = 4$, respectively.

In our reduction scheme, the number of clusters c corresponds to the number of supernodes obtained upon merging the nodes (and vice versa). Hence, value c determines the size of the reduced graph that is finally created. The number of nodes in each cluster is proportional to the reduction factor ρ, which specifies the amount by which the size of a graph must be divided to obtain the desired reduction. In particular, the number of clusters can be calculated as $c = \frac{|V|}{\rho}$, where $|V|$ is the total number of nodes in the graph. For example, if we have a graph with $|V| = 1{,}000$ nodes and a reduction factor of $\rho = 2$, the size of the graph would be reduced by a factor of two, resulting in $c = \frac{1{,}000}{2} = 500$ clusters.

In Fig. 5.1, we provide two real-world examples that illustrate the graph reduction process. We use graphs from the NCI1 dataset (see Subsection 3.1.8 for details on this particular dataset). We apply a graph reduction with parameters $\rho = 2$ and $\rho = 4$. As can be seen in the figure, both clustering and reduction reflect the communities present in the graphs. In the original graphs, each cluster found is represented by a unique color, and in the reduced graphs, the color corresponds to the reduction of nodes within that cluster into a single supernode. An interesting aspect of spectral partitioning is that it might produce unequally sized clusters, with some clusters containing only a single node while others containing multiple nodes. This behavior is noteworthy because we typically want to keep

the main communities within the same cluster.



(a) (b)

Fig. 5.2: Two examples of the graph reduction algorithm on two different graphs (a) and (b) from the NCI1 dataset using two reduction factors $\rho = 2$ and $\rho = 4$. Each color corresponds to a cluster in the original graph and the corresponding color in the reduced graph represents the condensation of all nodes in the respective cluster into a supernode.

5.3 Experimental Evaluation

This section is organized as follows. First, in Subsection 5.3.1, we briefly outline the datasets used in our evaluation. Next, in Subsection 5.3.2, we summarize the experimental setup. In Subsection 5.3.3, we present the major results obtained on the datasets, and finally, in Subsection 5.3.4, we present a qualitative evaluation of the effects of the proposed reduction.

5.3.1 *Datasets*

We conduct empirical evaluations of the novel reduction approach using nine datasets from the TUDataset graph repository [128]. Table 5.1 provides details on the number of graphs, and classes, as well as the average number of nodes and edges per graph for each dataset.

The first six datasets (BZR, DHFR, ENZYMES, MUTAGENICITY, NCI1, and NCI109) are composed of graphs that represent real-

world molecules and their potential effects or activities. The last three datasets (COLLAB, REDDIT-MULTI-5K, and REDDIT-MULTI-12K) contain graphs that represent different social media networks. For further information about those datasets, please refer to Chapter 3, which provides a detailed explanation for each.

Table 5.1: Statistics of the graph datasets. We show the number of graphs ($|\mathcal{G}|$), the number of classes ($|\Omega|$), and the average number of nodes and edges per dataset ($\emptyset|V|$, $\emptyset|E|$).

	Graph Dataset Property			
Dataset	$\|\mathcal{G}\|$	$\|\Omega\|$	$\emptyset\|V\|$	$\emptyset\|E\|$
BZR	405	2	35.7	38.4
DHFR	467	2	42.4	44.5
ENZYMES	600	6	32.6	62.1
MUTAGENICITY	4,337	2	30.3	30.8
NCI1	4,110	2	29.9	32.3
NCI109	4,127	2	29.7	32.1
COLLAB	5,000	3	74.5	2,457.8
REDDIT-MULTI-5K	4,999	5	508.5	594.9
REDDIT-MULTI-12K	11,929	11	391.4	456.9

Table 5.1 reveals that on most of the graph datasets, the number of nodes are quite similar to the number of edges. This implies that the graphs are sparse, which is positive in our scenario. Spectral graph clustering is particularly efficient on sparse graphs because it involves computing the eigendecomposition of the Laplacian matrix $\mathbf{L}$ (which is computationally efficient on sparse matrices).

We are aware that there are numerous additional datasets available in the TUDataset graph repository that could potentially be used in our evaluation. However, as demonstrated in Section 3.5, rather simple baseline approaches, such as global sum pooling of node features, which reduce graphs to a single feature vector (completely neglecting the edge structure), perform very well and even outperform elaborated graph kernels in some cases. Therefore, we only use datasets in our analysis for which the classification accuracy using this naïve baseline approach is lower than the accuracy achieved with an SVM based on 4-Weisfeiler-Lehman kernel (i.e.,

a Weisfeiler-Lehman kernel with $h = 4$)[4].

An interesting question is whether or not the graph reduction process has a substantial influence on the graph density. To find this out, we show in Table 5.2 the average graph density for both the original graphs and reduced graphs (with reduction factors of $\rho = 4$ and $\rho = 16$). We observe that the original graphs are generally sparse (except the COLLAB dataset where a mean density of 0.51 is observed). On the other datasets, the mean densities range from 0.01 (on REDDIT-MULTI-5K) to 0.16 (on ENZYMES). We observe a trend towards increasing graph densities as the reduction factor is increased. On some data sets the increase in density is substantial. For instance, on BZR the density is increased from 0.06 to 0.27 and 0.70 for $\rho = 4$ and $\rho = 16$, respectively. However, it is also observed that even with the strongest reduction, complete graphs are not obtained (the highest density is obtained on the DFHR data with 0.81).

Table 5.2: Mean of the graph densities of the original graphs and their reduced counterparts (for $\rho = 4$ and $\rho = 16$) for all datasets.

Dataset	Original	Reduction Factor ρ	
		$\rho = 4$	$\rho = 16$
BZR	0.06	0.27	0.70
DHFR	0.05	0.21	0.81
ENZYMES	0.16	0.32	0.57
MUTAGENICITY	0.09	0.37	0.50
NCI1	0.09	0.33	0.50
NCI109	0.09	0.34	0.49
COLLAB	0.51	0.54	0.79
REDDIT-MULTI-5K	0.01	0.04	0.15
REDDIT-MULTI-12K	0.02	0.08	0.21

5.3.2 *Experimental Setup*

For each dataset described in the previous subsection, we create reduced graph domains $\mathcal{G}_\rho$ by reducing the original graphs with reduction factors

[4]We report the results of this naïve baseline approach together with the results of the novel method in Subsection 5.3.3

$\rho \in \{2, 4, 8, 16\}$. These reduction factors lead to slightly reduced graphs (when $\rho = 2$) to quite strongly reduced graphs (when $\rho = 16$). The reduction process is not applied on graphs that have a number of nodes $|V|$ already smaller than, or equal to, ρ. For each reduction factor ρ, we generate reduced graphs using different dimensions for node embedding during spectral clustering. That is, the dimension of the node embedding space is varied in $d \in \{2, 3, 4, 5, 8\}$. Note that d is treated as an additional free hyperparameter and is chosen during the optimization phase.

For each experiment, we produce stratified splits of the datasets into training, validation, and test sets using a 60%, 20%, and 20% split size, respectively. For each dataset, we optimize the classifiers and hyperparameters five times with different data splits and random initialization by means of the validation sets. Finally, the mean and standard deviation of the classification results for the five runs obtained on the test sets are reported.

For the computation of GED, we use unit costs for both node and edge insertions/deletions. To calculate the node substitution cost $c(u_i \rightarrow v_i)$, we utilize the Euclidean distance between the node features $\mathbf{x}_i$ and $\mathbf{x}_j$, respectively, with a cost limit of 2.0. Formally, the substitution cost is $c(u_i \rightarrow v_i) = \min(||\mathbf{x}_i - \mathbf{x}_j||, 2.0)$. This definition ensures that the substitution cost is never greater than the sum of cost of a deletion and a subsequent insertion. Parameter $\alpha \in]0, 1[$ represents the relative importance of node and edge edit operation costs and is varied from 0.1 to 0.9 in increments of 0.1 in our evaluation (with the exception of both REDDIT datasets on the original graphs and REDDIT-MULTI-12K with reduction factor $\rho = 2$. Here, α is fixed to 0.5 due to the high computational cost).

The only parameter that needs to be optimized for the k-NN classifier is the number of neighbors k considered in the classification process. We optimize this parameter in the range $k \in \{3, 5, 7\}$. For the SVM we optimize parameter $C \in 10^{\{-2.0, -1.5, \ldots, 2.0\}}$, which serves as a regularization parameter to control the trade-off between the requirements of large margins and few misclassifications. In other words, C controls what is more important, the minimization of the structural risk or the minimization of the empirical risk. For the training process of the GNN experiments, we use the hyperparameters as proposed in [102].

The experimental evaluation is divided into two parts. In the first part, described in Subsection 5.3.3, we evaluate the classification performance of the four graph classification algorithms that are GED in conjunction with a k-NN (reviewed in Subsection 2.4.1), two Graph Kernels (i.e., SP and

WL) with an SVM (reviewed in Subsection 2.4.2), and a GNN (reviewed in Subsection 2.4.3). By comparing the classification accuracy of these four systems on both the original graphs and the reduced counterparts we can examine the power of the proposed reduction mechanism. We also compare the runtime of the matching and kernel algorithms on both the original and reduced graphs in order to observe the time gain attributable to our novel approach.

In the second part of the evaluation, in Subsection 5.3.4, we present scatter plots that visualize the correlations between the similarity/dissimilarity values obtained on the original graphs and the similarity/dissimilarity values obtained on the reduced graphs.

5.3.3 *Classification Accuracy and Computation Time*

In this subsection, we address the following two research questions:

Q.1 Does the proposed reduction lead to graphs on which a significant decline in classification accuracy is observed (compared to using the original graphs)? Are there substantial differences among the four classifiers employed on the reduced graphs?

Q.2 How large is the runtime improvement that can be achieved by performing graph classification on the reduced rather than on the original graphs?

In order to answer question **Q.1**, we present the classification accuracies for each reduced graph domain across all datasets and all classifiers in Fig. 5.3. Additionally, in each figure, the black horizontal lines represent the results obtained with the baseline where all graphs are reduced to a single feature vector by means of a global sum pooling. The classification of these vectors is then performed with an SVM based on a RBF kernel. Overall, we observe a general, yet relatively slight, decrease in classification accuracy as the size of the graphs is reduced. However, the classification accuracy remains relatively stable even with strongly reduced graphs. This is particularly noteworthy as it indicates that the reduced graphs still retain sufficient information for accurate classification. For example, the classification accuracies of the k-NN using GED obtained on the datasets BZR, DHFR, MUTAGENICITY, NCI1, and NCI109 remain relatively consistent even with strongly reduced graphs. This observation is also valid for the two kernel classifiers. However, on some datasets, we also observe rather

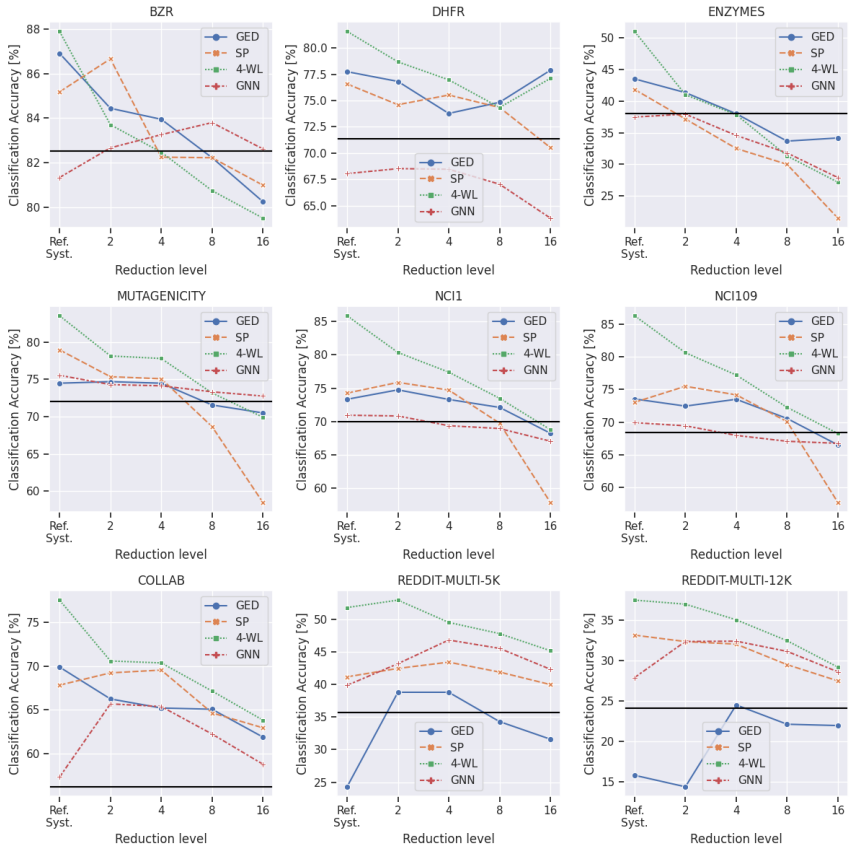


Fig. 5.3: Graph classification accuracies for all datasets and all reduction factors using GED (Graph Edit Distance and k-NN), SP (Shortest Path Kernel and SVM), 4-WL (4-Weisfeiler-Lehman Kernel and SVM), and GNN (DGCNN Graph Neural Network), including the corresponding reference systems that rely on the original graphs ($\rho = 1$). The black horizontal lines represent the results obtained with the baseline, where all the graphs are reduced to a single feature vector.

strong reductions of the classification accuracies – in particular when strong graph reductions are applied (see for instance the ENZYMES dataset where the accuracy drop is clearly visible for all classifiers).

Note that on some datasets – especially on the unlabeled datasets REDDIT-MULTI-5K and REDDIT-MULTI-12K – we can actually improve

the classification accuracy when the classification is performed on the reduced rather than the original graphs. This phenomenon might be attributed to the fact that we fixed parameter α to 0.5 for the reference system (with $\rho = 1$) in order to avoid computational expenses and thus the results shown here are somehow sub-optimal.

In general, we observe that the black horizontal line, representing the results obtained with the baseline where all graphs are reduced to a single feature vector, is only crossed when the graphs are strongly reduced (with reduction factors of $\rho = 8$ or $\rho = 16$). However, on the DHFR dataset and both REDDIT datasets, we observe that even for the original graphs and slightly reduced graphs with $\rho = 2$ and $\rho = 4$, the accuracies obtained with GNN and GED, struggle to surpass the accuracy of the naïve baseline. This indicates that neither GNN nor GED are well-suited methods for solving those tasks.

Overall the results suggest that our approach is effective in improving the accuracy of pattern recognition systems based on graph representations, although, there may be some instances where the baseline outperforms the novel approach (particularly for graphs that have undergone significant reduction).

To provide a more precise analysis of the relative differences among the different systems, we present the classification accuracies for all datasets, classification methods, and reduction factors in Table 5.3. This table allows us to compare the number of instances where the classification accuracy achieved on the reduced graphs is statistically significantly worse than the accuracy achieved on the original graphs. We employ a t-test using the classification accuracy of the five runs to determine if there is a statistically significant difference in accuracy between the reference system and the systems that use the reduced graphs.

We first analyze the effects of the different reduction factors by comparing all datasets and classifiers simultaneously. When the reduction factors are small, we observe no statistically significant difference to the original graphs in 26 out of 36 cases and 19 out of 36 cases for $\rho = 2$ and $\rho = 4$, respectively. However, as the reduction factors increase to $\rho = 8$ and $\rho = 16$, the results degrade and we find that only 12 out of 36 and 9 out of 36 cases are statistically equivalent to the original system, respectively. These observations, together with the insights from Fig. 5.3, suggest that it is possible to obtain reasonable results on the reduced graphs, but that it is considerably more difficult to do so (especially with strongly reduced graphs).

Table 5.3: Classification accuracies obtained by all classifiers, viz. a k-NN using Graph Edit Distance (GED), as well as the ShortestPath graph kernel (SP), the 4-Weisfeiler-Lehman graph kernel (4-WL) in conjunction with an SVM and the DGCNN Graph Neural Network (GNN). We present results on all datasets and reduction factors, i.e., Ref. System ($\rho = 1$) and $\rho \in \{2, 4, 8, 16\}$. Using symbols $\circ$/ $\bullet$, we indicate results that are statistically significantly better or worse than those achieved with the reference system, respectively.

Dataset	Classifier	Ref. System	Reduction Factor ρ			
			$\rho = 2$	$\rho = 4$	$\rho = 8$	$\rho = 16$
BZR	GED	86.9 ± 4.0	84.4 ± 5.4	84.0 ± 6.1	82.2 ± 3.5	80.2 ± 5.5
	SP	85.2 ± 3.7	86.7 ± 3.3	80.2 ± 3.7	82.2 ± 3.5	81.0 ± 3.5
	4-WL	87.9 ± 1.6	83.7 ± 3.0 $\bullet$	82.5 ± 2.7 $\bullet$	80.7 ± 2.8 $\bullet$	79.5 ± 3.2 $\bullet$
	GNN	81.3 ± 0.8	82.7 ± 1.0	83.3 ± 0.5 $\circ$	83.8 ± 0.7 $\circ$	82.6 ± 0.7 $\circ$
DHFR	GED	77.7 ± 1.0	76.8 ± 2.4	73.8 ± 3.1 $\bullet$	74.8 ± 1.1 $\bullet$	77.9 ± 3.4
	SP	76.6 ± 3.1	74.6 ± 1.9	75.5 ± 3.9	74.3 ± 2.9	70.5 ± 3.3 $\bullet$
	4-WL	81.6 ± 2.9	78.7 ± 2.0	77.0 ± 1.8 $\bullet$	74.3 ± 2.8 $\bullet$	77.1 ± 2.8 $\bullet$
	GNN	68.0 ± 1.4	68.5 ± 1.1	68.5 ± 1.1	67.0 ± 0.5	63.8 ± 0.7 $\bullet$
ENZYMES	GED	43.5 ± 5.5	41.3 ± 3.1	38.0 ± 3.9	33.7 ± 3.6	34.2 ± 5.1
	SP	41.8 ± 4.0	37.2 ± 1.9	32.5 ± 2.4 $\bullet$	30.0 ± 2.8 $\bullet$	21.5 ± 2.7 $\bullet$
	4-WL	51.0 ± 3.5	41.0 ± 3.9 $\bullet$	37.8 ± 4.3 $\bullet$	31.3 ± 2.9 $\bullet$	27.2 ± 1.9 $\bullet$
	GNN	37.5 ± 1.6	37.9 ± 1.6	34.6 ± 0.4 $\bullet$	31.8 ± 1.1 $\bullet$	27.9 ± 0.5 $\bullet$
MUTAGENICITY	GED	74.5 ± 1.6	74.7 ± 1.7	74.5 ± 1.2	71.6 ± 1.5 $\bullet$	70.5 ± 1.4 $\bullet$
	SP	79.0 ± 1.2	75.4 ± 1.5 $\bullet$	75.1 ± 1.5 $\bullet$	68.7 ± 1.3 $\bullet$	58.5 ± 1.4 $\bullet$
	4-WL	83.5 ± 1.3	78.2 ± 1.1 $\bullet$	77.8 ± 1.2 $\bullet$	73.2 ± 1.2 $\bullet$	69.9 ± 1.4 $\bullet$
	GNN	75.6 ± 0.6	74.3 ± 0.3 $\bullet$	74.1 ± 0.4 $\bullet$	73.3 ± 0.2 $\bullet$	72.8 ± 0.2 $\bullet$
NCI1	GED	73.3 ± 1.1	74.7 ± 1.4	73.3 ± 0.9	72.1 ± 0.9 $\bullet$	68.2 ± 1.6 $\bullet$
	SP	74.3 ± 1.0	75.9 ± 0.9	74.7 ± 1.3	69.8 ± 0.9 $\bullet$	57.9 ± 1.2 $\bullet$
	4-WL	85.8 ± 1.1	80.3 ± 1.1 $\bullet$	77.4 ± 0.5 $\bullet$	73.5 ± 1.1 $\bullet$	68.8 ± 1.2 $\bullet$
	GNN	70.9 ± 0.9	70.8 ± 0.7	69.4 ± 0.4 $\bullet$	69.0 ± 0.2 $\bullet$	67.1 ± 0.3 $\bullet$
NCI109	GED	73.5 ± 2.1	72.4 ± 1.3	73.5 ± 2.3	70.6 ± 0.9 $\bullet$	66.5 ± 0.7 $\bullet$
	SP	73.0 ± 0.8	75.5 ± 1.5 $\circ$	74.2 ± 2.0	70.1 ± 1.5 $\bullet$	57.7 ± 1.9 $\bullet$
	4-WL	86.2 ± 1.3	80.6 ± 1.0 $\bullet$	77.2 ± 1.9 $\bullet$	72.3 ± 0.8 $\bullet$	68.3 ± 1.3 $\bullet$
	GNN	69.9 ± 0.8	69.4 ± 0.5	68.0 ± 0.4 $\bullet$	67.1 ± 0.2 $\bullet$	66.8 ± 0.1 $\bullet$
COLLAB	GED	69.9 ± 1.2	66.3 ± 1.6 $\bullet$	65.2 ± 1.3 $\bullet$	65.1 ± 1.8 $\bullet$	61.9 ± 2.2 $\bullet$
	SP	67.8 ± 1.7	69.2 ± 1.0	69.6 ± 1.0	64.7 ± 2.2 $\bullet$	63.0 ± 1.9 $\bullet$
	4-WL	77.6 ± 0.5	70.6 ± 1.6 $\bullet$	70.4 ± 1.7 $\bullet$	67.2 ± 2.1 $\bullet$	63.8 ± 2.0 $\bullet$
	GNN	57.3 ± 0.3	65.9 ± 0.4 $\circ$	65.4 ± 0.2 $\circ$	62.2 ± 0.5 $\circ$	58.8 ± 0.1 $\circ$
REDDIT-MULTI-5K	GED	24.9 ± 1.1	38.3 ± 1.7 $\circ$	38.1 ± 1.2 $\circ$	34.2 ± 1.9 $\circ$	31.1 ± 2.1 $\circ$
	SP	41.2 ± 0.9	42.5 ± 0.5 $\circ$	43.4 ± 1.8 $\circ$	41.9 ± 1.6	40.0 ± 0.8 $\bullet$
	4-WL	52.3 ± 1.1	52.9 ± 1.2	49.5 ± 0.9 $\bullet$	47.8 ± 1.0 $\bullet$	45.2 ± 0.9 $\bullet$
	GNN	39.9 ± 0.2	43.2 ± 0.3 $\circ$	46.8 ± 0.3 $\circ$	45.4 ± 0.4 $\circ$	42.3 ± 0.7 $\circ$
REDDIT-MULTI-12K	GED	15.3 ± 0.7	14.1 ± 0.5 $\bullet$	24.3 ± 1.2 $\circ$	21.8 ± 1.6 $\circ$	21.0 ± 1.7 $\circ$
	SP	33.8 ± 0.4	32.4 ± 0.7	32.1 ± 0.8	29.5 ± 0.9 $\bullet$	27.5 ± 0.2 $\bullet$
	4-WL	37.0 ± 0.8	37.0 ± 1.2	35.1 ± 0.7i $\bullet$	32.5 ± 0.4 $\bullet$	29.2 ± 0.4 $\bullet$
	GNN	27.9 ± 0.9	32.3 ± 0.3 $\circ$	32.4 ± 0.3 $\circ$	31.2 ± 0.4 $\circ$	28.6 ± 0.4

Next, we compare the four different classifiers for all datasets and reduction factors simultaneously. We observe that when using GED, we obtain results that are statistically equivalent to those obtained on the original graphs in 23 out of 36 cases. When using the GNN (DGCNN), we obtain

results that are statistically equivalent to the original system also in 22 out of 36 cases. When using the SP graph kernel, we obtain results that are statistically equivalent to the original system in 19 out of 36 cases, when using the 4-WL graph kernel, we obtain results that are statistically equivalent in only 3 out of 36 cases. This analysis clearly shows that the 4-WL graph kernel does not cope well with the reduced graphs. One possible explanation for these rather bad results is that the similarity matrix obtained with the 4-WL kernel on the reduced graphs tends to shrink towards zero, as it will be explained in further detail in the following subsection.

To investigate the potential of the proposed graph reduction with respect to computation time, that is answering question **Q.2**, we present the runtimes of all classifiers[5] and reduction factors. In particular, Fig. 5.4 illustrates the average runtime, calculated over five runs for all datasets and classification methods.

In summary, our method demonstrates a clear improvement in runtime for all tested configurations. Already with the first reduction factor (i.e., $\rho = 2$), the results indicate a substantial decrease in computation time. That is, we observe an average reduction of the total runtime of about a factor of two for all datasets and classifiers. As the reduction factor is increased, we further observe a consistent decrease in runtime. When the graphs are strongly reduced (with $\rho = 16$), we see a reduction in the runtime of approximately an order of magnitude on all datasets and classification methods. This improvement is even more pronounced for the large graphs stemming from the REDDIT datasets, where we see a reduction of two orders of magnitude.

In general, we observe that computationally demanding classifiers (i.e., GED and the SP kernel) benefit to a greater extent from using the reduced graphs. However, also the 4-WL kernel, which is computationally more efficient than both GED and SP kernel, shows clear improvements through the use of reduced graphs. It is worth noting that these reductions in runtime are achieved while maintaining, or even improving, the classification accuracy (as we have seen before).

[5]We omit the analysis of the runtime achieved using the GNN classifier, as the classification runtime is negligible once the GNN has been trained, and no clear difference appears in the runtime between the reference system and the systems that operate on the reduced graphs.

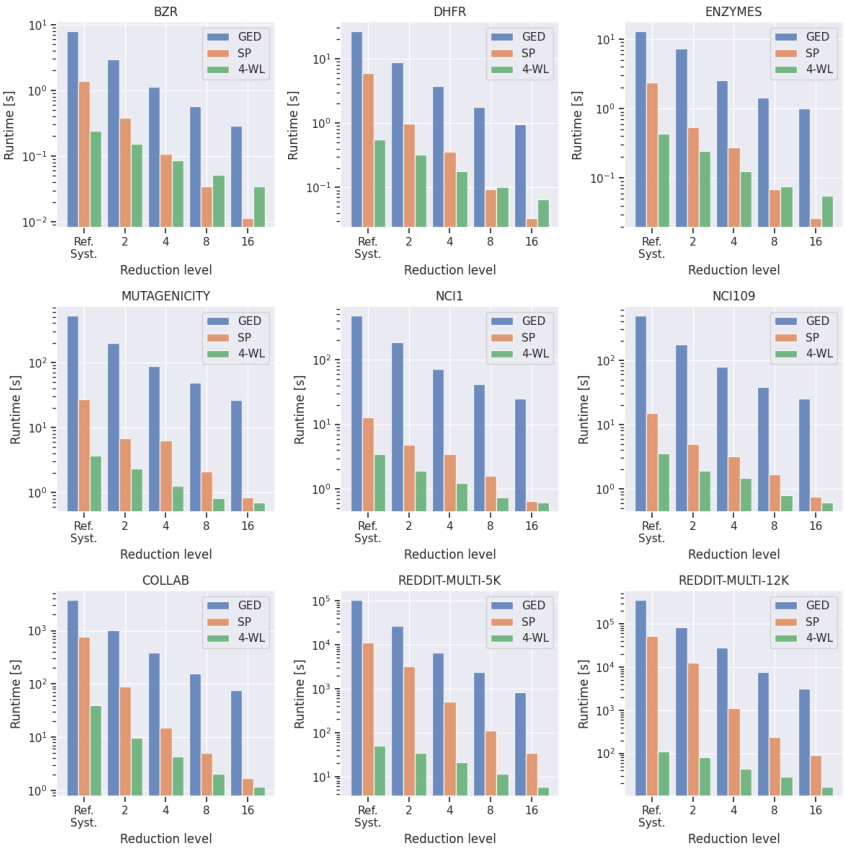


Fig. 5.4: Runtime in seconds on a logarithmic scale for all reduction factors and all datasets using all three classification systems, viz. a k-NN using Graph Edit Distance (GED), as well as the ShortestPath graph kernel (SP) and the 4-Weisfeiler-Lehman graph kernel (4-WL) in conjunction with an SVM.

5.3.4 *Similarity/Dissimilarity Quality Measure*

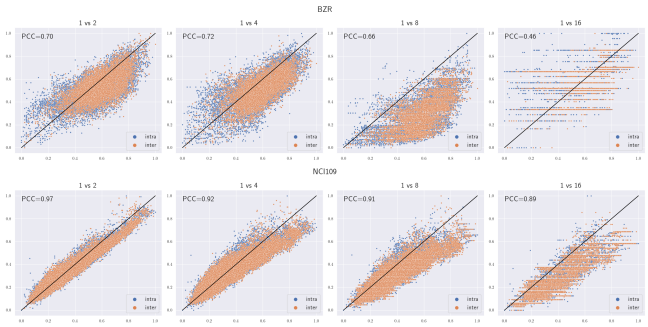
Graph edit distance and graph kernels are often used in conjunction with distance-based classifiers and SVMs, respectively. Thus, it is important to determine whether the similarities/dissimilarities obtained on reduced graphs are reliable. To validate this, we visually compare the similarities/dissimilarities obtained on the original graphs to those obtained

in the reduced graph domains using scatter plots (see Fig. 5.5). Each point in these plots represents a similarity/dissimilarity in the original graph domain (x-axis) and the corresponding distance on the reduced graph (y-axis). The Pearson Correlation Coefficient (PCC) is also shown to indicate the linear correlation between the distances obtained on the original and reduced graphs. The black diagonal represents the one-to-one correspondence between reduced and original graphs. Due to the lack of space, we show results on two datasets only (BZR and NCI1). Note, however, that similar behavior is observed for the remaining datasets as well. Detailed pairwise comparisons of similarity/dissimilarity for the GED, SP and WL classifiers for these datasets can be found in Appendix C.1, C.2, and C.3 respectively.

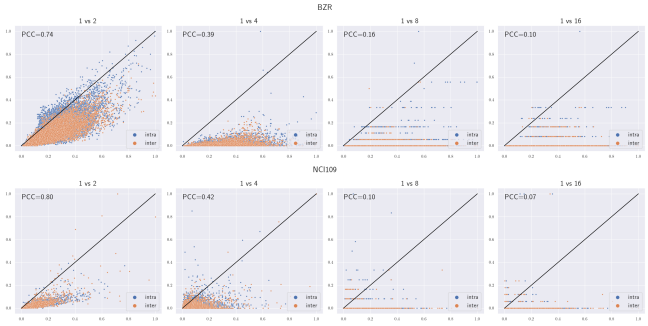
The GEDs obtained in the original graph domain and in the reduced ones appear to be quite correlated. In general, when the distance between two graphs is large in the original domain, it is also relatively large in the reduced domain. Conversely, when the distance is small in the original graph domain, it is also small in the reduced graph domain. Yet, the similarities obtained with both graph kernels are reduced when performed on the weakly reduced graphs (i.e., with $\rho = 2$) and shrink towards zero when the graphs are strongly reduced.

Overall, we observe that GEDs on the reduced graphs retain a coherent correlation and are still suitable for classification, while it may be more difficult to utilize the graph kernel similarities, as they tend to approach zero when the graphs are reduced in their sizes. This accounts for the lower classification accuracy obtained with the reduced graphs, as compared to the original graphs, for both graph kernels.

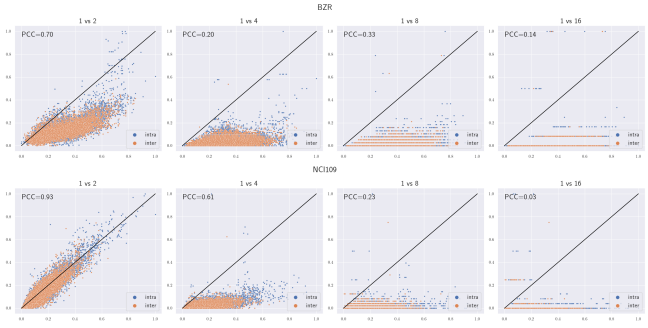
Upon deeper analysis, we observe that diverse graph similarities/dissimilarities in the original graph domain are mapped to the same value in the reduction space. The large number of equal similarities/dissimilarities between different pairs of graphs makes it difficult to discern any pattern in the data. Thus, it is no longer possible to extract trends from the intra-class similarities or dissimilarities (shown with red dots) and inter-class similarities/dissimilarities (shown with blue dots).



(a) GED



(b) SP



(c) 4-WL

Fig. 5.5: Comparison of pairwise similarities/dissimilarities between graphs in the original and the reduced graph domains for both BZR and NCI109 datasets using Graph Edit Distance (GED), the ShortestPath graph kernel (SP), and the 4-Weisfeiler-Lehman graph kernel (4-WL).

Table 5.4: Pearson correlation coefficient (PCC) between similarities (i.e., ShortestPath (SP) and the 4-Weisfeiler-Lehman graph kernel (4-WL))/dissimilarities (i.e., Graph Edit Distance (GED)) obtained in the original and reduced graph domain on all datasets. We show the correlations between the original domain and two reduction factors ($\rho = 2$ and $\rho = 16$).

	GED		**SP**		**4-WL**	
Dataset	$\rho = 2$	$\rho = 16$	$\rho = 2$	$\rho = 16$	$\rho = 2$	$\rho = 16$
BZR	0.70	0.46	0.74	0.06	0.70	0.14
DHFR	0.67	0.71	0.69	0.09	0.76	0.11
ENZYMES	0.91	0.79	0.84	0.02	0.87	0.03
MUTAGENICITY	0.90	0.82	0.87	0.05	0.86	0.04
NCI1	0.97	0.90	0.89	0.07	0.92	0.03
NCI109	0.96	0.88	0.88	0.11	0.92	0.04
COLLAB	0.80	0.90	0.77	0.73	0.91	0.89
REDDIT-MULTI-5K	0.95	0.99	0.94	0.93	0.96	0.91
REDDIT-MULTI-12K	0.94	0.99	0.92	0.90	0.93	0.92

In Table 5.4, we show the PCCs between similarities/dissimilarities obtained in the original and reduced graph domain (we show the correlations between the original domain and two reduction factors only ($\rho = 2$ and $\rho = 16$)).

As already seen in Fig. 5.5, the PCCs of GED remain stable when the graphs are strongly reduced. However, the PCCs tend to decrease when the reduction is increased and approaches zero when using both graph kernels. This trend is consistently visible across all labeled datasets. Yet, on the three unlabeled datasets (i.e., COLLAB, and both REDDIT datasets), we observe that the PCCs remain stable when the reduction is increased. This stability is actually also visible in the corresponding scatter plots. In Fig. 5.6, for instance, we show as an example a comparison of the WL kernel similarities in the original and in the reduced graph domain on the COLLAB dataset.

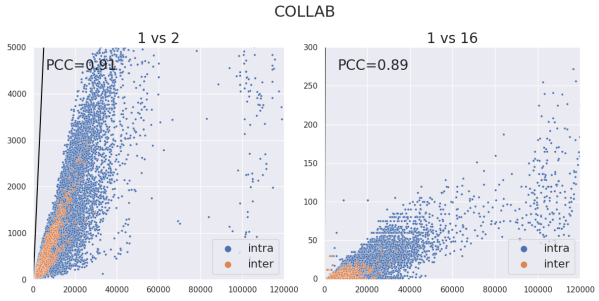


Fig. 5.6: Comparison of pairwise similarities, obtained with the 4-WL graph kernel, between the original graphs and their reduced counterparts (with $\rho = 2$ and $\rho = 16$) for the COLLAB dataset.

5.4 Conclusion

In the present chapter, we propose and research spectral graph clustering as basis for a novel graph reduction framework. In particular, we use spectral graph clustering to first partition the nodes of a graph and then condense each partition of the nodes into supernodes. The benefit of this reduction process is that it can efficiently discover significant communities in the underlying graphs, thus offering accurate clusters for reducing the graphs to their most significant structures.

The proposed procedure to reduce the size of the graphs can be easily controlled by a parameter that defines the size of the resulting graphs (basically by the number of clusters to be found in the graph). The general goal of the proposed reduction framework is to speed up the computation of standard graph classification algorithms while maintaining satisfactory classification accuracy. In order to demonstrate the effectiveness of the proposed graph reduction technique, we compare the classification accuracy as well as the computation time on the original and reduced graphs with four different classifiers (GED with a k-NN as well as SP, 4-WL kernel with SVMs, and GNN).

We conduct a comprehensive experimental evaluation on nine real-world graph datasets. Our experimental evaluation shows that while the classification accuracy decreases with the use of reduced graphs in general, the resulting accuracies are still comparable to those obtained with the original-sized graphs. In more detail, we are able to draw the following conclusions regarding the classification accuracy.

1. We find that the use of GED in conjunction with a k-NN on reduced graphs achieves comparable classification accuracies to that of the original graphs in the majority of the cases.
2. We also show that the SP kernel works quite well on the weakly reduced graphs (i.e., $\rho \in \{2, 4\}$), as the classification accuracy remains statistically similar to those achieved with the reference system on the majority of datasets.
3. The reduced graphs have the least beneficial effect when used in conjunction with the 4-WL kernel, as the classification accuracy drops statistically significantly for almost all the reduction factors and datasets.
4. We observe that GEDs obtained in the original and the reduced graph domain remain in the same order of magnitude, while the similarities (obtained with the SP and the 4-WL graph kernel) shrink toward zero as the reduction factor is increased.

The empirical evaluation also shows a significant decrease in computation time across all datasets and graph classifiers. That is, we observe a reduction of the runtime of about a factor of two when using weakly reduced graphs (i.e., $\rho = 2$) and at least one order of magnitude (and in some cases more than two orders of magnitude) when the graphs are strongly reduced (i.e., $\rho = 16$). These empirical results suggest that our approach may provide significant time savings compared to existing methods, which could be particularly useful in contexts where the time required for classification is a major limiting factor.

Further Graph Reduction Methods

6

> Car enfin, il faut en prendre son parti et se dire une fois pour toutes, que la bourgeoisie est condamnée à être chaque jour plus hargneuse, plus ouvertement féroce, plus dénuée de pudeur, plus sommairement barbare; que c'est une loi implacable que toute classe décadente se voit transformée en réceptacle où affluent toutes les eaux sales de l'histoire;
>
> *Discours sur le Colonialisme* (1950),
> Aimé Césaire

6.1 Introduction

As extensively detailed in Chapter 2, graphs are recognized as robust and flexible data representation in pattern recognition and related fields (e.g. in [172; 173; 174]). At the core of graph-based pattern recognition lies a central task known as *graph matching* [4; 5], which is known to be very time consuming in general.

As presented in Chapter 4 and 5, a possible idea to better handle graphs in pattern recognition, is to work with size-reduced versions of the original graphs [33; 112]. *Graph reduction methods* produce a graph $G_R = (V_R, E_R)$ from the original graph $G = (V, E)$, with reduced node and/or edge sets $|V_R|$ and/or $|E_R|$, respectively. The aim of such a reduction is to obtain a smaller graph G_R that maintains the main topology and properties of the original graph G.

In the present chapter, we introduce in two separate sections (Sections 6.2 and 6.3) two novel pattern recognition procedures that rely on

novel graph reduction frameworks.

- The first framework is presented in Section 6.2 and consists of two basic parts. First, we substantially reduce the size of the original graphs by means of a graph neural network. Eventually, we use the reduced graphs in conjunction with GED and a distance-based classifier. On five datasets we empirically confirm the benefit of the novel reduction scheme regarding both classification performance and computation time.
- The second framework is presented in Section 6.3. The core concept of this framework revolves around the extraction of graph-inherent regularity patterns within a classification scheme. To this end, we propose a framework based on a compressor-based metric associated with a k-Nearest Neighbor classifier. The compressor-based metric aims to identify regularities in the compressed graphs and assigns smaller distances to graphs that appear comparable and are assumed to belong to the same class. To evaluate the effectiveness of the proposed graph matching framework, we perform a series of evaluations on eleven datasets.

Note that both sections are based on preliminary works presented in a conference paper [47] and in a journal paper [48], respectively. Also note that in the present chapter, the first graph reduction method is a graph summarization method similar to the method presented in Chapter 4, while the second method is a is compression-based approach.

6.2 Graph Reduction Neural Networks for Structural Pattern Recognition

In the present section, we discuss a novel graph reduction method [47] that learns the relevant features of the graph topology by means of *Graph Neural Networks* (GNN). Research on GNNs is a rapidly emerging field in structural pattern recognition [100; 175]. The general idea of GNNs is to learn vector representations for nodes and/or (sub-)graphs such that given criteria are optimized. From a broader perspective, GNNs can be seen as a mapping from a graph domain into a vector space. Once an embedding is computed for a node and/or a (sub-)graph, the vectorial representation can be used to solve downstream tasks such as node or graph classification. For more detailed information about the concept of GNN, we refer to

Subsection 2.4.3.

The contribution of the proposed reduction framework is twofold. First, we show how a learned GNN model can be used to select the nodes of a graph to be removed (instead of applying explicitly defined rules for graph coarsening as proposed in [44], for instance). Once the GNN model is trained, we are able to readily reduce arbitrary graphs from various datasets. Second, we use the learned graph reductions in a graph matching scenario. In particular, we approximate the GED on the reduced graphs in order to solve an underlying graph classification task. To the best of our knowledge, such an approach – that combines GNNs for graph reduction with fast approximate GED – has not been proposed before.

The remainder of the present section is structured as follows. In Subsection 6.2.1 and 6.2.2, we delve into the proposed reduction algorithm based on GNNs as well as the complete graph matching framework, respectively. In Subsection 6.2.3, we provide a comprehensive overview of our experimental evaluation on five distinct graph datasets. Subsequently, in Subsection 6.2.4, we evaluate and analyze the structure and characteristics of the reduced graphs. This is followed by a large-scale classification experiment detailed in Subsection 6.2.5. Finally, in Subsection 6.2.6, we conduct an ablation study.

6.2.1 *Graph Reduction Neural Network (GReNN)*

In contrast with the proposed reduction methods described in Chapter 4 and 5, we aim for a reduction method that learns the nodes to be deleted. Actually, a learned model has the advantage that once it is trained it can readily be applied on unseen data. In particular, we propose to use a GNN to learn the graph reduction on a training set and eventually apply the model to the complete dataset.

In Table 6.1, we present an overview of the architecture of the proposed *Graph Reduction Neural Network* (GReNN). We use a mix of graph convolution layers (*GCNConv*) [100] to learn the node representation, and *gPool layers* [176] as node sampling method.

The gPool layer is based upon the projection of the feature vector $\vec{x}_u$ (attached to node u) on a trainable vector $\vec{p}$ (i.e., $y_u = \vec{x}_u\vec{p}/\|\vec{p}\|$). The scalar value y_u quantifies the retained information when projecting the feature vector of node $u \in V$ onto the direction of vector $\vec{p}$. The node sampling is done according to the largest scalar projection value in order to preserve the maximum of information.

Table 6.1: Overview of the architecture of our graph reduction scheme GReNN for both the 50% and 25% setting. Reduced graphs are obtained after the last graph convolution layer (marked with an asterisk *)

GReNN-50	GReNN-25
GCNConv($\mathbb{R}^n$, 64)	
gPool(0.5)	gPool(0.5)
GCNConv(64, 64)	GCNConv(64, 64)
-	gPool(0.5)
-	GCNConv(64, 64)
GCNConv(64, 64)*	
POOLING	
Linear(64, #classes)	

The gPool layer also incorporates a graph connectivity augmentation that turns out to be particularly advantageous in our framework. This augmentation method is originally employed to improve the information flow in subsequent layers. In our case it prevents the creation of reduced graphs that are sparsely connected or even completely edge-free.

We propose two versions of the reduction model, termed GReNN-50 and GReNN-25 (where the number indicates the percentage of remaining nodes). For GReNN-50 we apply a pooling ratio of 0.5 (means that we are deleting 50% of the nodes). For GReNN-25 we evaluate two ways to reduce the graphs. First, we train the network from scratch with a pooling ratio of 0.25. Second, we train the model with a reduction level of 50%, freeze the already trained layers, and add a new layer (again with pooling ratio 0.5) to obtain graphs with 25% of the nodes. The newly updated network is then trained with half of the original epochs. Preliminary experiments show that for the architecture of GReNN-25 the two step optimization process actually achieves better results than training from scratch. Hence, we apply this specific architecture of an extra pooling layer in our framework.

The last two layers of our model are used for training only. That is, they are used to produce a graph embedding which in turn is used to back-propagate the classification error during the training of the reduction model. The reduced graphs – actually used for classification part of our framework (detailed in the next Subsection) – are obtained after the last graph convolution layer (marked with an asterisk in Table 6.1).

6.2.2 *Graph Matching on GNN Reduced Graphs*

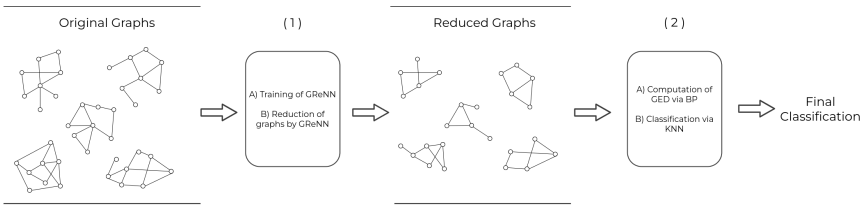


Fig. 6.1: The proposed graph matching framework consists of two basic parts. (1) Training of a GNN and reduction of the graphs with the optimized GNN. (2) Graph classification with a k-NN using approximated GED that is computed on the reduced graphs.

Major goal of the proposed method is to make graph-based pattern recognition more efficient. To this end, we combine two complementary research directions, viz. GNNs (for graph reduction) and approximate GED (for graph matching). Hence, the proposed framework can be split into two main parts as illustrated in Fig. 6.1.

- In part (1), we aim at producing reduced graphs via GNNs. This part includes both training of the network model and the actual reduction of the graphs. Goal of the GNN based reduction is to obtain strongly reduced graphs, that are still representative enough such that they can be used for pattern recognition tasks.
- In part (2) of our framework, we use the reduced graphs in a classification scenario by means of approximate graph matching. It is our main hypothesis that the power and flexibility of GED in conjunction with the strong generalization and learning power of GNNs lead to a fast and accurate graph recognition framework.

For classification, we employ a *k-Nearest Neighbor* classifier (k-NN) that operates on the GReNN reduced graphs. In particular, we compute the GED on GReNN reduced graphs using the fast approximation algorithm *BP-GED* [11] (as thoroughly described in Section 2.4).

Rather than a k-NN any other dissimilarity based classification algorithm could be employed as well in our framework. We feel, however, that the k-NN is particularly well suited because of its direct use of the dissimilarity information without any additional training. The same accounts somehow for the GED computation via algorithm BP. That is, any other

approximation algorithm could be used for this task (e.g., suboptimal algorithms surveyed in [4]).

6.2.3 *Datasets and Experimental Setup*

For the experimental evaluation, we use five standard datasets, namely, DD, ENZYMES, MUTAGENICITY, NCI1, and PROTEINS. Some graph properties, such as the number of graphs, number of classes, and the average number of nodes and edges per graph, are given in Table 6.2. For details about these datasets, please refer to Chapter 3, where a comprehensive description is provided for each dataset.

The datasets used in the present section are split into three disjoint sets for training, validation, and testing[1]. The splitting is carried out once at the beginning of our evaluation, and – for the sake of consistency – we maintain this splitting throughout each run. Yet, it is known that the performance of neural networks depend on the initialization of its weights and the actual split of the dataset [177]. To disentangle this random factor from our method and be sure that our reduction scheme works properly, each experiment is repeated five times with different weight initialization and different dataset splits.

The training and optimization of the hyperparameters for the GReNNs are exclusively achieved on the training and validation sets, respectively. The same accounts for the optimization of the k-NN classification and GED computation via BP in the second part of our framework.

For the training of the GReNNs, we use the hyperparameters as originally proposed in [176]. Yet, we reduce the number of epochs from 200 to 20 as preliminary experiments show limited improvements when using more than 20 epochs. For optimizing the k-NN we evaluate the number of nearest neighbors $k \in \{1, 3, 5\}$ and for the computation of GED we optimize a factor α that weights the relative importance between node and edge edit costs in the range $]0, 1]$ with a step size of 0.05.

6.2.4 *Analysis of the Structure of the Reduced Graphs*

When reducing a graph by removing nodes and their incident edges one might obtain edge-free, or at least sparsely connected graphs. To confirm that the proposed reduction does not produce graphs that consists of sets

[1] During the splitting of the datasets we sample the graphs such that the class balance is preserved in each subset.

Table 6.2: Properties of the graph datasets. We show the number of graphs ($|G|$) with the sizes of the training, validation, and test set ($|\mathcal{G}_{tr}|$, $|\mathcal{G}_{va}|$, $|\mathcal{G}_{te}|$), the number of classes ($|\Omega|$) and the average number of nodes and edges per graph ($\emptyset|V|$, $\emptyset|E|$).

	Graph Dataset Property				
Dataset	$\|\mathcal{G}\|$	($\|\mathcal{G}_{tr}\|$, $\|\mathcal{G}_{va}\|$, $\|\mathcal{G}_{te}\|$)	$\|\Omega\|$	$\emptyset\|V\|$	$\emptyset\|E\|$
DD	1,178	(707, 235, 236)	2	284.3	715.6
ENZYMES	600	(360, 120, 120)	6	32.6	62.1
MUTAGENICITY	4,337	(2,600, 867, 870)	2	30.3	30.8
NCI1	4,110	(2,466, 822, 822)	2	29.9	32.3
PROTEINS	1,113	(660, 220, 223)	2	39.1	72.8

of unconnected nodes only, we start our evaluation by thoroughly analyzing the reduced graphs. The following evaluations and visualizations are related to one dataset only, viz. DD. However, on the other datasets we obtain similar results (available in Appendix D.1 and D.2) and make similar observations.

In Fig. 6.2 we visualize the number of graphs on the y-axis that have a certain number of connected components (x-axis) in a histogram. We compare the original graphs with the reduced graphs obtained with GReNN-50 and GReNN-25. We observe that the vast majority of original graphs consist of one connected component. On the other hand, the reduced graphs tend to have more than only one connected component. However, we see that the number of connected components is smaller than, or equal to, five for about 80% of the graphs (for both reduction levels).

We also analyze the number of isolated nodes per graph obtained after reduction on all datasets. We can report that on three datasets (MUTAGENICITY, ENZYMES, and PROTEINS) there are less than three isolated nodes per graph for both reductions. On NCI1 and DD we observe a maximum of seven isolated nodes per graph when reducing the graphs with GReNN-25. This implies that the connected components of the reduced graphs do not mainly consist of single nodes but rather of connected subgraphs. Hence, the reduced graphs maintain their connectivity in general, which is mainly due to the connectivity augmentation method of the gPool layer (as discussed in Subsection 6.2.2).

In Fig. 6.3 we illustrate the effects of our graph reduction on a sample graph. Reductions from the original graph to 50% and 25% of the available

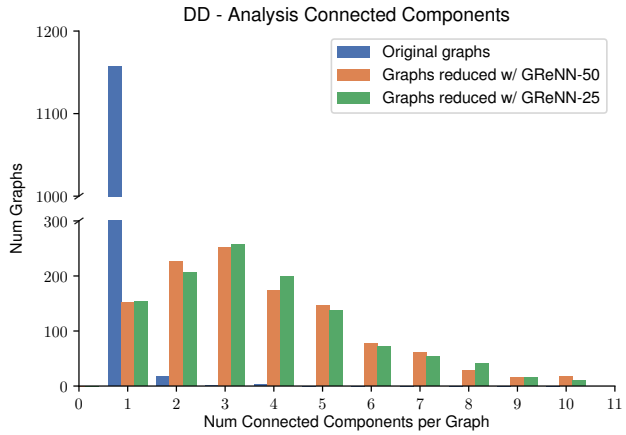
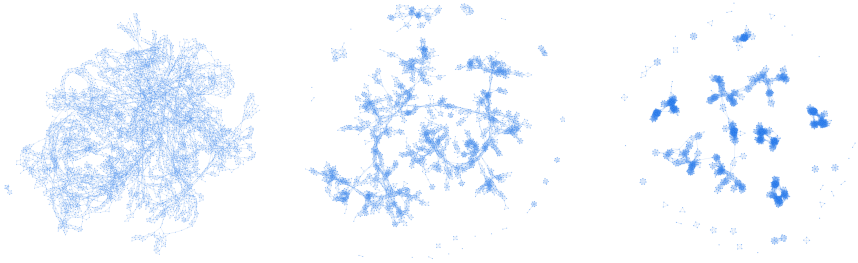


Fig. 6.2: Histogram that shows the number of graphs (on the y-axis) that have a given number of connected components per graph (on the x-axis) on the dataset DD (the frequencies of graphs with more than 11 connected components are clipped for the sake of conciseness).

nodes clearly lead to an increased number of connected components. However, we also observe the effects of the built-in edge augmentation. That is, the remaining nodes stay — in general — highly connected via dense and local edge structures.



(a) Original Graph (b) GReNN-50 (c) GReNN-25

Fig. 6.3: Example of an original graph and the corresponding reduced graphs via GReNN-50 and GReNN-25.

Table 6.3: Classification accuracies obtained on test sets with the reference system and our novel framework (GReNN-50, GReNN-25). We also show the relative speed up of the matching times. (•: indicates a statistically deterioration compared to the reference system using a t-test with p-value= 0.05.)

Dataset	Ref. System	GReNN-50		GreNN-25	
	Acc [%]	Acc [%]	Speed Up	Acc [%]	Speed Up
DD	73.4±1.7	74.3±2.2	2.2	70.1±2.0	12.6
ENZYMES	43.5±5.5	43.5±2.9	2.1	33.2±5.1 •	4.6
MUTAGENICITY	74.5±1.6	72.6±2.6	3.6	68.1±2.3 •	8.5
NCI1	73.3±1.1	71.6±1.7	3.7	63.8±1.6 •	9.9
PROTEINS	71.8±4.2	70.5±1.6	3.4	69.3±1.6	8.8

6.2.5 *Classification Results*

Next, we conduct classification experiments in order to evaluate our complete framework (using both parts (1) and (2)). The first experiment compares our framework (GReNN-50 and GReNN-25) with a k-NN classifier that operates on the original graphs (termed reference system from now on).

In Table 6.3, we show the classification accuracies of all systems and the speed up of the matching times achieved with our framework (compared to the reference system). With GreNN-50, we achieve quite similar classification accuracies as the reference system on four out of five datasets. As expected, we also observe a clear speed up on all datasets (we observe speed ups by factors of about three).

The improvement regarding computation time is even more pronounced when using graphs reduced to 25% of their original size. That is, with GReNN-25 we reduce the runtime by factors of about ten on four out of five datasets (on ENZYMES the speed up is about a factor of five).

Using such a strong reduction of the graphs, however, leads to more substantial deteriorations w.r.t the classification accuracy. That is, with GreNN-25 we observe three statistically significant deteriorations. On the other two datasets, however, no statistically significant difference is visible between our system and the reference system. In general, the results obtained are in a fairly similar range to the original results – this is quite astonishing, considering that we are using 25% of the nodes only.

6.2.6 *Ablation Study*

Finally, we conduct an ablation study in order to investigate the usefulness of the two separate parts of our framework (for the sake of conciseness we use GReNN-50 only). We compare the complete framework with the following systems

- *Without-1*: This refers to a system in which we replace the first part of our framework (1) with a random reduction of the graphs to 50% of the available nodes. This system helps us to verify whether or not our framework actually benefits from the elaborated GNN reduction.
- *Without-2*: This system refers to the proposed GNN architecture of the GReNN that is directly employed for graph classification without taking the detour of GED computation (that is, we omit part (2) of our framework). This system helps us to verify whether the proposed framework actually benefits from the combination of GNN reductions and GED computations.

In Table 6.4 we observe that our novel framework GReNN-50 outperforms the system Without-1 on all five data sets (three of the five improvements are statistically significant). On four out of five data sets GReNN-50 also outperforms the second reference system Without-2 (three out of five improvements are statistically significant).

The main finding of this comparison is twofold. First, it clearly shows that our novel learning-based reduction scheme outperforms a naïve graph reduction, and second, the proposed graph reduction based on GNNs in conjunction with GED computations is clearly beneficial as it outperforms the isolated GNN in general.

Table 6.4: Ablation study where we compare the classification accuracy obtained with and without the use of the two parts (1) and (2) of our framework. Symbols ①/② indicate a statistically significant improvement with and without the use of the two parts (1) and (2), respectively (using a t-test with p-value = 0.05). The best result per dataset is highlighted.

	Dataset				
	DD	**ENZYMES**	**MUTAGENICTY**	**NCI1**	**PROTEINS**
Without-1	72.5±1.7	23.2±3.3	64.4±1.3	59.5±0.9	69.7±1.5
Without-2	67.9±3.1	26.1±3.7	**73.3±4.4**	67.0±2.3	70.2±2.4
Ours	**74.3±2.2** -/②	**43.5±2.9** ①/②	72.6±2.6 ①/-	**71.6±1.7** ①/②	**70.5±1.6** -/-

6.3 Graph Classification With Normalized Compression Distance

In recent years, interest in the use of compression-based distances for graph classification purposes has increased. One algorithm that embodies this approach is, for instance, *Graphitour* [178]. This algorithm works with an iterative contraction of similar edges, focusing on the selection of the most frequently occurring edges. To achieve this, Graphitour solves an instance of the maximum cardinality matching problem, trying to find as many edges as possible without any two edges sharing common nodes. In [179], the authors use a modified version of the *Graphitour* algorithm in conjunction with the *Normalized Compression Distance* (NCD) [180]. The goal of this procedure is to ensure that two isomorphic graphs are compressed in the same way. In [181], a parameter-free method is proposed in the context of low resource text classification. In this approach, the authors use the NCD to compute the distance between two texts. It is noteworthy that the results obtained are comparable to those achieved by advanced, yet computationally expensive, text classification algorithms (such as BERT [182]).

In the present section, we introduce and research a novel method for graph matching that consists of an adaptation of a lossless compressor-based distance metric [183] to the graph domain. This particular metric offers the inherent ability to capture regularity patterns in the underlying data, which in turn motivates our effort to adapt its properties to a graph matching scenario. Roughly speaking, our novel graph matching method is based on two steps.

(1) First, we extract string representations from the underlying graphs.

(2) Second, we compute a lossless compression distance between the extracted strings.

The contribution of the presented section is twofold. First, we formally demonstrate how a lossless compression distance can be used in the context of graph matching. In doing so we have to overcome the challenge of extracting a string from a graph posed by the exponential number of node arrangements in a graph. In our novel method, we propose to rearrange the nodes of the graphs based on deterministic node permutation algorithms. Second, in an empirical evaluation, we show that the proposed framework can produce results comparable to those of a standard graph matching framework (i.e., GED), yet, substantially faster.

The remainder of the present paper is organized as follows. In Subsection 6.3.1, we present the *Normalized Compression Distance* (*NCD*) and review the general idea behind the *Kolmogorov complexity* which is at the heart of our novel graph matching framework. In Subsection 6.3.2, we explain in details the main steps of the proposed graph classification framework. Finally, in Subsection 6.3.3, we evaluate different parts of our framework. For instance, we analyze how the compression rate influences the classification accuracy and compare the resulting runtimes across different compression rates.

6.3.1 *The Normalized Compression Distance (NCD)*

The *Kolmogorov Complexity* [184] is a measure of complexity, or compressibility, of a finite string of symbols. Essentially, it formalizes the idea that the complexity of a binary string is closely linked to the length of the shortest program that can generate this string. That is, the Kolmogorov complexity $K_U(x)$ of a binary string x is the length of the shortest binary program p that outputs string x using a universal Turing Machine U [184]. Formally,

$$K_U(x) = min\{|p|, U(p) = x\} \tag{6.1}$$

The use of a universal Turing Machine U ensures that Eq. 6.1 is independent of any particular machine or programming language[2]. Intuitively, $K(x)$ is the minimum amount of information required to generate x by an

[2]For the sake of convenience, we write to $K(\cdot)$ rather than $K_U(\cdot)$ from now.

effective process, i.e., the shorter the program is, the more compressible or less complex the string x is.

The authors of [185] derived the *Algorithmic Information Distance* (AID) between strings x and y with the use of the Kolmogorov complexity. The AID is defined as the length of the shorter binary program that computes both x from y and y from x. By ensuring that the program is the shortest, it is guaranteed that it makes optimal use of any redundancy between the information needed to compute x from y and vice versa.

Formally, the $AID(x, y)$ is defined as follows.

$$\begin{aligned} AID(x, y) &= \max(K(x|y), K(y|x)) \\ &= K(xy) - \min(K(x), K(y)), \end{aligned} \tag{6.2}$$

where $K(x|y)$ is the conditional Kolmogorov complexity of x relative to y (that is, the length of a shortest program to compute x if y is given as an auxiliary input to the computation), while $K(xy)$ represent the Kolmogorov complexity of the concatenation of string x and y.

The AID leverages the absolute measure of complexity through the Kolmogorov complexity $K(x)$. This ensures that the distance is independent of the choice of the compressor. However, the Kolmogorov complexity $K(x)$ is actually a theoretical concept only, which is difficult to calculate in practice. Hence, the *Normalized Compression Distance* (NCD) [180] was proposed that uses a compression algorithm $C(\cdot)$ instead of the Kolmogorov complexity $K(\cdot)$.

NCD relies on the application of data compression algorithms to compress two strings independently (exploiting redundancies and patterns). The NCD computation can be split into the following three steps.

(1) Each string is compressed separately using a lossless compression algorithm $C(\cdot)$.
(2) The sizes of the compressed representations and the size of their concatenation is compared. The smaller the difference in the compressed representations, the more similar the two strings are considered to be.
(3) The resulting difference is normalized to ensure that the NCD falls within a specific range (commonly between 0 and 1). This normalization step makes the metric more interpretable and allows for comparisons across different types and sizes of data.

Formally, the $NCD(x, y)$ between string x and y is defined by

$$NCD(x, y) = \frac{C(xy) - \min(C(x), C(y))}{\max(C(x), C(y))} \tag{6.3}$$

where $C(x)$ and $C(y)$ represent the sizes of the compressed representations of strings x and y, and $C(xy)$ is the size of the compressed representation when the two strings are concatenated.

Note that higher compression rates generally result in $C(\cdot)$ being a closer approximation of $K(\cdot)$. This in turn means that the higher the compression rate can be defined, the better NCD (from Eq. 6.3) approximates AID (from Eq. 6.2).

In essence, the NCD is a measure of dissimilarity between two data objects based on how well their information content can be compressed and represented in a concise form. Lower $NCD(x, y)$ values generally indicate higher similarity between x and y, while higher values suggest greater dissimilarity.

6.3.2 *Graph Matching via NCD*

In the present subsection, we propose a novel graph matching framework based on the NCD. As formally introduced in Subsection 6.3.1, the NCD is a computable approximation of the normalized information distance that allows us to compute the compression distance between two strings x and y. Consequently, to enable the application of the NCD within a graph domain $\mathcal{G}$, we need to convert graphs $G \in \mathcal{G}$ into strings x_G.

In the proposed approach, the transformation of graphs to strings relies on using the adjacency matrices $\mathbf{A} = (a_{ij})_{n \times n}$ of the underlying graphs $G \in \mathcal{G}$ (with n nodes). First, we extract the upper diagonal part of the adjacency matrix $\mathbf{A}$ and flatten it into a binary string x_G. Formally, we define this string as

$$x_G = a_{12}, \ldots, a_{1n}, a_{23}, \ldots, a_{2n}, \ldots, a_{n-1n}.$$

Then, to preserve potential node labels, we append a one-hot encoded vector that represents the node labels to the end of x_G (using the same node ordering as used in $\mathbf{A}$). A visual summary of this procedure can be found in Fig. 6.4.

The primary goal of any lossless compression algorithm is to minimize the overall bit size of the data by assigning shorter bit codes to more probable symbols (and vice versa). Given that graphs within the same class

have more common regularities than graphs from different classes, our hypothesis states that performing compression on similar graphs will result in similar compressions.

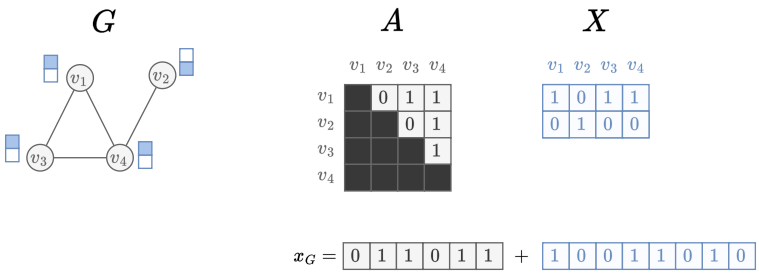


Fig. 6.4: Example of a graph G with labels on the nodes that is transformed into a binary string x_G. To this end, the upper part of the adjacency matrix $\mathbf{A}$ is concatenated with the flattened matrix $\mathbf{X}$ containing the one-hot encoded node labels.

One of the pivotal pillars of our approach is the use of lossless compression algorithms, which feature the ability to capture and encode regularity patterns within the data. However, both the encoding of the regularities and the resulting compression crucially depend on the actual arrangement of the nodes in the adjacency matrix. Consequently, this dependency can potentially lead to different distances for pairs of graphs by simply changing their node orderings. Given the exponential number of possible adjacency matrices $\mathbf{A}$ for a given graph $G \in \mathcal{G}$ (resulting from different node orderings) this problem is further exacerbated.

Essentially, our goal is to develop a permutation-invariant method that achieves the same compression regardless of the original node arrangement. Hence, we propose to first compute a deterministic representation of the adjacency matrix $\mathbf{A}$. As it is uncertain in the first place which node permutation is useful in our context, we treat the permutation as a free hyperparameter. In this regard, we propose and evaluate the five node permutation algorithms presented below (stemming from two categories, viz. *direct node rearrangement algorithms* and *community detection algorithms*).

Direct Node Rearrangement Algorithm

- *Cuthill-Mckee* [186] is an algorithm that reorders sparse matrices. The idea is to permute the nodes of a graph such that the *bandwidth* of the resulting adjacency matrix is minimized. The bandwidth of a matrix is the maximum distance between non-zero elements along any diagonal.

Community Detection Algorithm

- *Girvan Newman* [187] is a hierarchical algorithm that finds communities in a network by iteratively removing edges based on centrality indices induced by the betweenness measure [162].
- *Label Propagation* [188] operates on the idea that nodes within the same community should share equal labels. The algorithm iteratively propagates the labels through the edges of the graph. In each iteration, the nodes are assigned the label that is most prevalent among their neighbors. This process continues until the labels of all nodes have converged, indicating the resulting communities.
- *Louvain* [189] is a greedy algorithm that iteratively refines the *modularity* in two subsequent steps (modularity is an often used measure of the quality of a partition of a network). In a first step, it identifies small communities by locally optimizing modularity across all nodes. Then, in a second step, each small community is merged into a node. These two steps are repeated until no more changes in the graph occur.
- *Scalable Community Detection* (*SCD*) [190] is an efficient version of the Label Propagation algorithm [188]. Its distinctive feature is that SCD partitions the graph by maximizing the *Weighted Community Clustering* (WCC) [191], which is a triangle-based community detection metric.

Given a certain permutation algorithm and two graphs G and G', we now define the *Normalized Graph Compression Distance* (NGCD), as follows

$$NGCD(G, G') = \frac{C(x_G x_{G'}) - \min(C(x_G), C(x_{G'}))}{\max(C(x_G), C(x_{G'}))}, \tag{6.4}$$

where x_G and $x_{G'}$ are the string representations extracted from the adjacency matrices $\mathbf{A}$ and $\mathbf{A'}$ arranged according to the given permutation algorithm.

Using the NGCD: $\mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}$, any distance-based classification can be applied to the graphs from $\mathcal{G}$. In this paper we use – for the sake of simplicity – the k-nearest neighbor classifier.

In order to speed up the distance computation in a k-nearest neighbor classifier, we propose to use the following heuristic procedure. First, we divide the training set into bins of graphs based on their number of nodes. Subsequently, for a given test graph $G^{(t)} = (V^{(t)}, E^{(t)})$ with $|V^{(t)}|$ nodes, we first retrieve training graphs from $\mathcal{G}_{tr}$ that have the same number of nodes as the test graph $G^{(t)}$. Then, while the maximum number of graphs $\epsilon \cdot |\mathcal{G}_{tr}|$ is not yet selected (with $\epsilon \in [0, 1]$ being a user-defined parameter), we add the training graphs to our selection $\mathcal{G}_{sel}$ that have $|V^{(t)}| \pm i$ nodes with $i = 1, 2, \ldots$. In Alg. 5 this idea is formalized.

After obtaining the training set corresponding to graph $G^{(t)}$, the NGCD is computed between graph $G^{(t)}$ and any graph in $\mathcal{G}_{sel}$.

Algorithm 5: Heuristic Selection of Training Graphs

Input: Parameter ϵ, training graphs $\mathcal{G}_{tr}$, and test graph $G^{(t)} = (V^{(t)}, E^{(t)})$

Output: Selection of training graphs $\mathcal{G}_{sel}$ for $G^{(t)}$

```
i = 0
G_sel = {}
while |G_sel| < ε · |G_tr| do
    G_sel = G_sel ∪ {G ∈ G_tr with |V^(t)| ± i nodes }
    i += 1
```

6.3.3 *Empirical Evaluation*

6.3.3.1 *Experimental Setup*

We empirically evaluate our novel graph matching approach on eleven real-world datasets from the TUDataset graph repository [128]. Details on the number of graphs, classes, as well as the average numbers of nodes and edges per graph are presented in Table 6.7 for each dataset. Eight datasets (BZR, BZR-MD, COX2, COX2-MD, DHFR, DHFR-MD, MUTAG, and NCI1) consists of graphs representing molecules stemming from two classes (based on their potential effects or activities). Two datasets, namely ENZYMES and OHSU, are from the bioinformatics domain and contain graphs

Table 6.5: Properties of the graph datasets. We show the number of graphs ($|\mathcal{G}|$), the number of classes ($|\Omega|$) and the average number of nodes and edges per graph ($\emptyset|V|$, $\emptyset|E|$).

	Graph Dataset Property			
Dataset	$\|\mathcal{G}\|$	$\|\Omega\|$	$\emptyset\|V\|$	$\emptyset\|E\|$
BZR	405	2	35.8	38.4
BZR-MD	306	2	21.3	225.1
COX2	467	2	41.2	43.4
COX2-MD	303	2	26.3	335.1
DHFR	756	2	42.4	44.5
DHFR-MD	393	2	23.9	283.0
ENZYMES	600	6	32.6	62.1
IMDB-BINARY	1,000	2	19.8	96.5
MUTAG	188	2	17.9	19.8
NCI1	4,110	2	29.9	32.3
OHSU	79	2	82.0	199.7

that represent proteins and segmented brain scans, respectively. The remaining dataset, IMDB-BINARY, includes graphs representing social media networks from two classes.

In our evaluation, we use a 10-fold cross-validation scheme and report the balanced classification accuracy achieved on a test set. In particular, the training and optimization of the NGCD hyperparameters are performed in a 3-fold inner loop exclusively on the training and validation sets.

The selection of the node permutation (among the node permutation algorithms presented above) is treated as a free parameter and is thus also optimized. In our experiments, if two or more node permutation methods achieve the best classification accuracy, the method with the shorter runtime is selected. Additionally, we optimize the heuristic parameter $\epsilon \in \{0.1, 0.3, 0.5\}$ which represents the percentage of remaining graphs in the selected training set (see Alg. 5) and the number of nearest neighbors $k \in \{1, 3, 5\}$ for k-NN classification.

For the computation of the NGCD we use the python module *gzip* as compression function $C(\cdot)$ [3]. The gzip algorithm uses the dictionary-based compression algorithm called *LZ77* [192]. This compression algorithm re-

[3]Note that any other compression algorithm can be used.

Table 6.6: Properties of the graph datasets. We show the number of graphs ($|\mathcal{G}|$), the number of classes ($|\Omega|$) and the average number of nodes and edges per graph ($\emptyset|V|$, $\emptyset|E|$).

Dataset	Graph Dataset Property			
	$\|\mathcal{G}\|$	$\|\Omega\|$	$\emptyset\|V\|$	$\emptyset\|E\|$
AIDS	2,000	2	9.5	10.0
BZR	405	2	35.8	38.4
BZR-MD	306	2	21.3	225.1
COX2	467	2	41.2	43.4
COX2-MD	303	2	26.3	335.1
DD	1,178	2	284.3	715.6
DHFR	756	2	42.4	44.5
DHFR-MD	393	2	23.9	283.0
ENZYMES	600	6	32.6	62.1
MUTAG	188	2	17.9	19.8
MUTAGENICITY	4,337	2	30.3	30.8
NCI1	4,110	2	29.9	32.3
NCI109	4,127	2	29.7	32.1
OHSU	79	2	82.0	199.7
PROTEINS	1,113	2	39.1	72.8

places repeated patterns in a string with references to their entries in a dictionary. The output of the LZ77 algorithm is further compressed using *Huffman coding* [193]. This particular compression function allows us to control the compression rate, denoted by r from now on. Lower values of r means faster but lower compressions, while larger values of r indicate slower but higher compressions.

6.3.3.2 *Impact of the Compression Rate*

We start our analysis with a study on the impact of the compression rate r on the classification accuracy of a k-NN operating on NGCD values. In Fig. 6.5, we plot the classification accuracy across all datasets and for four compression rates, viz. $r \in \{1, 3, 6, 9\}$ (remember that lower compression rates correspond to faster but less effective compression, while higher rates leads to stronger but slower compressions). To highlight the general trend,

Table 6.7: Properties of the graph datasets. We show the number of graphs ($|\mathcal{G}|$), the number of classes ($|\Omega|$) and the average number of nodes and edges per graph ($\emptyset|V|$, $\emptyset|E|$).

Dataset	Graph Dataset Property			
	$\|\mathcal{G}\|$	$\|\Omega\|$	$\emptyset\|V\|$	$\emptyset\|E\|$
IMDB-BINARY	1,000	2	19.8	96.5
COLLAB	5,000	3	74.5	2,457.8
REDDIT-MULTI-5K	4,999	5	508.5	594.9
REDDIT-MULTI-12K	11,929	11	391.4	456.9

we additionally fit a regression line on the accuracies for each dataset.

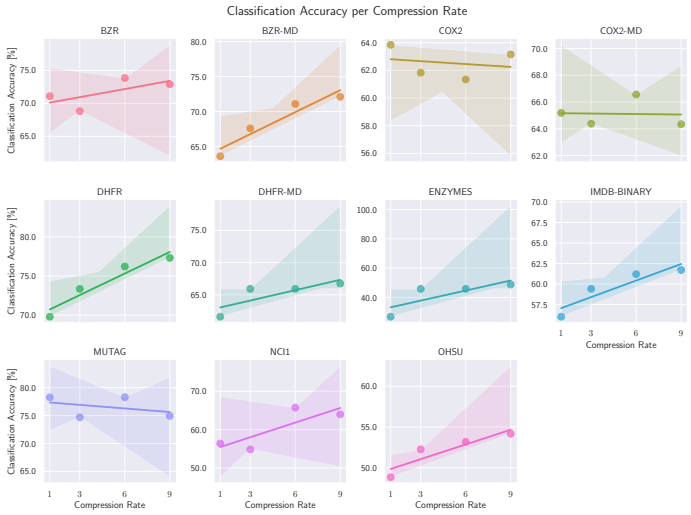


Fig. 6.5: Classification accuracies on the validation sets for each compression rate $r \in \{1, 3, 6, 9\}$.

There are three out of twelve datasets (viz. COX2, COX2-MD, and MUTAG) where the classification tends not to improve as the compression rate increases. That is, on these datasets, the regression line tends to decrease (or remains stable) with higher compression rates. In general, however, we observe that the classification accuracy tends to increase with higher compression rates (which makes sense as greater compression rates better

approximate the theoretical AID induced by the Kolmogorov complexity). For the BZR-MD and DHFR datasets, for instance, we observe improvements of about five percent points when the compression rate r is increased from 1 to 9.

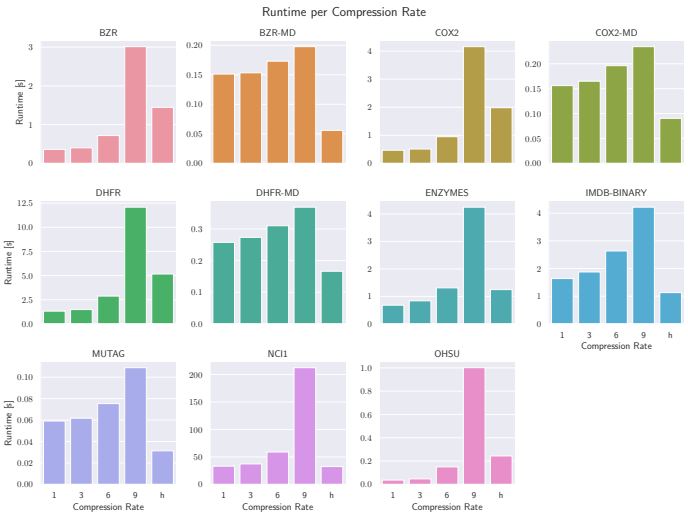


Fig. 6.6: Classification runtime across all datasets with different compression rates $r \in \{1, 3, 6, 9\}$. We also show the runtime with $r = 9$ in conjunction with our heuristic selection of training graphs (denoted by h in the figure).

However, the major drawback of larger compression rates r is the subsequent increase of the runtime (due to more complex computations). In Fig. 6.6, we plot the runtime of the graph classification framework for the four compression rates $r \in \{1, 3, 6, 9\}$. Additionally, we show the runtime of the framework that employs the proposed heuristic where the training set $\mathcal{G}_{tr}$ is reduced according to Alg. 5 (with a compression rate of $r = 9$)[4]. We observe a consistent pattern where the runtimes steadily increase with higher compression rates. For some datasets (such as BZR or COX2), we observe that the runtime for $r \in \{1, 3, 6\}$ remains relatively constant but increases sharply when the compression rate becomes $r = 9$. This behavior can be attributed to the relatively low edge density in these datasets, which

[4]In this analysis, the runtime for the node permutation is not taken into account, i.e., only the runtime for the classification is reported.

facilitate compression at lower compression rates.

We clearly observe the positive impact on the runtime when the proposed training set selection heuristic is applied [5]. That is, despite using a compression rate of $r = 9$, the runtime is significantly improved. In seven out of eleven datasets, the runtime is even lower than the runtime obtained with a compression rate of $r = 1$.

On the basis of these observations, we decide to use a compression rate of $r = 9$ across all datasets for the remaining parts of the present evaluations.

6.3.3.3 *Impact of the Node Permutation*

The objective of this subsection is to analyze the effects of different node permutations in our framework. To this end, we first compare the classification accuracy achieved with deterministic node permutations against random node permutations.

In Fig. 6.7, we show the classification accuracies obtained by our graph matching framework on the test sets with both random and deterministic node permutations. Across all datasets, we observe that results obtained with random node permutation show lower classification accuracies compared to their counterparts using deterministic node permutation. This highlights the crucial role of the node permutation step within our graph matching framework.

In Table 6.8, we present the runtimes in seconds of the five node permutation methods per dataset. Cuthill-Mckee is the fastest permutation method on all data sets (followed by Label Propagation and Louvain). The two methods Girvan Newman and SCD are in part drastically slower.

The runtime of the node permutation method that achieves the best classification accuracy (on the validation set) is highlighted for each dataset (e.g., the SCD node permutation method achieves the best classification accuracy for the COX2 dataset). When two or more node permutation methods achieve the best classification accuracy, two or more runtimes are highlighted per dataset.

We also calculate rank 1 to 5 for each node permutation method and each data set according to the classification accuracy and report the average rank in the bottom row of Table 6.8. We observe that both Girvan Newman and SCD achieve the lowest ranks, which suggests that these two node permutation methods produce the most meaningful orderings of the adja-

[5] Here, the runtime is reported with parameter ϵ being optimized to yield the best classification accuracy on the validation set.

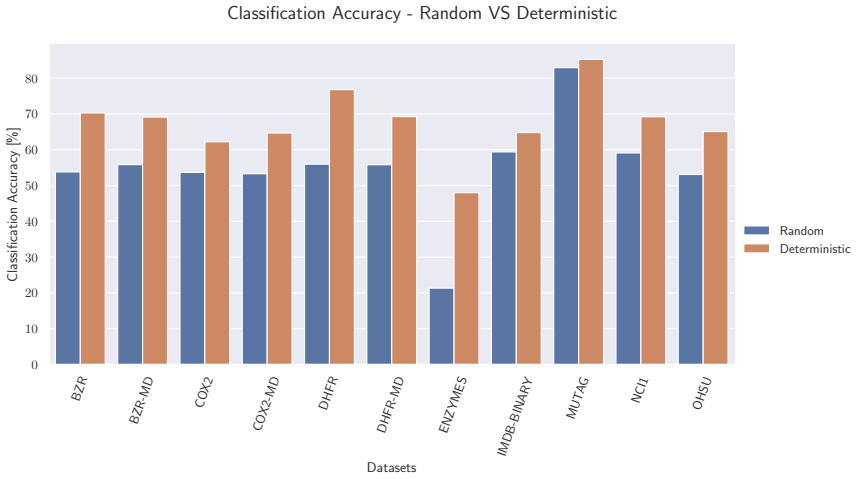


Fig. 6.7: Classification accuracy [%] obtained with random or deterministic node permutation across all datasets.

cency matrices. However, in our evaluations, we observe that Cuthill-Mckee performs poorly on the three specific MD datasets (BZR-MD, COX2-MD, and DHFR-MD). Actually, if those datasets were not taken into account in the ranking, Cuthill-Mckee would have the lowest average rank. In fact, we also find that this method is the only method to achieve the best result on three datasets (while Louvain is the only winner on two datasets and Girvan Newman, Label Propagation and SCD are the only winners on one of the datasets).

6.3.3.4 *NGCD vs. GED*

In the last comparison, we evaluate the novel graph matching approach using NGCD in a classification scenario and compare the results with the standard approach of GED. For the computation of GED, we use an optimized version of the *BP-GED* algorithm with cubic time complexity [11]. In Table 6.9, we present the classification accuracies of a k-NN classifier on all datasets obtained with NGCD and GED. We observe that our novel method outperforms the GED baseline in seven out of eleven cases. For instance, our approach outperforms the reference system on BZR-MD, COX2-MD, and DHFR-MD, by about 5 to 10 percent points. On the OHSU dataset, which is known to be a very challenging datasets [46], GED yields results

Table 6.8: Runtime in seconds [s] of the node permutation methods per dataset. The method(s) that achieve(s) the best classification result on the validation set is highlighted. The last row represents the average rank obtained on the validation set.

Dataset	Node Permutation Method: Cuthill-Mckee	Girvan Newman	Label Propagation	Louvain	SCD
BZR	0.25	6.09	**0.36**	0.66	7.87
BZR-MD	0.18	**1.47**	**0.31**	**0.57**	**11.07**
COX2	0.29	7.84	0.44	0.87	**10.60**
COX2-MD	0.19	**2.21**	**0.37**	**0.71**	**15.92**
DHFR	0.47	**14.42**	0.73	1.34	17.11
DHFR-MD	0.23	**2.35**	**0.40**	**0.80**	**16.97**
ENZYMES	**0.35**	20.38	0.57	0.96	11.45
IMDB-BINARY	**0.55**	45.08	0.72	1.27	18.83
MUTAG	0.11	0.83	0.13	**0.21**	1.93
NCI1	**2.48**	44.90	3.38	5.98	63.67
OHSU	0.07	83.88	0.21	**0.29**	4.47
Average Rank	2.9	2.7	3.3	3.4	2.7

slightly better than random decision. Our approach, however, achieves a notable increase of 15 percentage points (suggesting the efficacy of NGCD on this dataset).

We also compare the computation time of NGCD and GED. We report the total time for computing the complete distance matrix for all available graphs in seconds and the time per graph matching in milliseconds for both NGCD (including node permutation computation) and GED in Table 6.10.

We observe that the novel dissimilarity measure NGCD consistently shows faster computation times than GED. Specifically, the runtime is about four times faster for datasets such as BZR or OSHU, while it is up to about 12 to 14 times faster for datasets like DHFR-MD or IMDB-Binary.

Table 6.9: Mean classification accuracy [%] achieved using NGCD and GED. The best result, for each dataset, is highlighted in blue.

Dataset	NGCD	GED
BZR	70.28 ± 2.43	$\mathbf{74.17 \pm 1.44}$
BZR-MD	$\mathbf{69.12 \pm 2.11}$	64.62 ± 1.72
COX2	$\mathbf{62.21 \pm 2.17}$	60.46 ± 0.59
COX2-MD	$\mathbf{64.67 \pm 2.01}$	57.55 ± 0.86
DHFR	76.80 ± 1.21	$\mathbf{77.18 \pm 0.72}$
DHFR-MD	$\mathbf{69.26 \pm 2.15}$	55.49 ± 1.47
ENZYMES	$\mathbf{48.00 \pm 1.62}$	46.21 ± 1.20
IMDB-BINARY	64.80 ± 1.04	$\mathbf{69.59 \pm 1.01}$
MUTAG	$\mathbf{85.25 \pm 1.48}$	84.76 ± 1.66
NCI1	69.22 ± 1.16	$\mathbf{75.13 \pm 0.51}$
OHSU	$\mathbf{65.08 \pm 3.49}$	51.55 ± 4.96

Table 6.10: Runtime time in seconds [s] of the NGCD and GED for comparing all graphs of the dataset with each other and the time per comparison in miliseconds [ms].

Dataset	NGCD		GED		Speed-up Factor
	Total [s]	Time / Comp. [ms]	Total	Time / Comp. [ms]	
BZR	44.13	0.54	172.47	2.10	4
BZR-MD	3.67	0.08	44.57	0.95	12
COX2	71.86	0.66	288.69	2.65	4
COX2-MD	4.77	0.10	61.96	1.35	12
DHFR	195.85	0.69	807.38	2.85	4
DHFR-MD	6.26	0.08	88.31	1.14	14
ENZYMES	37.80	0.21	338.25	1.88	9
IMDB-BINARY	35.10	0.07	428.67	0.86	12
MUTAG	1.51	0.09	12.15	0.69	6
NCI1	2367.88	0.28	14,000.87	1.66	6
OHSU	9.29	2.98	32.52	10.42	3

6.4 Conclusion

In this chapter, we propose two novel methods for graph-based pattern recognition to achieve two distinct objectives in two parts.

In the first part of the chapter, we introduce a novel framework that uses recent GNN layers and adopt them for the specific task of graph reduction. Eventually, we classify the learned graph reductions with well-known pat-

tern recognition techniques based on GED. To the best of our knowledge, such an architecture has not been proposed before.

By means of an experimental evaluation on diverse datasets we empirically confirm that our graph reduction process is useful for downstream graph classification tasks. That is, we show that our framework maintains satisfactory classification accuracy when deleting 25%, and on some datasets even 50%, of the nodes. Simultaneously, we show that the runtime can – as expected – be substantially reduced by means of our novel reduction method. The conducted experiments also clearly reveal that the novel approach for graph reduction performs significantly better than randomized graph reductions. Last but not least, we observe that combining graph matching with our specific GNN for graph reduction achieves better results than an isolated GNN model (that employs the same architecture as we use for reduction).

In the second part of the present chapter, we propose a novel framework for graph matching, termed NGCD, that is based on a lossless compression algorithm. Roughly speaking, the proposed method works in two separate steps. In the first step, we apply a node permutation algorithm to fix the node ordering of the graphs. Second, we compute the normalized compression distance (NCD) on the adjacency matrices of the graphs and perform graph classification using the NCD as a distance metric.

We validate the benefits of our novel method with a thorough evaluation on eleven datasets. For instance, we research the impact of the compression rate on the classification accuracy and observe that the classification rate generally increases when the compression rate is increased. We also evaluate a novel heuristic to reduce the set of training graphs which in turn speeds up the complete classification framework. Furthermore, we compare the proposed framework with a standard graph matching framework, viz. GED. Overall, we demonstrate that NGCD outperforms the reference system GED on seven out of eleven datasets. Additionally, the runtime is consistently lower when using NGCD instead of GED for graph matching. Last but not least, our framework performs quite well in domains where standard models struggle, as evidenced by its performance on the challenging OHSU dataset.

Conclusion and Future Work

7

> La densité de l'Histoire ne détermine aucun de mes actes. Je suis mon propre fondement. Et c'est en dépassant la donnée historique, instrumentale, que j'introduis le cycle de ma liberté.
>
> *Peau noire, masques blancs* (1952), Frantz Fanon

A Graph is a data structure that consists of a set of nodes which are in turn potentially connected by edges. In contrast to other data representations (e.g., feature vectors), graphs have an inherent structure encoded in their representation. Due to this unique property, graphs can be used in many fields to represent complex systems such as molecules or social media networks. Structural pattern recognition is the field of research that develops and investigates algorithms that take graph data structures as their input.

Over the years, many structural pattern recognition algorithms have been developed and researched. These algorithms can broadly be categorized into three groups, namely, graph matching algorithms that find similar parts between graphs, graph kernels that implicitly embed graphs into a Hilbert space, and graph neural networks that compute a explicit embedding of the graphs. These methods are commonly used for graph classification or graph clustering.

A major challenge for any graph-based pattern recognition algorithm is, in general, the high computation cost, which hinders their application on large-scale problems or in real-time processing tasks. A common approach to address this issue is to use faster approximation algorithms for graph processing (e.g., approximate graph matching algorithms). While these al-

gorithms reduce accuracy to some extent, they can significantly improve the computation time and in some cases achieve polynomial-time complexity (rather than exponential-time complexity).

Another idea to counteract the high computational cost of graph matching, is offered by graph reduction methods. These methods pursue the strategy of data approximation instead of algorithm approximation. Graph reduction algorithms aim to reduce the size of the input graph while preserving its essential properties. However, determining the properties which are actually essential and should thus be preserved during the reduction process is not straightforward and often context-dependant. This means that the effectiveness of a reduction method depends on the problem to be solved and therefore it might not be possible to define a generic method for graph reduction.

The main objective of the present thesis is to thoroughly research graph reduction in the context of graph matching. In particular, the goal is to develop graph reduction algorithms that generate reduced graphs which are able to maintain the classification accuracy as high as possible when used in combination with different graph classification algorithms. The present thesis introduces and researches the following four graph reduction approaches.

- In a first approach (presented in Chapter 4), two centrality measures are used to rank the nodes of a graph from the least to the most important. Based on these centrality scores, the least important nodes are iteratively discarded, resulting in differently reduced graphs. We evaluate the performance of those reduced graphs in a graph classification task on multiple datasets. The results show that the use of centrality reduced graphs can maintain satisfactory classification accuracies in general and simultaneously reduce the computation time.

  We extend the use of those centrality reduced graphs in two novel frameworks with the goal to improve the classification accuracy. In the first case, we propose a two-step graph classification framework that first selects graphs in the reduced graph space and then, if the performance is not convincing enough, classifies them in the original graph space. In the second case, the centrality reduced graphs obtained at each reduction level are combined in a multiple classifier system which is weighted according to the importance of each level. Both systems are convincing in terms of the resulting

classification accuracy (even if an increase in computation time must generally be accepted).

- In a second approach (presented in Chapter 5), spectral graph clustering is used to find communities in the graphs. The found clusters are then aggregated into supernodes to reduce the size of the graph. The classification experiments conducted show that even strongly reduced graphs maintain satisfactory classification accuracies across the four graph classification algorithms tested. Moreover, the computation time can be accelerated up to two orders of magnitude on datasets with large graphs. In a detailed study of the dissimilarities, we observe that the dissimilarity shrinks as the reduction level is reduced, yet, the distances on the reduced graphs remain coherent with the original distances.
- In a third approach (presented in the first part of Chapter 6), we propose to use a modified graph neural network to directly learn the importance of a node from the data. Through several experimental evaluations, we show that this approach maintains satisfactory classification accuracies while significantly speeding up the computation time (compared to processing on the original graphs).
- The fourth approach (presented in the second part of Chapter 6) is based on a compression-based distance measure. This method involves the computation of a deterministic node ordering and then computing the normalized compression distance (NCD) on the resulting adjacency matrices. Evaluations of the proposed framework show that it outperforms standard graph matching algorithms both in terms of classification accuracy and computation time.

Although we have thoroughly researched the four different graph reduction methods in the present thesis, we see at least three different future research activities that could be rewarding.

- A first line of research might concern the method presented in Chapter 5, where the process of aggregating nodes of one cluster into supernodes consists of summing up the node features. As a first direction for future research, more advanced methods for merging nodes into supernodes could be explored. Specialized graph neural networks may be used in this process, for instance, with the goal of identifying optimal feature vectors for the super-nodes.
- A second line of research could involve the use of graph neural

network architectures to learn graph coarsening. This approach would aim to optimize coarsening specifically to generate reduced graphs that perform particularly well for the graph classifier being used. To this end, the classification of the reduced graphs obtained with the subsequent graph classifier must be incorporated into the loss function used to train the coarsening graph neural networks, thus creating an end-to-end pipeline.

- A third line of research could involve the development of a function that assess how well the dissimilarity in the reduced graph space evolves compared to the original graph space. An idea could be to compute the topological persistence diagrams of both the original and reduced graph space and then perform a comparison on these persistence diagrams.

Appendix A

Appendix Chapter 3

A.1 T-SNE Visualization of Labeled Datasets

Fig. A.1: Visualization of the T-SNE embedding of labeled datasets

Appendix B

Appendix Chapter 4

B.1 Visualization of Reduced Graphs by Means of Centrality Measures

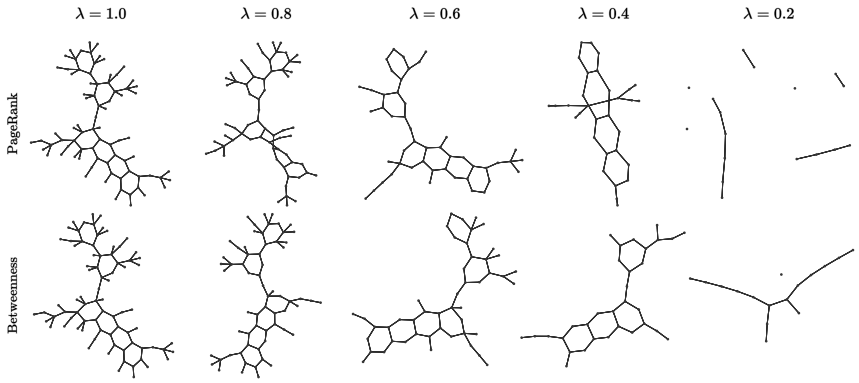


Fig. B.1: Reduced graph from Mutagenicity

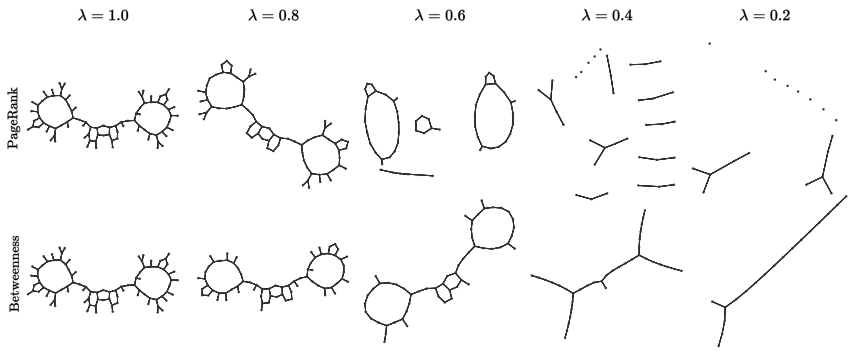


Fig. B.2: Reduced graph from NCI1

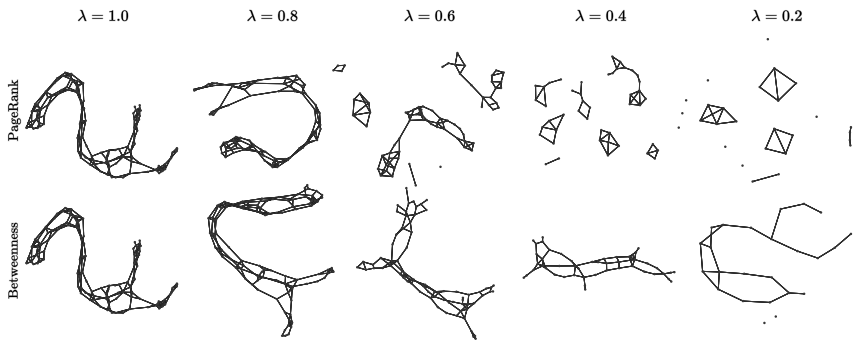


Fig. B.3: Reduced graph from Proteins

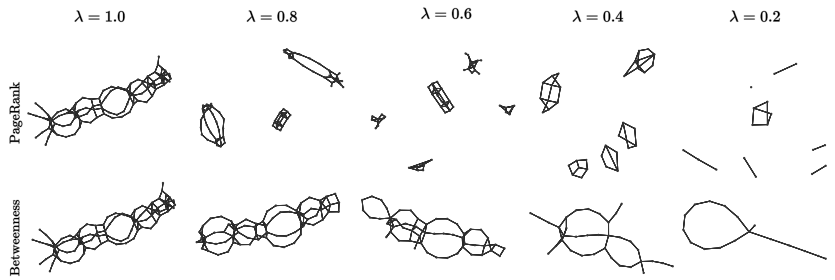


Fig. B.4: Reduced graph from Enzymes

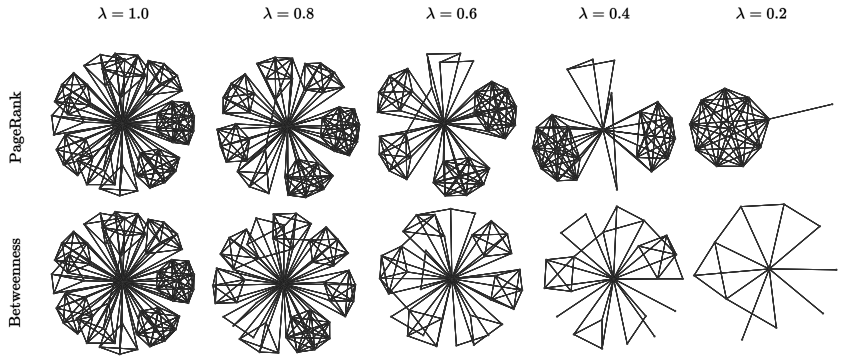


Fig. B.5: Reduced graph from IMDB-BINARY

B.2 Visualization of the Pairwise GED between the Original Graphs and their Reduced Counterpart

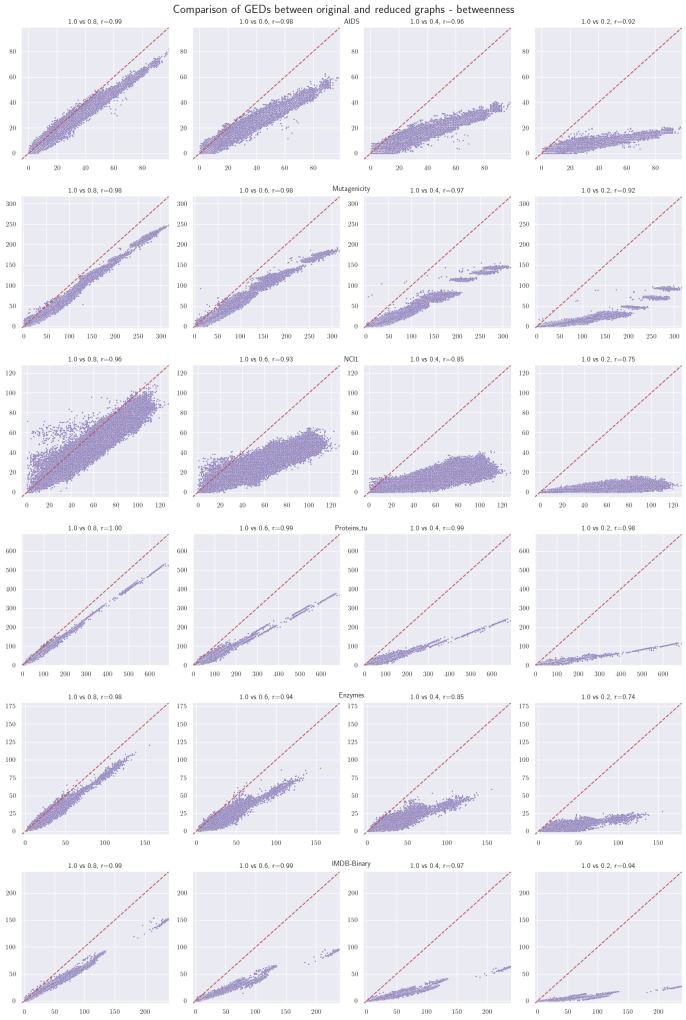
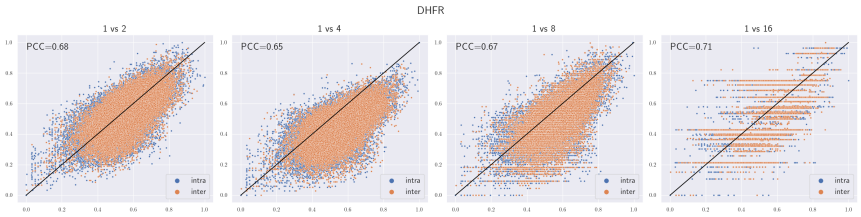


Fig. B.6: Pairwise distances between graphs in the original graph space and their resulting counterpart in the reduced graph domain using Betweenness on all datasets.

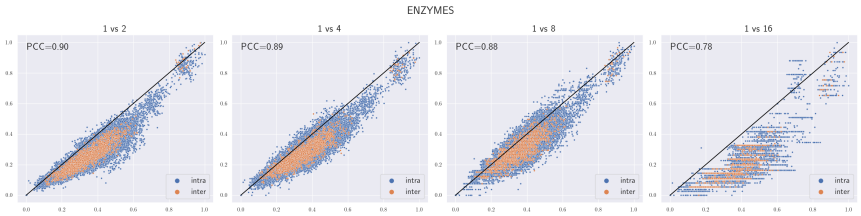
Appendix C

Appendix Chapter 5

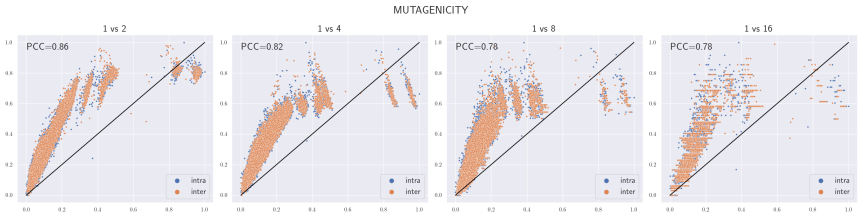
C.1 Visualization of the Pairwise GED between the Original Graphs and their Reduced Counterpart



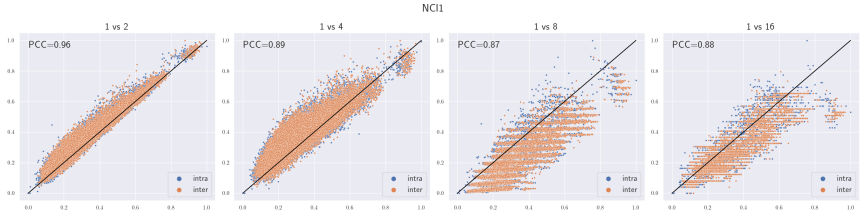
(a) Pairwise distances DHFR



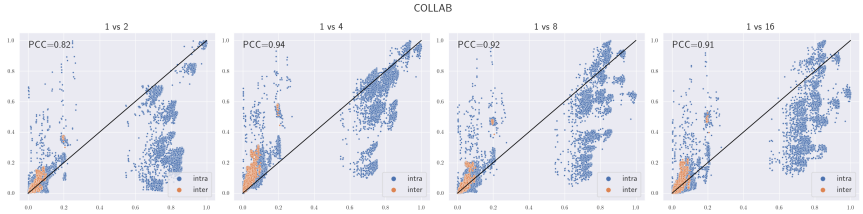
(b) Pairwise distances ENZYMES



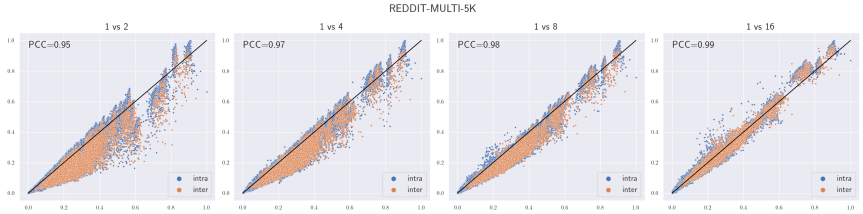
(c) Pairwise distances Mutagenicity



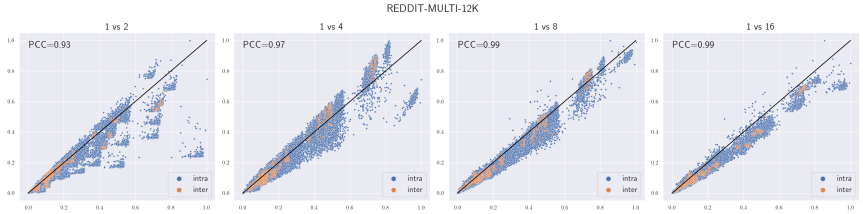
(d) Pairwise distances NCI1



(e) Pairwise distances COLLAB



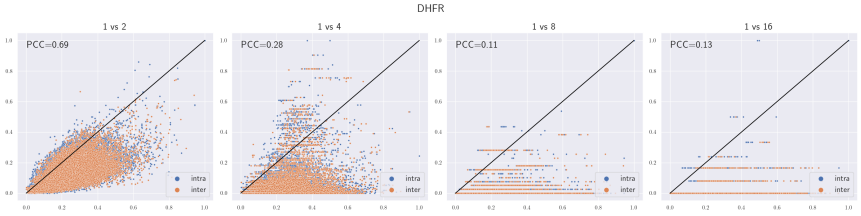
(f) Pairwise distances REDDIT-MULTI-5K



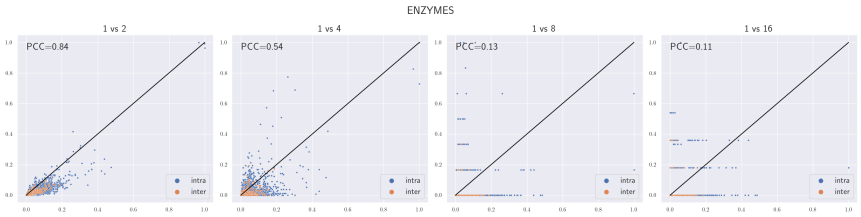
(g) Pairwise distances REDDIT-MULTI-12K

Fig. C.1: Comparison of pairwise similarities/dissimilarities between graphs in the original and the reduced graph domains for both DHFR, ENZYMES, Mutagenicity, NCI1, COLLAB, REDDIT-5K, REDDIT-12K and NCI109 datasets using Graph Edit Distance (GED).

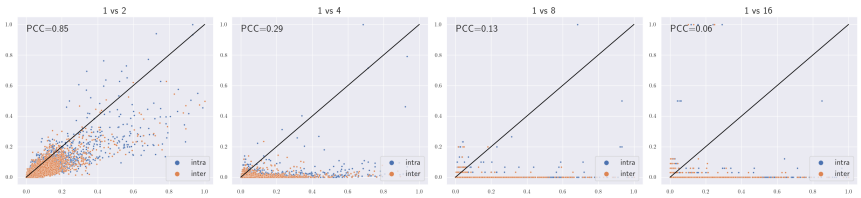
C.2 Visualization of the Pairwise SP between the Original Graphs and their Reduced Counterpart



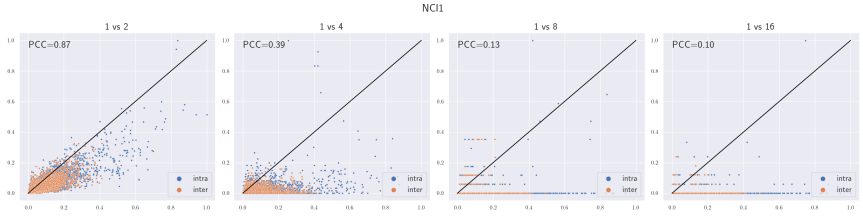
(a) Pairwise similarities DHFR



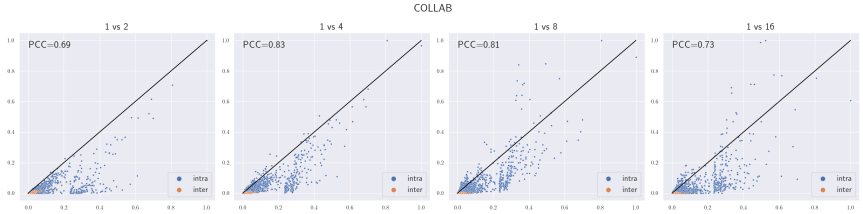
(b) Pairwise similarities ENZYMES



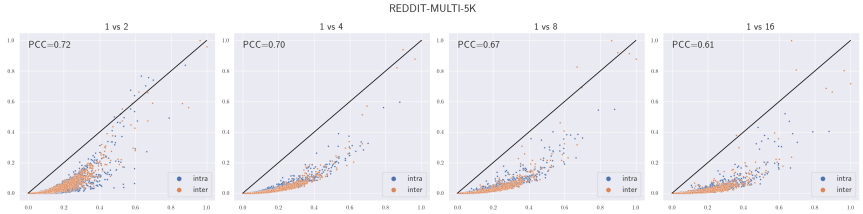
(c) Pairwise similarities Mutagenicity



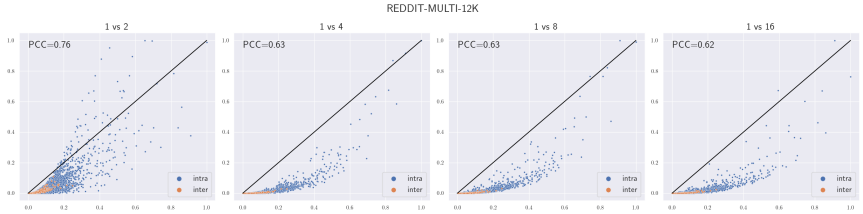
(d) Pairwise similarities NCI1



(e) Pairwise similarities COLLAB



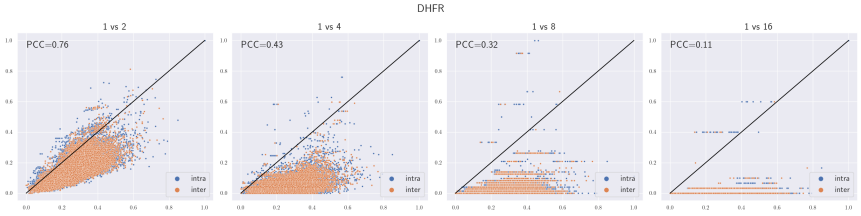
(f) Pairwise similarities REDDIT-MULTI-5K



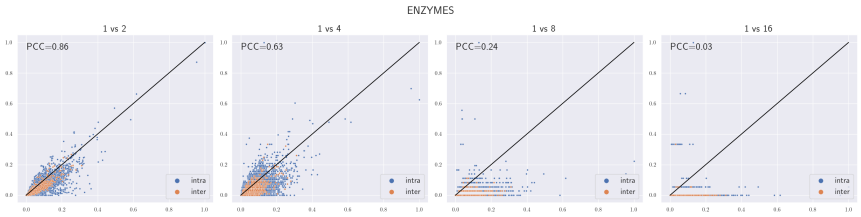
(g) Pairwise similarities REDDIT-MULTI-12K

Fig. C.2: Comparison of pairwise similarities/dissimilarities between graphs in the original and the reduced graph domains for both DHFR, ENZYMES, Mutagenicity, NCI1, COLLAB, REDDIT-5K, REDDIT-12K and NCI109 datasets using the ShortestPath graph kernel (SP)

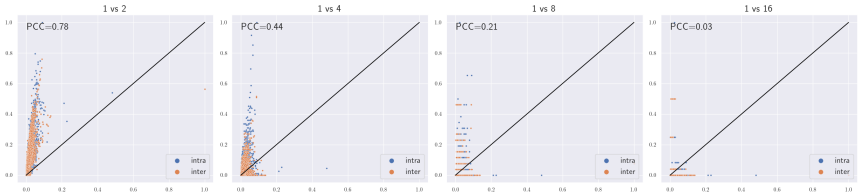
C.3 Visualization of the Pairwise WL between the Original Graphs and their Reduced Counterpart



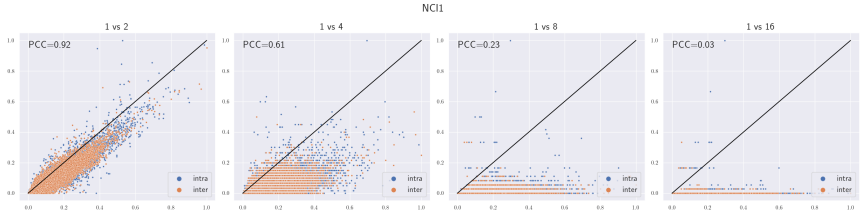
(a) Pairwise similarities DHFR



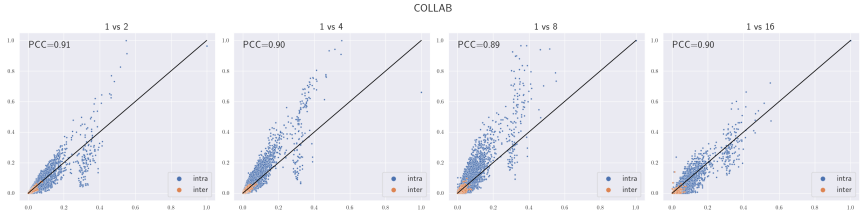
(b) Pairwise similarities ENZYMES



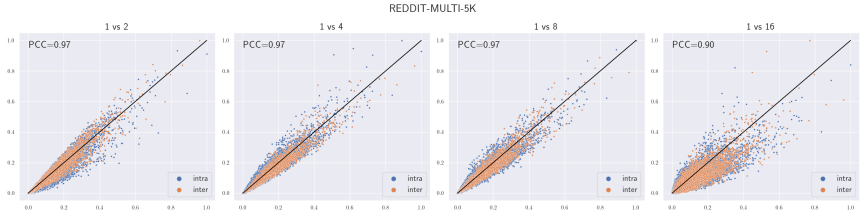
(c) Pairwise similarities Mutagenicity



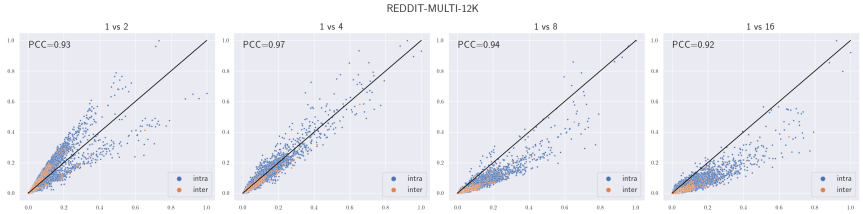
(d) Pairwise similarities NCI1



(e) Pairwise similarities COLLAB



(f) Pairwise similarities REDDIT-MULTI-5K



(g) Pairwise similarities REDDIT-MULTI-12K

Fig. C.3: Comparison of pairwise similarities/dissimilarities between graphs in the original and the reduced graph domains for both DHFR, ENZYMES, Mutagenicity, NCI1, COLLAB, REDDIT-5K, REDDIT-12K and NCI109 datasets using the 4-Weisfeiler-Lehman graph kernel (WL).

Appendix D

Appendix Chapter 6

D.1 Analysis of the Connected Components of Graphs Reduced with GReNN

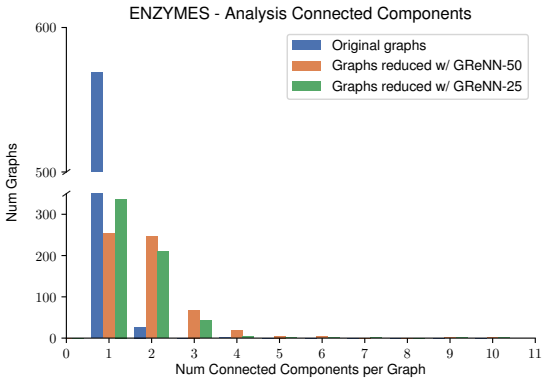


Fig. D.1: Histogram that shows the number of graphs (on the y-axis) that have a given number of connected components per graph (on the x-axis) on the ENZYMES dataset.

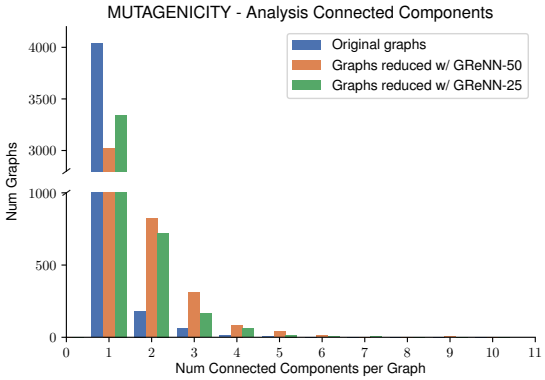


Fig. D.2: Histogram that shows the number of graphs (on the y-axis) that have a given number of connected components per graph (on the x-axis) on the MUTAGENICITY dataset.

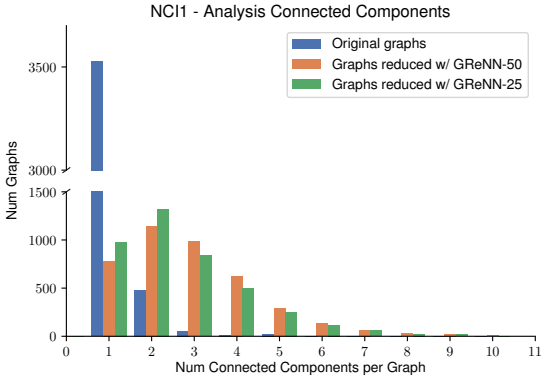


Fig. D.3: Histogram that shows the number of graphs (on the y-axis) that have a given number of connected components per graph (on the x-axis) on the NCI1 dataset.

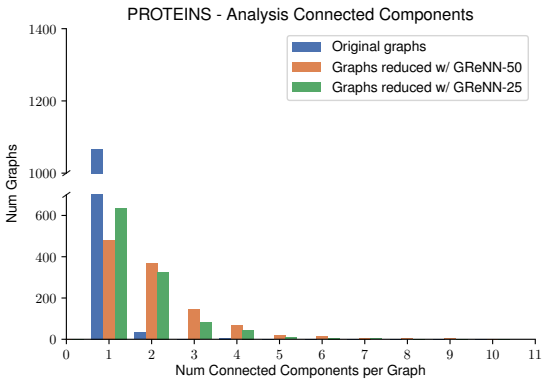
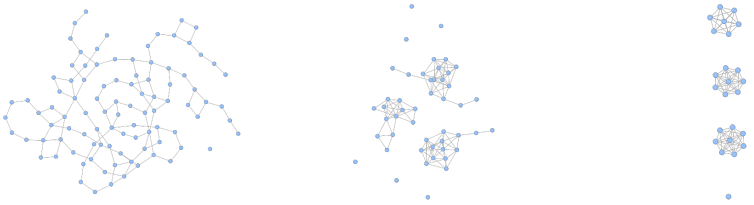


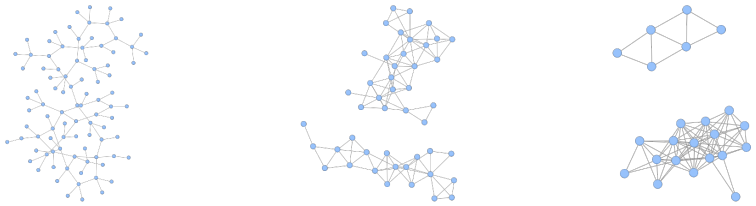
Fig. D.4: Histogram that shows the number of graphs (on the y-axis) that have a given number of connected components per graph (on the x-axis) on the PROTEINS dataset.

D.2 Example of Reduced Graphs using GReNN



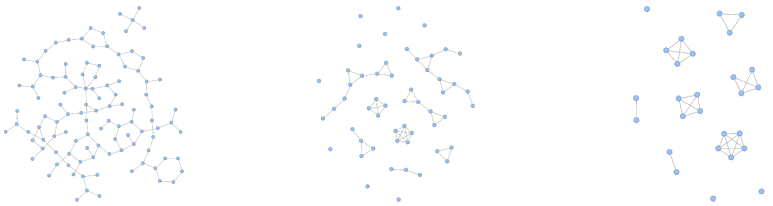
(a) Original Graph (b) GReNN-50 (c) GReNN-25

Fig. D.5: Example of an original graph and the corresponding reduced graphs via GReNN-50 and GReNN-25 for the ENZYMES dataset.



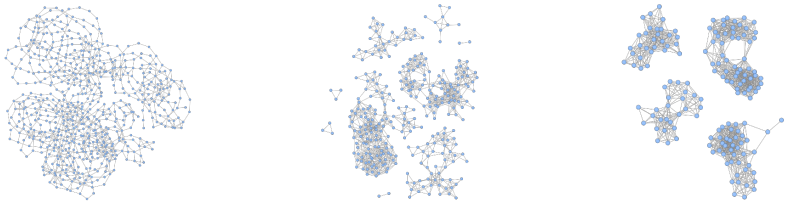
(a) Original Graph (b) GReNN-50 (c) GReNN-25

Fig. D.6: Example of an original graph and the corresponding reduced graphs via GReNN-50 and GReNN-25 for the MUTAGENCITY dataset.



(a) Original Graph (b) GReNN-50 (c) GReNN-25

Fig. D.7: Example of an original graph and the corresponding reduced graphs via GReNN-50 and GReNN-25 for the NCI1 dataset.



(a) Original Graph (b) GReNN-50 (c) GReNN-25

Fig. D.8: Example of an original graph and the corresponding reduced graphs via GReNN-50 and GReNN-25 for the PROTEINS dataset.

Bibliography

[1] Joel Serey, Miguel D. Alfaro, Guillermo Fuertes, Manuel Vargas, Claudia A. Durán, Rodrigo Ternero, Ricardo Rivera, and Jorge Sabattin. Pattern Recognition and Deep Learning Technologies, Enablers of Industry 4.0, and Their Role in Engineering Research. *Symmetry*, 15(2):535, 2023.

[2] John S. Mattick, Marie A. Dziadek, Bronwyn N. Terrill, Warren Kaplan, Allan D. Spigelman, Frank G. Bowling, and Marcel E. Dinger. The impact of genomics on the future of medicine and health. *Medical Journal of Australia*, 201(1), July 2014.

[3] Gilbert Brunet, David B. Parsons, Dimitar Ivanov, Boram Lee, Peter Bauer, Natacha B. Bernier, Veronique Bouchet, Andy Brown, Antonio Busalacchi, Georgina Campbell Flatter, Rei Goffer, Paul Davies, Beth Ebert, Karl Gutbrod, Songyou Hong, P. K. Kenabatho, Hans-Joachim Koppert, David Lesolle, Amanda H. Lynch, Jean-François Mahfouf, Laban Ogallo, Tim Palmer, Kevin Petty, Dennis Schulze, Theodore G. Shepherd, Thomas F. Stocker, Alan Thorpe, and Rucong Yu. Advancing Weather and Climate Forecasting for Our Changing World. *Bulletin of the American Meteorological Society*, 104(4):E909–E927, May 2023. Publisher: American Meteorological Society Section: Bulletin of the American Meteorological Society.

[4] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty Years Of Graph Matching In Pattern Recognition. *IJPRAI*, 18:265–298, May 2004.

[5] 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.

[6] Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *Appl. Netw. Sci.*, 5(1):6, 2020.

[7] Karsten M. Borgwardt, M. Elisabetta Ghisu, Felipe Llinares-López, Leslie O'Bray, and Bastian Rieck. Graph Kernels: State-of-the-Art and Future Challenges. *Found. Trends Mach. Learn.*, 13(5-6), 2020.

[8] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The Graph Neural Network Model. *IEEE Transac-*

tions on Neural Networks, 20(1):61–80, January 2009. Conference Name: IEEE Transactions on Neural Networks.

[9] 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.

[10] Martin Grohe and Pascal Schweitzer. The graph isomorphism problem. *Commun. ACM*, 63(11):128–134, 2020.

[11] Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vis. Comput.*, 27(7):950–959, 2009.

[12] T. Caelli and S. Kosinov. An eigenspace projection clustering method for inexact graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(4):515–519, April 2004. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[13] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[14] Francisco Escolano, Boyan Bonev, and Miguel A. Lozano. Information-Geometric Graph Indexing from Bags of Partial Node Coverages. In Xiaoyi Jiang, Miquel Ferrer, and Andrea Torsello, editors, *Graph-Based Representations in Pattern Recognition*, Lecture Notes in Computer Science, pages 52–61, Berlin, Heidelberg, 2011. Springer.

[15] Julien Lerouge, Zeina Abu-Aisheh, Romain Raveaux, Pierre Héroux, and Sébastien Adam. New binary linear programming formulation to compute the graph edit distance. *Pattern Recognit.*, 72:254–265, 2017.

[16] Mikhail Zaslavskiy, Francis R. Bach, and Jean-Philippe Vert. Many-to-Many Graph Matching: A Continuous Relaxation Approach. In José L. Balcázar, Francesco Bonchi, Aristides Gionis, and Michèle Sebag, editors, *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part III*, volume 6323 of *Lecture Notes in Computer Science*, pages 515–530. Springer, 2010.

[17] Vince Lyzinski, Donniell E. Fishkind, Marcelo Fiori, Joshua T. Vogelstein, Carey E. Priebe, and Guillermo Sapiro. Graph Matching: Relax at Your Own Risk. *IEEE Trans. Pattern Anal. Mach. Intell.*, 38(1):60–73, 2016.

[18] 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.

[19] Karsten M. Borgwardt and Hans-Peter Kriegel. 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.

[20] 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.

[21] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *CoRR*, abs/1609.02907, 2016. arXiv: 1609.02907.

[22] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *CoRR*, abs/1710.10903, 2017. arXiv: 1710.10903.

[23] Luana Ruiz, Fernando Gama, and Alejandro Ribeiro. Gated Graph Recurrent Neural Networks. *IEEE Trans. Signal Process.*, 68:6303–6318, 2020.

[24] 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.

[25] 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.

[26] Maryam Khalid and Akane Sano. Exploiting social graph networks for emotion prediction. *Scientific Reports*, 13(1):6069, 2023.

[27] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

[28] 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.

[29] S. V. N. Vishwanathan, Karsten M. Borgwardt, and Nicol N. Schraudolph. Fast Computation of Graph Kernels. In Bernhard Schölkopf, John C. Platt, and Thomas Hofmann, editors, *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 1449–1456. MIT Press, 2006.

[30] U. Kang, Hanghang Tong, and Jimeng Sun. Fast Random Walk Graph Kernel. In *Proceedings of the Twelfth SIAM International Conference on Data Mining, Anaheim, California, USA, April 26-28, 2012*, pages 828–838. SIAM / Omnipress, 2012.

[31] Xiaoyang Chen, Hongwei Huo, Jun Huan, and Jeffrey Scott Vitter. Fast Computation of Graph Edit Distance. *CoRR*, abs/1709.10305, 2017. arXiv: 1709.10305.

[32] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. Graph Summarization Methods and Applications: A Survey. *ACM Comput. Surv.*, 51(3):62:1–62:34, 2018.

[33] Jie Chen, Yousef Saad, and Zechen Zhang. Graph coarsening: From scientific computing to machine learning. *CoRR*, abs/2106.11863, 2021. arXiv:

2106.11863.

[34] Mohammad Hashemi, Shengbo Gong, Juntong Ni, Wenqi Fan, B. Aditya Prakash, and Wei Jin. A Comprehensive Survey on Graph Reduction: Sparsification, Coarsening, and Condensation. *CoRR*, abs/2402.03358, 2024. arXiv: 2402.03358.

[35] Deepayan Chakrabarti, Spiros Papadimitriou, Dharmendra Modha, and Christos Faloutsos. Fully Automatic Cross-Associations. *KDD-2004 - Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2004.

[36] Matthijs van Leeuwen, Jilles Vreeken, and Arno Siebes. Compression Picks Item Sets That Matter. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Knowledge Discovery in Databases: PKDD 2006, 10th European Conference on Principles and Practice of Knowledge Discovery in Databases, Berlin, Germany, September 18-22, 2006, Proceedings*, volume 4213 of *Lecture Notes in Computer Science*, pages 585–592. Springer, 2006.

[37] Cody Dunne and Ben Shneiderman. Motif simplification: improving network visualization readability with fan, connector, and clique glyphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 3247–3256, New York, NY, USA, April 2013. Association for Computing Machinery.

[38] Florian Dörfler and Francesco Bullo. Kron Reduction of Graphs With Applications to Electrical Networks. *IEEE Trans. Circuits Syst. I Regul. Pap.*, 60-I(1):150–163, 2013.

[39] Luc Brun, Pasquale Foggia, and Mario Vento. Trends in graph-based representations for Pattern Recognition. *Pattern Recognit. Lett.*, 134:3–9, 2020.

[40] Seyedeh Fatemeh Mousavi, Mehran Safayani, Abdolreza Mirzaei, and Hoda Bahonar. Hierarchical graph embedding in vector space by graph pyramid. *Pattern Recognit.*, 61:245–254, 2017.

[41] Anjan Dutta. Hierarchical stochastic graphlet embedding for graph-based pattern recognition. *Neural Computing and Applications*, page 18, 2020.

[42] Anthony Gillioz and Kaspar Riesen. Speeding up Graph Matching by Means of Systematic Graph Reductions Using Centrality Measures. In *2022 12th International Conference on Pattern Recognition Systems (ICPRS)*, pages 1–7, 2022.

[43] Anthony Gillioz and Kaspar Riesen. Two-Step Graph Classification on the Basis of Hierarchical Graphs. In Maria De Marsico, Gabriella Sanniti di Baja, and Ana L. N. Fred, editors, *Proceedings of the 12th International Conference on Pattern Recognition Applications and Methods, ICPRAM 2023, Lisbon, Portugal, February 22-24, 2023*, pages 296–303. SCITEPRESS, 2023.

[44] Anthony Gillioz and Kaspar Riesen. Improving Graph Classification by Means of Linear Combinations of Reduced Graphs. In Maria De Marsico, Gabriella Sanniti di Baja, and Ana L. N. Fred, editors, *Proceedings of the 11th International Conference on Pattern Recognition Applications and Methods, ICPRAM 2022, Online Streaming, February 3-5, 2022*, pages 17–

23. SCITEPRESS, 2022.

[45] Anthony Gillioz and Kaspar Riesen. Building Multiple Classifier Systems Using Linear Combinations of Reduced Graphs. *SN Comput. Sci.*, 4(6):743, 2023.

[46] Anthony Gillioz and Kaspar Riesen. Graph-Based vs. Vector-Based Classification: A Fair Comparison. In Mario Vento, Pasquale Foggia, Donatello Conte, and Vincenzo Carletti, editors, *Graph-Based Representations in Pattern Recognition - 13th IAPR-TC-15 International Workshop, GbRPR 2023, Vietri sul Mare, Italy, September 6-8, 2023, Proceedings*, volume 14121 of *Lecture Notes in Computer Science*, pages 25–34. Springer, 2023.

[47] Anthony Gillioz and Kaspar Riesen. Graph Reduction Neural Networks for Structural Pattern Recognition. In Adam Krzyzak, Ching Y. Suen, Andrea Torsello, and Nicola Nobile, editors, *Structural, Syntactic, and Statistical Pattern Recognition - Joint IAPR International Workshops, S+SSPR 2022, Montreal, QC, Canada, August 26-27, 2022, Proceedings*, volume 13813 of *Lecture Notes in Computer Science*, pages 64–73. Springer, 2022.

[48] Anthony Gillioz and Kaspar Riesen. Graph Classification with NCD. *Pattern Recognition Letters*, 2024. Currently under review for possible publication.

[49] Kaspar Riesen. *Structural Pattern Recognition with Graph Edit Distance - Approximation Algorithms and Applications*. Advances in Computer Vision and Pattern Recognition. Springer, 2015.

[50] Arthur B. Yeh. A Modern Introduction to Probability and Statistics. *Technometrics*, 49(3):359, 2007.

[51] Kevin P. Murphy. *Machine learning - a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, 2012.

[52] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.

[53] Boyang Liu, Pang-Ning Tan, and Jiayu Zhou. Unsupervised Anomaly Detection by Robust Density Estimation. 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 4101–4108. AAAI Press, 2022.

[54] Michel Verleysen and John Aldo Lee. Nonlinear Dimensionality Reduction for Visualization. In Minho Lee, Akira Hirose, Zeng-Guang Hou, and Rhee Man Kil, editors, *Neural Information Processing - 20th International Conference, ICONIP 2013, Daegu, Korea, November 3-7, 2013. Proceedings, Part I*, volume 8226 of *Lecture Notes in Computer Science*, pages 617–622. Springer, 2013.

[55] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data mining: practical machine learning tools and techniques, 3rd Edition*. Morgan Kaufmann, Elsevier, 2011.

[56] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. Data Clustering: A Review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[57] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, pages 226–231. AAAI Press, 1996.

[58] Xiangli Yang, Zixing Song, Irwin King, and Zenglin Xu. A Survey on Deep Semi-Supervised Learning. *IEEE Trans. Knowl. Data Eng.*, 35(9):8934–8954, 2023.

[59] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. *Workshop on challenges in representation learning, ICML*, 3(2):896, 2013.

[60] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[61] Beakcheol Jang, Myeonghwi Kim, Gaspard Harerimana, and Jong Wook Kim. Q-Learning Algorithms: A Comprehensive Classification and Applications. *IEEE Access*, 7:133653–133667, 2019.

[62] Dimitri P. Bertsekas. Value and Policy Iteration in Optimal Control and Adaptive Dynamic Programming. *CoRR*, abs/1507.01026, 2015. arXiv: 1507.01026.

[63] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A Brief Survey of Deep Reinforcement Learning. *IEEE Signal Processing Magazine*, 34(6):26–38, November 2017.

[64] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Comput. Electr. Eng.*, 40(1):16–28, 2014.

[65] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios D. Doulamis, and Eftychios Protopapadakis. Deep Learning for Computer Vision: A Brief Review. *Comput. Intell. Neurosci.*, 2018:7068349:1–7068349:13, 2018.

[66] 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.

[67] Saeed Rahmani, Asiye Baghbani, Nizar Bouguila, and Zachary Patterson. Graph Neural Networks for Intelligent Transportation Systems: A Survey. *IEEE Trans. Intell. Transp. Syst.*, 24(8):8846–8885, 2023.

[68] Kaige Yang and Laura Toni. Graph-Based Recommendation System. In *2018 IEEE Global Conference on Signal and Information Processing, GlobalSIP 2018, Anaheim, CA, USA, November 26-29, 2018*, pages 798–802. IEEE, 2018.

[69] Olumide Owolabi. An efficient graph approach to matching chemical structures. *J. Chem. Inf. Comput. Sci.*, 28(4):221–226, 1988.

[70] Ingo Wegener. *Complexity theory - exploring the limits of efficient algorithms*. Springer, 2005.

[71] Mario Vento. A long trip in the charming world of graphs for Pattern Recognition. *Pattern Recognit.*, 48(2):291–301, 2015.

[72] Baida Zhang, Yuhua Tang, Junjie Wu, and Shuai Xu. Ld: a polynomial time algorithm for tree isomorphism. *Procedia Engineering*, 15:2015–2020, 2011. Publisher: Elsevier.

[73] Eugene M. Luks. Isomorphism of Graphs of Bounded Valence can be Tested in Polynomial Time. *J. Comput. Syst. Sci.*, 25(1):42–65, 1982.

[74] Xiaoyi Jiang and Horst Bunke. Optimal quadratic-time isomorphism of ordered graphs. *Pattern Recognit.*, 32(7):1273–1283, 1999.

[75] 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.

[76] Charles J. Colbourn. On testing isomorphism of permutation graphs. *Networks*, 11(1):13–21, 1981.

[77] 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.

[78] Peter J. Dickinson, Horst Bunke, Arek Dadej, and Miro Kraetzl. Matching graphs with unique node labels. *Pattern Anal. Appl.*, 7(3):243–254, 2004.

[79] Péter Englert and Péter Kovács. Efficient Heuristics for Maximum Common Substructure Search. *J. Chem. Inf. Model.*, 55(5):941–955, 2015.

[80] Kaspar Riesen, Xiaoyi Jiang, and Horst Bunke. Exact and Inexact Graph Matching: Methodology and Applications. In Charu C. Aggarwal and Haixun Wang, editors, *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*, pages 217–247. Springer, 2010.

[81] Junchi Yan, Xu-Cheng Yin, Weiyao Lin, Cheng Deng, Hongyuan Zha, and Xiaokang Yang. A Short Survey of Recent Advances in Graph Matching. In John R. Kender, John R. Smith, Jiebo Luo, Susanne Boll, and Winston H. Hsu, editors, *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval, ICMR 2016, New York, New York, USA, June 6-9, 2016*, pages 167–174. ACM, 2016.

[82] Terry Caelli and Serhiy Kosinov. Inexact Graph Matching Using Eigen-Subspace Projection Clustering. *Int. J. Pattern Recognit. Artif. Intell.*, 18(3):329–354, 2004.

[83] Zhou Fan, Cheng Mao, Yihong Wu, and Jiaming Xu. Spectral Graph Matching and Regularized Quadratic Relaxations: Algorithm and Theory. 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 2985–2995. PMLR, 2020.

[84] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. arXiv: 1609.04747.

[85] Osman Güler, Dick den Hertog, Cornelis Roos, Tamás Terlaky, and Takashi Tsuchiya. Degeneracy in interior point methods for linear programming: a survey. *Ann. Oper. Res.*, 46-47(1):107–138, 1993.

[86] Horst Bunke and Gudrun Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245–253, May 1983.

[87] 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.

[88] Kaspar Riesen and Horst Bunke. *Graph Classification and Clustering Based on Vector Space Embedding*. World Scientific Publishing Co., Inc., USA, 2010.

[89] Xavier Cortés and Francesc Serratosa. Learning graph-matching edit-costs based on the optimality of the oracle's node correspondences. *Pattern Recognition Letters*, 56:22–29, April 2015.

[90] 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.

[91] 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.

[92] Carlos Garcia-Hernandez, Alberto Fernández, and Francesc Serratosa. Ligand-Based Virtual Screening Using Graph Edit Distance as Molecular Similarity Measure. *J. Chem. Inf. Model.*, 59(4):1410–1421, 2019.

[93] Francesc Serratosa. Fast computation of Bipartite graph matching. *Pattern Recognit. Lett.*, 45:244–250, 2014.

[94] Mostafa Darwiche, Donatello Conte, Romain Raveaux, and Vincent T'kindt. A local branching heuristic for solving a Graph Edit Distance Problem. *Comput. Oper. Res.*, 106:225–235, 2019.

[95] 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.

[96] Tamás Horváth, Thomas Gärtner, and Stefan Wrobel. Cyclic pattern kernels for predictive graph mining. In Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel, editors, *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, pages 158–167. ACM, 2004.

[97] 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*, pages 1–20. icml.cc / Omnipress, 2012.

[98] Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. A Training Algorithm for Optimal Margin Classifiers. In David Haussler, editor, *Proceedings of the Fifth Annual ACM Conference on Computational Learning Theory, COLT 1992, Pittsburgh, PA, USA, July 27-29, 1992*, pages 144–152. ACM, 1992.

[99] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, 1962.

[100] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv:1609.02907 [cs, stat]*, February 2017. arXiv: 1609.02907.

[101] 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.

[102] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An End-to-End Deep Learning Architecture for Graph Classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018. Number: 1.

[103] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph WaveNet for Deep Spatial-Temporal Graph Modeling. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 1907–1913. ijcai.org, 2019.

[104] Vijay Prakash Dwivedi and Xavier Bresson. A Generalization of Transformer Networks to Graphs. *CoRR*, abs/2012.09699, 2020. arXiv: 2012.09699.

[105] Pau Riba, Andreas Fischer, Josep Lladós, and Alicia Fornés. Learning graph edit distance by graph neural networks. *Pattern Recognit.*, 120:108132, 2021.

[106] Daniel A. Spielman and Nikhil Srivastava. Graph Sparsification by Effective Resistances. *SIAM J. Comput.*, 40(6):1913–1926, 2011.

[107] Joshua D. Batson, Daniel A. Spielman, Nikhil Srivastava, and Shang-Hua Teng. Spectral sparsification of graphs: theory and algorithms. *Commun. ACM*, 56(8):87–94, 2013.

[108] Alfred V. Aho, M. R. Garey, and Jeffrey D. Ullman. The Transitive Reduction of a Directed Graph. *SIAM J. Comput.*, 1(2):131–137, 1972.

[109] Zhuo Feng. Spectral graph sparsification in nearly-linear time leveraging efficient spectral perturbation analysis. In *Proceedings of the 53rd Annual Design Automation Conference, DAC 2016, Austin, TX, USA, June 5-9, 2016*, pages 57:1–57:6. ACM, 2016.

[110] Paolo Boldi and Sebastiano Vigna. The webgraph framework I: compression techniques. In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*, pages 595–602. ACM, 2004.

[111] Hossein Maserrat and Jian Pei. Community Preserving Lossy Compression

of Social Networks. In Mohammed Javeed Zaki, Arno Siebes, Jeffrey Xu Yu, Bart Goethals, Geoffrey I. Webb, and Xindong Wu, editors, *12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, December 10-13, 2012*, pages 509–518. IEEE Computer Society, 2012.

[112] Daniel A. Spielman and Shang-Hua Teng. Spectral Sparsification of Graphs. *arXiv:0808.4134 [cs]*, July 2010. arXiv: 0808.4134.

[113] Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. Graph Clustering with Graph Neural Networks. *J. Mach. Learn. Res.*, 24:127:1–127:21, 2023.

[114] Yuanyuan Tian, Richard A. Hankins, and Jignesh M. Patel. Efficient aggregation for graph summarization. In Jason Tsong-Li Wang, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 567–580. ACM, 2008.

[115] Shalev Itzkovitz, Reuven Levitt, Nadav Kashtan, Ron Milo, Michael Itzkovitz, and Uri Alon. Coarse-graining and self-dissimilarity of complex networks. *Physical Review. E, Statistical, Nonlinear, and Soft Matter Physics*, 71(1 Pt 2):016127, January 2005.

[116] Hannu Toivonen, Fang Zhou, Aleksi Hartikainen, and Atte Hinkka. Compression of weighted graphs. In Chid Apté, Joydeep Ghosh, and Padhraic Smyth, editors, *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 965–973. ACM, 2011.

[117] Yll Haxhimusa and Walter G. Kropatsch. Segmentation Graph Hierarchies. In Ana L. N. Fred, Terry Caelli, Robert P. W. Duin, Aurélio C. Campilho, and Dick de Ridder, editors, *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshops, SSPR 2004 and SPR 2004, Lisbon, Portugal, August 18-20, 2004 Proceedings*, volume 3138 of *Lecture Notes in Computer Science*, pages 343–351. Springer, 2004.

[118] R. Marfil, L. Molina-Tanco, A. Bandera, and F. Sandoval. The Construction of Bounded Irregular Pyramids with a Union-Find Decimation Process. In Francisco Escolano and Mario Vento, editors, *Graph-Based Representations in Pattern Recognition*, Lecture Notes in Computer Science, pages 307–318, Berlin, Heidelberg, 2007. Springer.

[119] Horst Bunke, Peter J. Dickinson, Miro Kraetzl, and Walter D. Wallis. Matching Hierarchical Graphs. In *A Graph-Theoretic Approach to Enterprise Network Dynamics*, Progress in Computer Science and Applied Logic (PCS), pages 199–210. Birkhäuser, Boston, MA, 2007.

[120] Huaijun Qiu and Edwin R. Hancock. Graph matching and clustering using spectral partitions. *Pattern Recognit.*, 39(1):22–34, 2006.

[121] Pau Riba, Josep Lladós, and Alicia Fornés. Hierarchical graphs for coarse-to-fine error tolerant matching. *Pattern Recognition Letters*, 134:116–124, June 2020.

[122] Matthew Hutson. Artificial intelligence faces reproducibility crisis. *Science*, 359(6377):725–726, 2018.

[123] Joy Buolamwini and Timnit Gebru. Gender Shades: Intersectional Accu-

racy Disparities in Commercial Gender Classification. In Sorelle A. Friedler and Christo Wilson, editors, *Conference on Fairness, Accountability and Transparency, FAT 2018, 23-24 February 2018, New York, NY, USA*, volume 81 of *Proceedings of Machine Learning Research*, pages 77–91. PMLR, 2018.

[124] Galit Shmueli. To Explain or to Predict? *Statistical Science*, 25(3), 2010.

[125] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A Fair Comparison of Graph Neural Networks for Graph Classification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[126] Till Schulz and Pascal Welke. On the Necessity of Graph Kernel Baselines. In *Graph Embedding and Mining Workshop at ECML PKDD*, pages 0–13, 2019.

[127] Jeremy Ramsden. *Bioinformatics - An Introduction, 4th Edition*. Computational Biology. Springer, 2023.

[128] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. TUDataset: A collection of benchmark datasets for learning with graphs. *arXiv:2007.08663 [cs, stat]*, July 2020.

[129] Development Therapeutics Program. AIDS Antiviral Screen, 2004.

[130] Kaspar Riesen, Michel Neuhaus, and Horst Bunke. Graph Embedding in Vector Spaces by Means of Prototype Selection. In Francisco Escolano and Mario Vento, editors, *Graph-Based Representations in Pattern Recognition, 6th IAPR-TC-15 International Workshop, GbRPR 2007, Alicante, Spain, June 11-13, 2007, Proceedings*, volume 4538 of *LNCS*, pages 383–393. Springer, 2007.

[131] 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.

[132] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, February 1991.

[133] Ashwin Srinivasan, Stephen H. Muggleton, Michael J. E. Sternberg, and Ross D. King. Theories for Mutagenicity: A Study in First-Order and Feature-Based Induction. *Artif. Intell.*, 85(1-2):277–299, 1996.

[134] Jeroen Kazius, Ross McGuire, and Roberta Bursi. Derivation and Validation of Toxicophores for Mutagenicity Prediction. *Journal of Medicinal Chemistry*, 48(1):312–320, 2005. Publisher: American Chemical Society.

[135] 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.

[136] Natiobnal Center for Biotechnology Information. The PubChem Project.

[137] Christoph Helma, Ross D. King, Stefan Kramer, and Ashwin Srinivasan. The Predictive Toxicology Challenge 2000-2001. *Bioinform.*, 17(1):107–108,

2001.

[138] Thomas Dandekar and Meik Kunz. *Bioinformatics - An Introductory Textbook*. Springer, 2023.

[139] 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. Backup Publisher: National Institute of Allergy and Infectious Diseases, National Institutes of Health, Rockville, Maryland, USA Publisher: Wiley Online Library.

[140] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003. Publisher: Elsevier.

[141] 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.

[142] Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, June 2005.

[143] 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.

[144] Shirui Pan, Jia Wu, Xingquan Zhu, Guodong Long, and Chengqi Zhang. Task Sensitive Feature Exploration and Learning for Multitask Graph Classification. *IEEE Trans. Cybern.*, 47(3):744–758, 2017.

[145] R. Cameron Craddock, G. Andrew James, Paul E. Holtzheimer, Xiaoping P. Hu, and Helen S. Mayberg. A whole brain fMRI atlas generated via spatially constrained spectral clustering. *Human Brain Mapping*, 33(8):1914–1928, August 2012.

[146] Qi Xuan, Yun Xiang, and Dongwei Xu. *Deep Learning Applications - In Computer Vision, Signals and Networks*. WorldScientific, 2023.

[147] Sheng Wan, Shirui Pan, Shengwei Zhong, Jie Yang, Jian Yang, Yibing Zhan, and Chen Gong. Multi-level graph learning network for hyperspectral image classification. *Pattern Recognition*, 129:108705, 2022.

[148] John M. Winn, Antonio Criminisi, and Thomas P. Minka. Object Categorization by Learned Universal Visual Dictionary. In *10th IEEE International Conference on Computer Vision (ICCV 2005), 17-20 October 2005, Beijing, China*, pages 1800–1807. IEEE Computer Society, 2005.

[149] Marion Neumann, Roman Garnett, Christian Bauckhage, and Kristian Kersting. Propagation kernels: efficient graph kernels from propagated information. *Mach. Learn.*, 102(2):209–245, 2016.

[150] Mark E. J. Newman. *Networks: An Introduction*. Oxford University Press, 2010.

[151] Pinar Yanardag and S.V.N. Vishwanathan. Deep Graph Kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374. Association for Computing Machinery, New York, NY, USA, August 2015.

[152] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In Robert Grossman, Roberto J. Bayardo, and Kristin P. Bennett, editors, *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, pages 177–187. ACM, 2005.

[153] Kay Henning Brodersen, Cheng Soon Ong, Klaas Enno Stephan, and Joachim M. Buhmann. The Balanced Accuracy and Its Posterior Distribution. In *20th International Conference on Pattern Recognition, ICPR 2010, Istanbul, Turkey, 23-26 August 2010*, pages 3121–3124. IEEE Computer Society, 2010.

[154] Remco R. Bouckaert. Choosing Between Two Learning Algorithms Based on Calibrated Tests. In Tom Fawcett and Nina Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 51–58. AAAI Press, 2003.

[155] Andreas Fischer, Réjean Plamondon, Yvon Savaria, Kaspar Riesen, and Horst Bunke. A Hausdorff Heuristic for Efficient Computation of Graph Edit Distance. In Pasi Fränti, Gavin Brown, Marco Loog, Francisco Escolano, and Marcello Pelillo, editors, *Structural, Syntactic, and Statistical Pattern Recognition*, Lecture Notes in Computer Science, pages 83–92, Berlin, Heidelberg, 2014. Springer.

[156] 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.

[157] 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.

[158] 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.

[159] 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. *arXiv:1810.07491 [cs]*, 11004:470–480, 2018. arXiv: 1810.07491.

[160] Mathias Fuchs and Kaspar Riesen. Matching of Matching-Graphs - A Novel Approach for Graph Classification. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 6570–6576, January 2021. ISSN:

1051-4651.

[161] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1):107–117, April 1998.

[162] Linton C. Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40(1):35–41, 1977.

[163] Arnoldo Uber Junior, Ricardo Azambuja Silveira, Paulo José de Freitas Filho, Julio Cezar Uzinski, and Reinaldo Augusto da Costa Bianchi. MASDES-DWMV: Model for Dynamic Ensemble Selection Based on Multiagent System and Dynamic Weighted Majority Voting. In Lourdes Martínez-Villaseñor, Oscar Herrera-Alcántara, Hiram E. Ponce, and Félix Castro-Espinoza, editors, *Advances in Computational Intelligence - 19th Mexican International Conference on Artificial Intelligence, MICAI 2020, Mexico City, Mexico, October 12-17, 2020, Proceedings, Part II*, volume 12469 of *Lecture Notes in Computer Science*, pages 419–434, 2020.

[164] A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd edition, 2015.

[165] Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, August 2007.

[166] Ulrike von Luxburg. A tutorial on spectral clustering. *Stat. Comput.*, 17(4):395–416, 2007.

[167] Mariá Cristina Vasconcelos Nascimento and André Carlos Ponce de Leon Ferreira de Carvalho. Spectral methods for graph clustering - A survey. *Eur. J. Oper. Res.*, 211(2):221–231, 2011.

[168] Jacob Lurie. Review of Spectral Graph Theory: by Fan R. K. Chung. *SIGACT News*, 30(2):14–16, 1999.

[169] Na Shi, Xumin Liu, and Yong Guan. Research on k-means Clustering Algorithm: An Improved k-means Clustering Algorithm. In *Third International Symposium on Intelligent Information Technology and Security Informatics, IITSI 2010, Jinggangshan, China, April 2-4, 2010*, pages 63–67. IEEE Computer Society, 2010.

[170] Yu Jin, Andreas Loukas, and Joseph F. JáJá. Graph Coarsening with Preserved Spectral Properties. In Silvia Chiappa and Roberto Calandra, editors, *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 of *Proceedings of Machine Learning Research*, pages 4452–4462. PMLR, 2020.

[171] Arpit Merchant, Michael Mathioudakis, and Yanhao Wang. Graph Summarization via Node Grouping: A Spectral Algorithm. In Tat-Seng Chua, Hady W. Lauw, Luo Si, Evimaria Terzi, and Panayiotis Tsaparas, editors, *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining, WSDM 2023, Singapore, 27 February 2023 - 3 March 2023*, pages 742–750. ACM, 2023.

[172] Lu Bai, Lixin Cui, Yuhang Jiao, Luca Rossi, and Edwin R. Hancock. Learning Backtrackless Aligned-Spatial Graph Convolutional Networks for Graph Classification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(2):783–

798, 2022.

[173] David B. Blumenthal, Nicolas Boria, Sébastien Bougleux, Luc Brun, Johann Gamper, and Benoit Gaüzère. Scalable generalized median graph estimation and its manifold use in bioinformatics, clustering, classification, and indexing. *Inf. Syst.*, 100:101766, 2021.

[174] Hoda Bahonar, Abdolreza Mirzaei, Saeed Sadri, and Richard C. Wilson. Graph Embedding Using Frequency Filtering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 43(2):473–484, 2021.

[175] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation Learning on Graphs: Methods and Applications. *arXiv:1709.05584 [cs]*, April 2018. arXiv: 1709.05584.

[176] Hongyang Gao and Shuiwang Ji. Graph U-Nets. *arXiv:1905.05178 [cs, stat]*, May 2019. arXiv: 1905.05178.

[177] Christofer Fellicious, Thomas Weissgerber, and Michael Granitzer. Effects of Random Seeds on the Accuracy of Convolutional Neural Networks. In Giuseppe Nicosia, Varun Ojha, Emanuele La Malfa, Giorgio Jansen, Vincenzo Sciacca, Panos Pardalos, Giovanni Giuffrida, and Renato Umeton, editors, *Machine Learning, Optimization, and Data Science*, Lecture Notes in Computer Science, pages 93–102, Cham, 2020. Springer International Publishing.

[178] Leonid Peshkin. Structure induction by lossless graph compression. In *2007 Data Compression Conference (DCC 2007), 27-29 March 2007, Snowbird, UT, USA*, pages 53–62. IEEE Computer Society, 2007.

[179] Morihiro Hayashida and Tatsuya Akutsu. Comparing biological networks via graph compression. *BMC Syst. Biol.*, 4(S-2):S13, 2010.

[180] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul M. B. Vitányi. The similarity metric. *IEEE Trans. Inf. Theory*, 50(12):3250–3264, 2004.

[181] Zhiying Jiang, Matthew Y. R. Yang, Mikhail Tsirlin, Raphael Tang, Yiqin Dai, and Jimmy Lin. "Low-Resource" Text Classification: A Parameter-Free Classification Method with Compressors. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 6810–6828. Association for Computational Linguistics, 2023.

[182] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[183] É. Rivals, M. Dauchet, J. P. Delahaye, and O. Delgrange. Compression and genetic sequence analysis. *Biochimie*, 78(5):315–322, January 1996.

[184] Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications, Third Edition*. Texts in Computer Science. Springer, 2008.

[185] Charles H. Bennett, Péter Gács, Ming Li, Paul M. B. Vitányi, and Wojciech H. Zurek. Information Distance. *CoRR*, abs/1006.3520, 2010. arXiv: 1006.3520.

[186] Elizabeth H. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In Solomon L. Pollack, Thomas R. Dines, Ward C. Sangren, Norman R. Nielsen, William G. Gerkin, Alfred E. Corduan, Len Nowak, James L. Mueller, Joseph Horner III, Pasteur S. T. Yuen, Jeffery Stein, and Margaret M. Mueller, editors, *Proceedings of the 24th national conference, ACM 1969, USA, 1969*, pages 157–172. ACM, 1969.

[187] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, June 2002. Publisher: Proceedings of the National Academy of Sciences.

[188] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106, September 2007. Publisher: American Physical Society.

[189] Vincent D. Blondel, Jean-Loup Guillaume, and Renaud Lambiotte. Fast unfolding of communities in large networks: 15 years later. *CoRR*, abs/2311.06047, 2023. arXiv: 2311.06047.

[190] Arnau Prat-Pérez, David Dominguez-Sal, and Josep Lluís Larriba-Pey. High quality, scalable and parallel community detection for large real graphs. In Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel, editors, *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*, pages 225–236. ACM, 2014.

[191] Arnau Prat-Pérez, David Dominguez-Sal, Josep M. Brunat, and Josep Lluís Larriba-Pey. Shaping communities out of triangles. In Xue-wen Chen, Guy Lebanon, Haixun Wang, and Mohammed J. Zaki, editors, *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pages 1677–1681. ACM, 2012.

[192] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 23(3):337–343, 1977.

[193] Alistair Moffat. Huffman Coding. *ACM Comput. Surv.*, 52(4):85:1–85:35, 2019.