UNIVERSITÄT
BERN

Analysis of Matching Graphs

Bachelor Thesis

Veronika Zhao Xuan Wu

Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

Autumn Semester 2020

PD Dr. Kaspar Riesen

Institut für Informatik
University of Bern, Switzerland

Abstract

In recent years the amounts of data available in diverse areas has increased rapidly. Given the magnitude and complexity of the data, pattern recognition as a subarea of machine learning has transpired to be very important in Computer Science. Therefore a large amount of graph based methods for pattern recognition have emerged. In this thesis there will be an introduction to graph based pattern recognition and a description of a novel approach for graph classification. Given this novel approach for graph classification - the matching of matching graphs - an analysis will be performed on said matching graphs. Further we will analyze the quality and the quantity of the matching graphs by means of an algorithm for subgraph isomorphism by J.R. Ullmann.

Contents

1 Introduction

Computer Science covers a whole range of fields. From algorithms, to networking and communication, to human-computer interactions and much more. This thesis belongs to the field of machine learning and artificial intelligence. Put simply, the field of machine learning tries to answer the following question: "How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes?" [14]. By way of illustration a few real-world examples:

- Handwriting recognition refers to the ability of a computer to receive and interpret handwritten input as text [17]. This input can come in form of paper documents, photographs, touch-screens and other devices. Handwriting recognition methods can broadly be classified into two types: online methods and offline methods [15]. Online methods involve a digital pen/stylus of some sort and use dynamic information like stroke order, writing speed and pen pressure to differentiate between different characters and ultimately recognize the text. Offline methods on the other hand take input where the text has already been written down. There is only static information available, like the shapes of handwritten characters, and possibly some background noise from the source itself e.g. paper.

  One subapplication of handwriting recognition is signature recognition. Since a signature is a handwritten representation of the name of a person and the person must become active for signing, handwritten signatures and personal signs can be seen as behavioral biometric characteristics of the person [15]. Handwritten signatures are generally used for verification rather than for identification. That's to say, they are generally used for confirming a claimed identity through one-to-one

comparison of biometric features [15]. Seminal contributers in this field include F. Leclerc and R. Plamondon [6] who have been working in this field for over two decades.

- Biometric pattern recognition systems are pattern recognition systems which use biological traits to recognize individuals. A recent contribution from S. Hariprasath and M. Santhi [21] proposes an automated biometric authentication system which accepts the individual's biometric data (in this case from iris and palmprint), extracts the features and compares those features againt the template stored in the database. Based on the outcome of this comparison the identity is either verified or the individual's identity is determined.

- Machine learning can also be applied in chemistry. A general statement of the pattern recognition problem in chemistry is: "given a collection of objects characterized by a list of measurements made on each object, is it possible to find and/or predict a useful property of the objects that may not be measureable, but is thought to be related to the measurements?" [2]. The objects observed can range from pure chemical compounds to complex mixtures and chemical reactions. Measuring the structural similarity between proteins is another common application of pattern recognition in chemistry [22].

As indicated by the examples above the area of pattern recognition is a big part of the field of Machine Learning. Pattern recognition transpired to be very important in Computer Science. When breaking down a complex problem we often find patterns among the smaller problems created. Pattern recognition involves finding these similarities or patterns among small, decomposed problems that can help us solve more complex problems. We rely on pattern recognition to verify our identities/signatures [15], to predict protein function and strctures [1], detect Alzheimer's Disease [9] and much more.

The goal of pattern recognition in general is to define algorithms, which automate or at least support the process of recognizing patters stemming from the real world, like for example in handwriting recognition [19].
The IAPR (International Association for Pattern Recognition) [16] hosts several conferences every year. The wide range of topics discussed include Image Mining, Mobile and Wearable Biometrics, Computer Vision and of course Pattern Extraction and Recognition. As the recent conference "IAPR Joint International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)" (S+SSPR 2020) suggests, we can differentiate between statistical and structural pattern recognition (see Figure 1.1). That's to say, we distinguish the way in which we present the patterns to our computer.

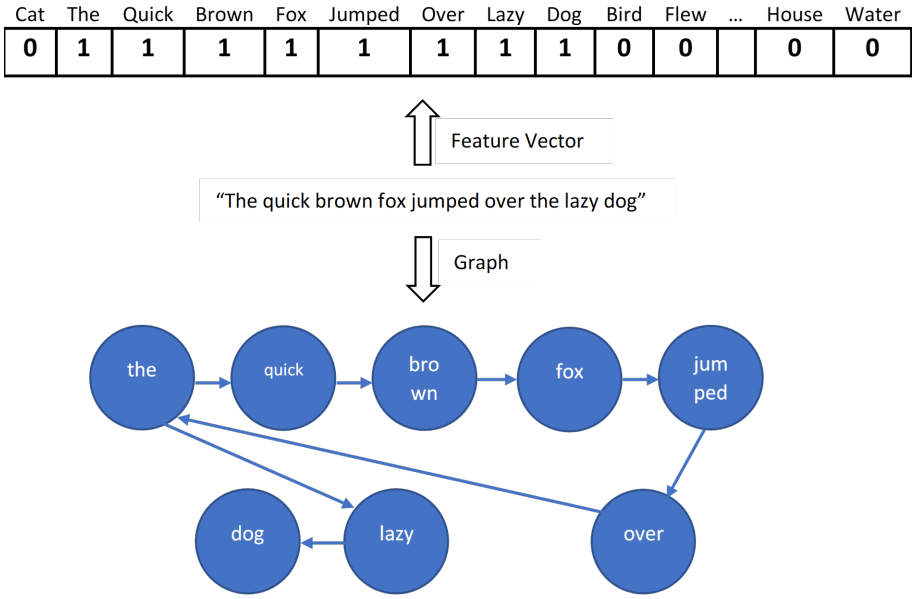


Figure 1.1: The same sentence represented as feature vector and as graph.

The statistical approach uses feature vectors and numbers. The pattern is represented as a vector x = (x_1, x_2, ..., x_n) $\in R^n$ of n numerical features. The mathematical operations available in vector space enable a very efficient performance and result in a rich repository of algorithmic tools for statistical pattern recognition. However, there are also limitations to consider: since vectors always represent a predefined set of features, all vectors have the same length. This can not be changed even if the complexity or the size of the corresponding pattern varies. Another disadvantage of feature vectors is, that there is no direct possibility to describe binary (or higher-order) relationships between different parts of the pattern [19].

The structural approach focuses on symbolic data structures like strings, trees and graphs. Graphs are the most common data structure in Computer Science. They consist of a finite set of nodes connected by edges. Strings and trees are special cases of graphs [19]. A string can be represented as a graph, in which each node is one character and the next character is connected with the previous one by an edge. A tree is a graph where any two nodes are connected by exactly one edge. The limitations of feature vectors mentioned above can be overcome by graphs. Graphs can describe properties of a pattern and also (binary) relationships among different parts. Another advantage of graphs is, that they can be adjusted to any size. The number of nodes and edges is not limited and can be adapted to the complexity of the pattern. The most prominent drawback of graphs, however, is that because of the flexibility they offer, the complexity of many algorithms is increased significantly compared to the feature vectors [19]. One real-world example

of graphs representing proteins can be seen in Figure 1.2.

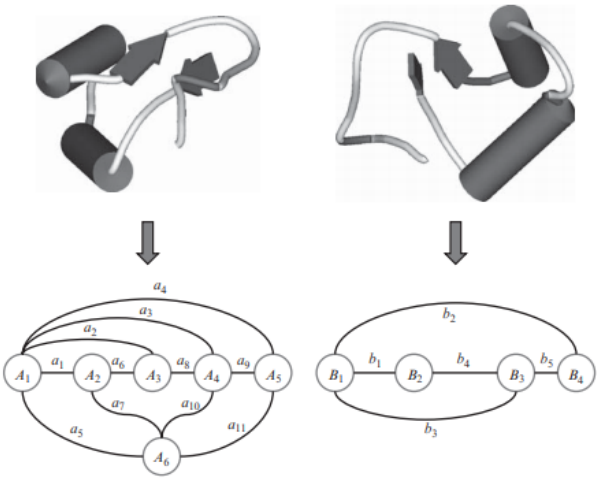


Figure 1.2: Protein graph transformation [22].

In this thesis the focus lies on structural pattern recognition. More precisely graph matching. Graph matching describes the process of finding similarities between two graphs. This (dis-) similarity between graphs can be measured with the so called graph edit distance. If we compare several graphs with each other, we can gather information about the similarities of the whole set of graphs and store this information in a new graph, called the matching graph [10]. This is an idea similar in spirit to the maximal common subgraph [5].

In a recent paper by M. Fuchs and K. Riesen [10] it has been proposed that considering a classification problem these new structures, the matching graphs, can be used to solve it. Given multiple classes of graphs A, B, ..., Z and a new graph, which has not yet been classified. The task is to decide in which class the new graph belongs to. One way to solve this problem, is by creating matching graphs (which represent the core structures) for each class, and comparing the new graph to those matching graphs. Based on the outcome of this comparison we then assign the new graph to its class. My research question relates to these newly generated matching graphs:

Do matching graphs created from a given class A appear more often in class A or in any of the other classes B-Z?

The idea is to find out, if matching graphs from class A actually appear as subgraphs in the graphs of class A. If so, how often in comparison with the other classes B,..., Z.

The remainder of this thesis is structured as follows. Chapter 2 provides basic definitions, terms and algorithms used throughout this paper. Next, in Chapter 3, matching graphs are introduced and the research question will be formally formulated. Chapter 4 holds the experimental evaluation and in Chapter 5 we conclude the thesis and discuss some future work ideas.

2 Graph Representation and Graph Matching

In this chapter we give the reader the necessary theoretical background to understand this thesis. We will present an overview over the representation of graphs and discuss different categories and methods of graph matching. Further we take a look at one particular algorithm - the Algorithm for Subgraph Isomorphism by Ullmann. This algorithm is based on the concept of graph isomorphism and has been used for the experimental evaluation in this thesis and will therefore be discussed in detail.

2.1 Graphs

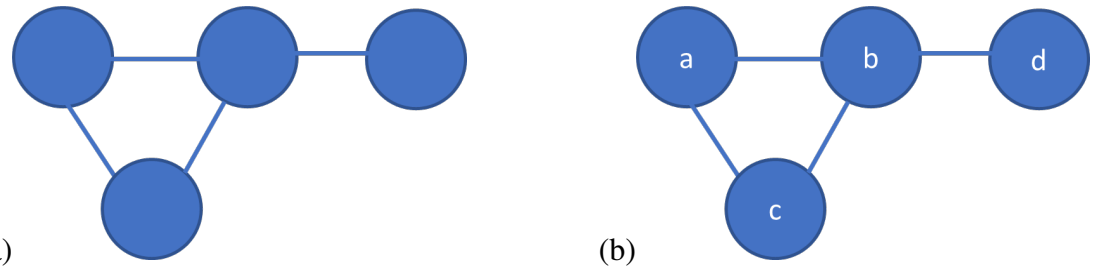
This section holds the definition of a graph and advantages as well as disadvantages of this particular data structure.

Definition (Graph) [19]. Let L_V and L_E be finite or infinite label sets for nodes and edges, respectively. A *graph* is a four-tuple g = (V, E, μ, ν), where

- V is the finite set of nodes,
- E $\subseteq$ V $\times$ V is the set of edges,
- μ : V $\rightarrow$ L_V is the node labeling function, and

- $\nu : \mathrm{E} \rightarrow \mathrm{L}_E$ is the edge labeling function.

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



(a) (b)

Figure 2.1: Examples of graphs.
(a) Graph size 4, where L_V and L_E are empty.
(b) Graph size 4, where $\mu = (1 \rightarrow \mathrm{a}, 2 \rightarrow \mathrm{b}, 3 \rightarrow \mathrm{c}, 4 \rightarrow \mathrm{d})$ and L_E is empty.

Advantages of graphs include:

- Modelling real-world problems e.g. road networks, where intersections can be considered as nodes and the road segments between them as edges [7].
- One can easily work with large data, since the size of graphs is not limited [7].
- They are an effective visual tool (present information quickly as well as easily) [7].
- One can map (binary) relations between individual data points [19].

Disadvantages of graphs that need to be considered are:

- The complexity of algorithms is increased significantly [19].
- High computational costs resulting from the complexity [5].

2.2 Graph Matching

In this section we discuss graph matching and take a closer look at the different categories and methods of graph matching.

As already mentioned, graph matching describes the process of finding similarities between two graphs. A more precise definition reads as follows:

Definition (Graph Matching) [5]. "Graph matching is the process of finding a correspondence between the nodes and the edges of two graphs that satisfies some (more or less stringent) constraints ensuring that similar substructures in one graph are mapped to similar substructures in the other".

There exist two categories of matching methods: Exact matching methods and inexact matching methods. As the name suggests, the exact matching methods require a strict correspondence/agreement among the two matched objects. In different words: "The aim in exact graph matching is to determine whether two graphs, or at least part of them, are identical in terms of structure and labels." [5]. Whereas inexact matching methods can be used on graphs, which are structurally different to some extent.

2.2.1 Overview of Exact Graph Matching

The most prominent characteristics of exact matching methods is that the mapping between the nodes must be edge-preserving. That's to say if two nodes in the first graph are connected by an edge, they must be mapped onto two nodes in the second graph which are also linked by an edge.
Exact graph matching has exponential time complexity in the worst case since no polynomial algorithms are known for the general case. However the actual computation time can still be acceptable in many cases because of two reasons: first, the graphs which the algorithms are performed on usually differ from the worst case graphs. And second, node and edge attributes can often be used to reduce search time drastically.
There are different forms of exact graph matching: graph isomorphism, subgraph isomorphism, monomorphism, homomorphism and maximum common subgraph. The matching problems above are all NP-complete except for graph isomorphism (for which it has not yet been demonstrated if it belongs to NP or not) [5].

Most of the exact graph matching methods are based on some form of tree search with backtracking. The basic idea is, that a partial match (initially empty) is iteratively expanded by adding new pairs of matched nodes to it. The new pair is chosen by using some condition which ensures that all the constraints are respected. Additionally heuristics can be used to prune futile search paths as early as possible [5].
One important algorithm from this family is a contribution from Ullmann in 1976 [5]. The algorithm can be used to solve the problems of graph isomorphism, subgraph isomorphism and monomorphism. The method proposed to prune unsuccessful matches in this algorithm is the so called *refinement procedure*. It works on a matrix of possible future matches and removes these node pairs, which are not consistent with the current partial matching. This algorithm will be discussed in more detail in Section 2.3.1.
Other techniques used in exact graph matching methods include group theory [13],

recursive decomposition [3] and decision trees [12].

2.2.2 Overview of Inexact Graph Matching

Two main reasons have led to the development inexact graph matching: first, the constraints imposed by exact graph matching are in some cases too stringent because the observed graphs are subject to deformations. These deformations can have several causes: intrinsic variability of the patterns themselves and noise in the acquisition process are among the possible reasons why the actual graphs are different from their ideal models [5]. Second, exact graph matching algorithms require in the worst case exponential time. This can be very costly and may lead to the usage of algorithms, which do not guarantee to find the best solution, but can give a good approximate solution more cheaply and in reasonable time.
Frequently, inexact matching algorithms do not forbid the matching between two nodes that do not satisfy the edge-preservation requirements like the exact matching algorithms, but they assign a penalty to it. This usually happens in form of an assigned cost, depending on the mismatch (also considering other differences e.g. different attributes). As a result inexact graph matching algorithms have to find a mapping that minimizes the matching costs.

Similar to exact graph matching methods, there exist inexact graph matching methods that are based on tree search with backtracking. The basic idea here is, that the algorithm uses the cost of the partial match already obtained and some kind of heruristic estimate of the matching cost of the remaining nodes. This information can be used to either prune futile branches or to determine in which order the tree will be traversed.
The first tree based inexact algorithm has been proposed by Tsai and Fu in 1979 [24][5]. The authors introduce a formal definition of error-correcting graph matching based on the introduction of a graph edit cost (each type of error will be assigned a different cost). The proposed heuristic is based on the computation of the future node matching cost. By neglecting the constraint that the mapping has to be injective, this search method ensures to find the optimal solution. In their first algorithm (1979) the graphs being matched are still required to be structurally isomorphic because the authors only allowed the operations of node and edge substitution. But in the extension proposed in 1983 the same authors also considered insertion and deletion of nodes and edges, allowing for an error-correcting subgraph isomorphism [5].
Another family of inexact matching methods stems from continuous optimization. The idea is to turn the graph matching problem (so far a discrete optimization problem) into a continuous, nonlinear optimization problem. Then known optimization algorithms become available to solve this task. After applying the optimization algorithms the solution has to be converted back into the original discrete problem. These algorithms do not ensure the optimality of the solution but they often reduce computational costs [5].

One of the pioneering contributions comes from Fischler and Elschlager in 1973, where the authors propose the usage of relaxation labeling [8][5].

2.3 Graph Isomorphism

This section covers the definition of a subgraph and (sub-)graph isomorphism and elaborations on the Algorithm for Subgraph Isomorphism by Ullmann.

Definition (Subgraph) [19]. Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be graphs. Graph g_1 is a *subgraph* of g_2, denoted by $g_1 \subseteq g_2$, if

- $V_1 \subseteq V_2$,
- $E_1 \subseteq E_2$,
- $\mu_1(u) = \mu_2(u)$ for all $u \in V_1$, and
- $\nu_1(e) = \nu_2(e)$ for all $e \in E_1$.

Definition (Graph Isomorphism) [19]. Given two graphs $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$. A *graph isomorphism* is a bijective function $f: V_1 \to V_2$ satisfying

- $\mu_1(u) = \mu_2(f(u))$ for all nodes $u \in V_1$,
- for each edge $e_1 = (u, v) \in E_1$, there exists an edge $e_2 = (f(u), f(v)) \in E_2$ such that $\nu_1(e_1) = \nu_2(e_2)$,
- for each edge $e_2 = (u, v) \in E_2$, there exists an edge $e_1 = (f^{-1}(u), f^{-1}(v)) \in E_1$ such that $\nu_1(e_1) = \nu_2(e_2)$.

Two graphs g_1 and g_2 are called *isomorphic* if there exists an isomorphism between them, and we write $g_1 \approxeq g_2$.

Definition (Subgraph Isomorphism) [19]. Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be graphs. An injective function $f: V_1 \to V_2$ from g_1 to g_2 is a *subgraph isomorphism* if there exist a subgraph $g \subseteq g_2$ such that f is a graph isomorphism between g_1 and g. In this case we write $g_1 \subseteq g_2$.

It is important to note, that the definitions of (sub-) graph isomorphism as described above only work for alphabets L_{V_i} and L_{E_i} with discrete labels, like for example numbers or strings. The experimental evaluation as described in Chapter 4 has been performed on graphs with non discrete labels: coordinates. Therefore we use the following heuristic to

adapt to the concepts of (sub-) graph isomorphism:

μ_1(u) = μ_2(f(u)) iff $\| \mu_1(u) - \mu_2(f(u)) \| \leq c$, where c is constant.

This means, assuming each node has an X and a Y coordinate, we label two nodes as equal in their coordinate attributes if the Euclidean distance between them is less than a constant c.

2.3.1 Ullmann Algorithm

Part of the research question involves counting how many matching graphs appear as subgraph in the original graphs from their respective classes. This is achieved by implementing the Subgraph Isomorphism algorithm by Ullmann 1976 [23].

The general idea of this algorithm is to achieve some sort of enumeration and performing isomorphism checks when appropriate. The method proposed by Ullmann is to determine isomorphism by brute-force enumeration for which a depth-first tree search algorithm is used. Given two graphs *g1* and *g2* it is possible to list all the possible mappings from nodes in graph *g1* to nodes in graph *g2*. This list is generated by saving all the mappings in a tree data structure and then traversing it depth-first (see Figure 2.2).

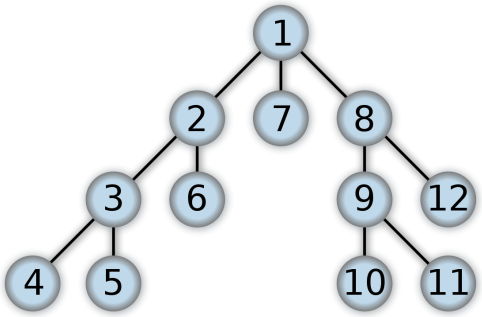


Figure 2.2: Depth-first traversal of a tree. The order in which the nodes will be traversed is: 1-2-3-4-5-6-7-8-9-10-11-12 [4].

Since the number of possible mappings is enormous even between two relatively small graphs, the author suggests using some of the characteristics of graphs to reduce the number of mappings considered. One such characteristics is the degree of a node.

Definition (Degree) [19]. Given a graph g = (V, E, μ, ν) the *degree* of a node $u \in$ V, referred to as *deg(u)*, is the number of adjacent nodes to u.

It is evident from the definition of (sub-) graph isomorphism, that mappings from nodes in *g1* with degree higher than the degree from nodes in *g2* need not be considered. Other characteristics which can be used to reduce the number of mappings are the attributes of the nodes and edges from *g1* resp. *g2*. For example when comparing two molecules, where the nodes represent the atoms and the edges represent the connections between the atoms, it makes little sense to map an H-atom from *g1* to a C-atom in *g2*.

Now that we have seen how the enumeration works, let's take a closer look at how Ullmann proposed to perform the actual isomorphism checks. The way chosen by the author is to utilise matrices. Therefore a way of translating graphs into matrices is necessary. One common way to do that is to create an adjacency matrix.

Definition (Adjacency matrix) [19]. Let g = (V, E, μ, ν) be a graph with $|g|$ = n. The *adjacency matrix* A = $(a_{ij})_{n \times n}$ of graph g is defined by

$$a_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E, \\ 0 & \text{otherwise} \end{cases} \quad \text{where } v_i \text{ and } v_j \text{ are nodes from g, i.e. } v_i, v_j \in V.$$

Figure 2.3: Adjacency matrix A generated by the graph on the left.

Given graph g_1, with size l and adjacency matrix A and graph g_2, with size k and adjacency matrix B. The isomorphism check between g_1 and subgraphs of g_2 consists of the following parts:

- Define a binary matrix M = (m_{ij}) with l rows $\times$ k columns
- M can then be used to permute the rows and columns of the adjacency matrix B of g_2 s.d. C = (c_{ij}) = M(MB)^T
- If it is true that
  $(\forall i\ \forall j)\ (a_{ij} = 1) \Rightarrow (c_{ij} = 1)$ for $1 \leq i, j \leq l$,
  then M specifies an isomophism between g_1 and a subgraph of g_2.

Ullmann proposes to construct the binary matrix M = (m_{ij}) as follows:

$$m_{ij} = \begin{cases} 1 & \text{if the } \deg(g_1(j)) \leq \deg(g_2(i)), \\ 0 & \text{otherwise} \end{cases}$$

In this suggestion the author only considered the degree to create the matrix M. However it is reasonable to also take into account other characteristics available e.g. attributes of nodes like mentioned above. An example for the matrix M created between graphs $g_1 = (V_1, E_1, \mu_1, \nu_1)$ where $V_1 = \{1, 2, 3\}$, $E_1 = \{(1, 2), (2, 3)\}$, $\mu_1 = (1 \rightarrow H, 2 \rightarrow O, 3 \rightarrow H)$ and ν_1 is empty and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ where $V_2 = \{1, 2, 3, 4\}$, $E_2 = \{(1, 2), (2, 3), (2, 4)\}$, $\mu_2 = (1 \rightarrow H, 2 \rightarrow O, 3 \rightarrow H, 4 \rightarrow C)$ and ν_2 is empty:

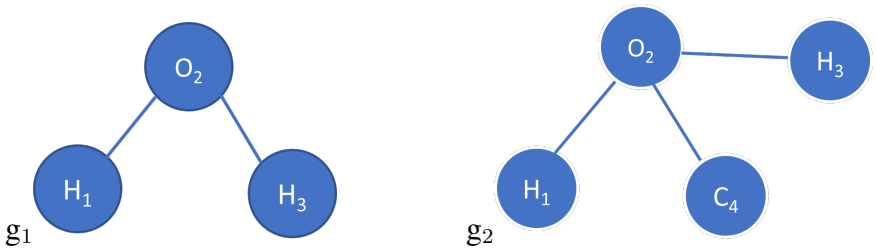


Figure 2.4: Example graphs g_1 and g_2.

$$M_0 = \begin{matrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{matrix} \quad \text{(a)} \qquad M_0 = \begin{matrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{matrix} \quad \text{(b)}$$

Figure 2.5: (a) M considering degrees only, (b) M considering degrees and attributes.

The way matrix M is constructed we can assume, that if M describes an isomorphism, we need exactly one 1 in each row and at most one 1 in each column. It is exactly for this reason, that the algorithm systematically changes all entries in a row to 0 except for one, if appropriate it checks for an isomorphism and if no isomorphism has been found it moves on to the next iteration. This will result in a tree structure which will be traversed with the depth-first search method (see Figure 2.6).

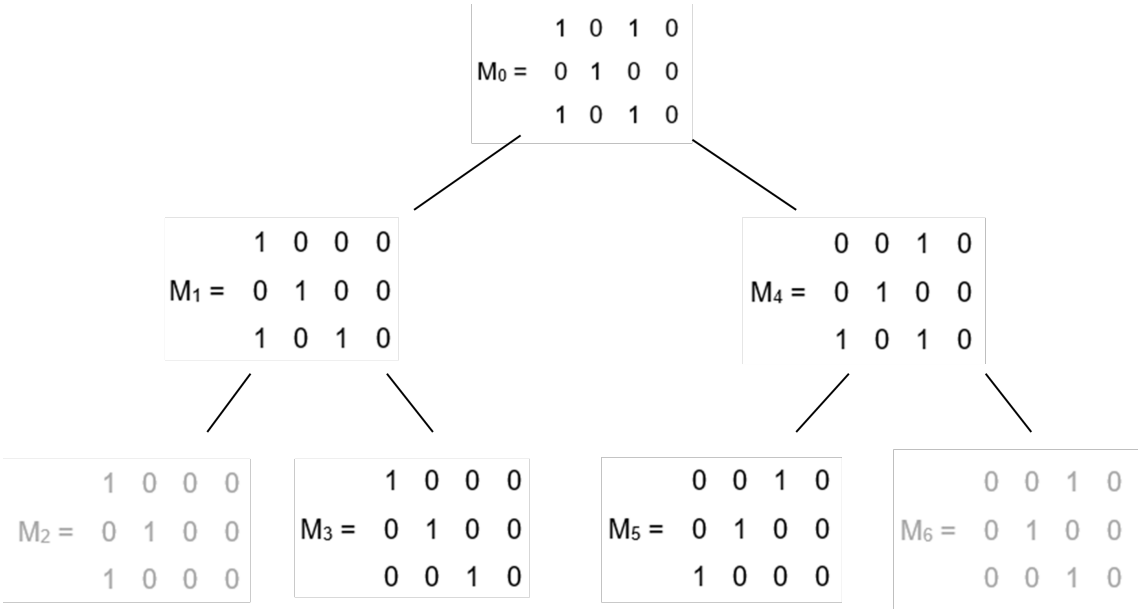


Figure 2.6: Tree structure with all the possible mappings from g_1 to g_2. The grey mappings will not actually be considered but are in the graphic for completeness.

In conclusion the enumeration algorithm proposed by Ullmann is designed to find all of the isomorphisms between a given graph g_1 and subgraphs of g_2. See Figure 2.7 for the proposed brute-force enumeration.

Step 1 $M = M^0$, $d := 1$; $H_1 = 0$;
for all $i = 1, \cdots, p_\alpha$, set $F_i := 0$;
Step 2 If there is no value of j such that $m_{dj} = 1$ and $F_j = 0$ then go to step 7;
$M_d = M$,
if $d = 1$ then $k := H_1$ else $k := 0$,
Step 3 $k := k + 1$,
if $m_{dk} = 0$ or $F_k = 1$ then go to step 3;
for all $j \neq k$ set $m_{dj} := 0$,
Step 4. If $d < p_\alpha$ then go to step 6 else use condition (1) and give output if an isomorphism is found;
Step 5 If there is no $j > k$ such that $m_{dj} = 1$ and $F_j = 0$ then go to step 7;
$M := M_d$,
go to step 3;
Step 6 $H_d = k$, $F_k = 1$; $d = d + 1$;
go to step 2,
Step 7 If $d = 1$ then terminate algorithm,
$F_k = 0$; $d := d - 1$, $M = M_d$, $k := H_d$,
go to step 5,

Figure 2.7: Steps of the subgraph isomorphism algorithm by Ullmann 1976 [23].

This algorithm can be made more efficient by implementing the refinement procedure.

Ullmann defines this to be a procedure, which reduces the number of successor nodes in the tree that must be searched by the methods mentioned before. It is entered after each node in the search tree. This naturally yields a reduction in the total computation time for the algorithm run [23]. The actual refinement procedure can be seen in the Appendix. For the purposes of this thesis it suffices to understand the following psudo code:

```
do
    for all (i,j) where M is 1
        for all neighbors x of vi in the set of nodes of g1
            if there is no neighbor y of vj s.d. M(x,y) = 1
                M(i,j) = 0
```

The idea is, to not only consider the characteristics of the current node which we are trying to match, but also information that we have about its neighbors.

Let's consider an example with (made up) molecules:

- $g_1 = (V_1, E_1, \mu_1, \nu_1)$ where
  $V_1 = \{1, 2, 3\}$, $E_1 = \{(1, 2), (2, 3)\}$, $\mu_1 = (1 \rightarrow H, 2 \rightarrow O, 3 \rightarrow H)$ and ν_1 is empty.
- $g_2 = (V_2, E_2, \mu_2, \nu_2)$ where
  $V_2 = \{1, 2, 3, 4, 5, 6\}$, $E_2 = \{(1, 2), (2, 3), (2, 4), (4, 5), (5, 6)\}$, $\mu_2 = (1 \rightarrow H, 2 \rightarrow O, 3 \rightarrow H, 4 \rightarrow C, 5 \rightarrow O, 6 \rightarrow H)$ and ν_2 is empty.

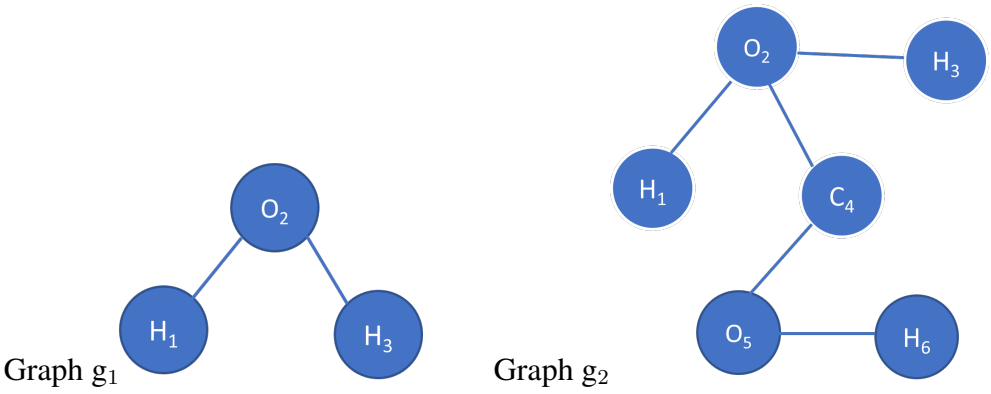


Figure 2.8: Example graphs for refinement procedure.

Considering node 2 in graph g_1, we can immediately see by checking its attribute O, that there are two candidates in graph g_2 for a match: nodes 2 and 5. Now we can also access the information about the neighbors of node 2 in graph g_1: two neighbors,

both with attribute H. With this information, we can eliminate node 5 from graph g_2 as possible match, because even though it also has two neighbors, only one of them has the H attribute. In this manner the number of possible mappings overall can be reduced and the number of necessary isomorphism checks as well.

3
Matching Graphs

In this chapter we discuss matching graphs as described in *Matching of Matching-Graphs - A Novel Approach for Graph Classification* by M. Fuchs and K. Riesen [10].

3.1 Intuition

Matching graphs from a given class can be thought of as representations of the core structure of this particular class. More precisely the authors state that the information stored in the matching graphs offer the potential to achieve a better understanding of the regularities and stable parts of a given class, as well as improve the matching quality of unknown patterns [10].

To be able to build matching graphs from a given class, we need a way of comparing and recording the similarity between two given graphs. In this case the authors chose to use the concept of graph edit distance.

3.2 Graph Edit Distance

In this section we take a closer look at the graph edit distance which is being used to construct matching graphs.

In order to define the graph edit distance we need the concept of an edit path. There is

a standard set of edit operations with which graphs can be modified: insertions, deletions and substitutions. These operations exist for both nodes and edges. With these operations it is possible to transform any source graph into any other target graph. We denote the insertion of a node $v \in V_2$ by ($\epsilon \rightarrow$ v), the deletion of a node $u \in V_1$ by (v $\rightarrow \epsilon$) and the substitution of node $u \in V_1$ and $v \in V_2$ by (u $\rightarrow$ v). For edge edit operations, similar notations can be used. To keep track of these operation we define the edit path λ.

Definition (Edit path) [10]. A set $\{e_1, ..., e_k\}$ of k edit operations e_i that transform a source graph g_1 completely into a target graph g_2 is called an *edit path* $\lambda(g_1, g_2)$ between g_1 and g_2.

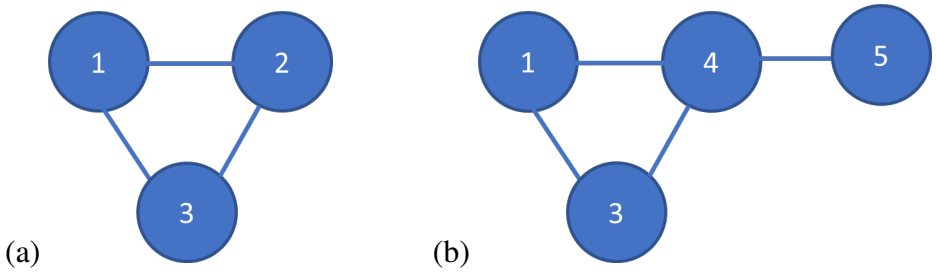


Figure 3.1: Edit path between (a) source graph g_1 and (b) target graph g_2. $\lambda(g_1, g_2) = \{1 \rightarrow 1, 2 \rightarrow 4, 3 \rightarrow 3, \epsilon \rightarrow 5\}$.

We denote the set of all edit paths transforming g_1 into g_2 as $\Upsilon(g_1, g_2)$, while c(e_i) denotes the cost function measuring the strength c(e_i) of edit operation e_i. Now each edit path has a corresponding total cost. Then we define the graph edit distance to be the path with the smallest cost out of all possible paths.

Definition (Graph Edit Distance) [10]. Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ be the source and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ the target graph. The *graph edit distance* between g_1 and g_2 is defined by $d\lambda_{min}(g_1, g_2) = \min_{\lambda \in \Upsilon(g_1, g_2)} \sum_{e_i \in \lambda} c(e_i)$.

The authors propose using the suboptimal graph edit distance algorithm, referred to as BP, to computate the graph edit distance between two graphs [10]. The algorithm BP generates a cost matrix C, which contains all the costs of all possible edit operations (substitution, insertion and deletion) that can occur between the nodes of g_1 and the nodes of g_2. Then an optimal assignment is found for the nodes of both graphs from which an approximate graph edit distance can be derived [18].

3.3 Matching Graph

This section describes how matching graphs are built from a given class.

The basic idea of a matching graph $m_{g_i \times g_j}$ is, that it represents a way to formalize information on the matching of two graphs g_i and g_j. The authors intend the matching graph $m_{g_i \times g_j}$ to represent both nodes and edges of g_i and g_j which have been matched by some particular graph matching algorithm [10].
For all pairs of graphs stemming from the same class the authors computed the graph edit distance between them by means of the suboptimal algorithm BP. This results in a (suboptimal) edit path $\lambda(g_i, g_j)$ for each pair of graphs g_i and g_j. For each edit path $\lambda(g_i, g_j)$, two matching graphs are built: $m_{g_i \times g_j}$, where g_i is the source graph and g_j the target graph, and $m_{g_j \times g_i}$ the other way around. Therefore all nodes of g_i and g_j which are substituted in the edit path are added to $m_{g_i \times g_j}$ and $m_{g_j \times g_i}$, respectively. It is important to note that all nodes which are deleted in g_i resp. g_j or inserted in g_j resp. g_i are not considered in either matching graphs. In the procedure chosen by the authors isolated nodes, i.e. nodes without neighbours, will also be removed [10].
It is clear, that if a node is not included in the matching-graph, the associated edges will not be considered either. However, edges which connect two substituted nodes might be included. Here the authors propose two different strategies for edge handling.

If two nodes $u_1, u_2 \in V_i$ of a source graph g_i are substituted with nodes $v_1, v_2 \in V_j$ in a target graph g_j there is an edge (u_1, u_2) to be considered. Edge (u_1, u_2) is included in the matching graph $m_{g_i \times g_j}$...

- ... regardless whether or not edge $(v_1, v_2) \in E_j$. In this case *no pruning* is applied to the edges.
- ... if, and only if, there is an edge $(v_1, v_2) \in E_j$. Here *pruning* is applied to the edges.

Note that the the costs for insertion and deletion have a crucial impact on the resulting edit paths and therefore on the matching graphs themselves. The higher the costs for insertion and deletion are, the more substitutions will be in the edit path, which will result in larger matching graphs [10].

Now we have two matching graphs for each pair of graphs in our given class. Assuming we have a total of *n* graphs, that gives us *n(n-1)* matching graphs. The authors propose to use the *set median graph* to reduce the number of matching graphs.

Definition (Set median graph) [10]. Let S be an arbitrary set of graphs. Then the *set median graph* $g_{mdn} \in S$ is defined as

median(S) = $g_{mdn} = \arg\min_{g_1 \in S} \sum_{g_2 \in S} d(g_1, g_2)$.

The set median graph is the graph whose sum of edit distances to all other graphs in S is minimal. The authors propose to iteratively select (and remove) the set median graph from the set of all constructed matching graphs until the desired number of matching graphs has been selected [10]. This way the matching graphs which are situated in or near the center of the complete set of matching graphs will be selected for further use.

In general, when working with a machine learning model the data set is split up into three subsets: training set, validation set and test set [20]. The idea behind this is, that we train the model with the training set. That's to say we use the data in the training set to build our model. By adapting parameters of the classifier to this data we fit our model to the training set. When our model is roughly fitted, we use the validation set to fine tune the parameters of the model. And the data set used to evaluate the final model performance is called the test set. The reason behind this is, that we get an unbiased sense of model effectiveness if the model is evaluated on samples that were not used to build or fine-tune the model [11].

3.4 Hypothesis

In this section we will take a closer look at the research question.

Assuming we have different classes A, B, ..., Z of graphs. With the method described in the previous section it is possible to create matching graphs for each of these classes. The research question reads:

Do matching graphs created from a given class A appear more often in class A or in any of the other classes B-Z?

The goal is to find out, if matching graphs from a given class A actually appear as subgraphs in the (original) graphs of class A and if so, how often compared to other classes. Of course this applies to all classes: we want to compare each class A-Z with all other classes, not only class A.

The approach used in this case is counting the occurrences of the matching graphs in the original graphs by implementing and running the Algorithm for Subgraph Isomorphism by Ullmann [23]. This algorithm as described in section 2.3.1 takes two graphs and returns the output, if an isomorphism has been found or not. With this tool, all the matching graphs have been compared with all the original graphs and the outputs collected. The outcomes are displayed and described in chapter 4.

With the results we hope to be able to find evidence that the matching graphs are a sensible representation of their class. For example, if we were to find, that matching graphs from class A really do appear more often in the original graphs from class A, then we would consider those matching graphs a good/sensible representation of their class. But if we were to find, that the matching graphs from class A do not appear at all in the original graphs from the same class, but very often in the other classes B-Z, we need to take a closer look at why this is the case.

4 Experimental Evaluation

This chapter focuses on the used data set and on the evaluation of the research question.

4.1 Data Set

The data sets used for this thesis are artificial capital letters. They have been generated by artificially distorting handwritten letters, which themselves consist only of straight lines. These kinds of data sets are often used in the research community for preliminary or feasibility tests. The different classes in this case consist of the capital letters A, E, F, H, I, K, L, M, N, T, V, W, X, Y and Z. Examples of some of the letters can be seen in Figures 4.1, 4.2 and 4.3.

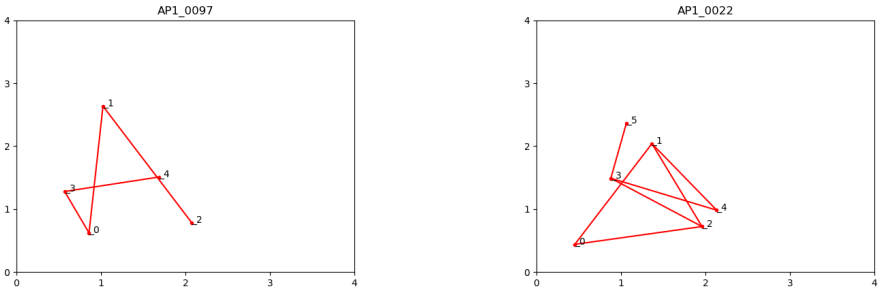


Figure 4.1: Examples of original graphs from class A.

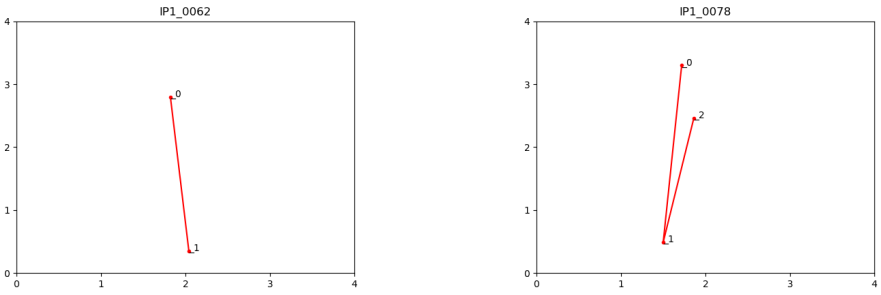


Figure 4.2: Examples of original graphs from class I.

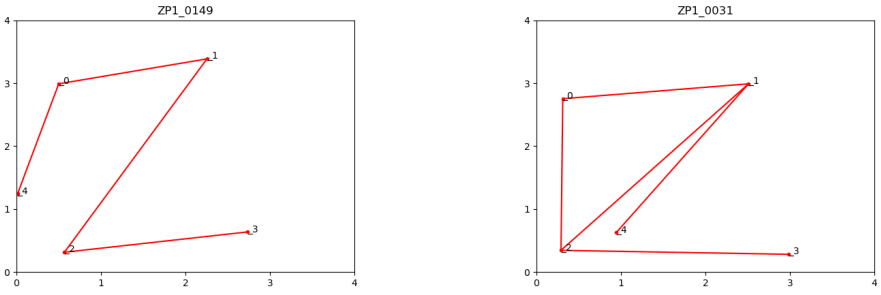


Figure 4.3: Examples of original graphs from class Z.

As we have seen in the previous chapter, while constructing matching graphs there are a few things to consider. First, the costs of insertion and deletion when creating the edit paths. Matching graphs for each class have been constructed with costs 0.2, 0.3, 0.4, 0.6, 0.9 and 1.6 respectively. This has been done because the costs can heavily influence the size of the resulting matching graphs. But for the remainder of this section we will only distinguish between low cost and high cost matching graphs, in order to respect the scope of this thesis. The second thing to consider is the edge handling. As we have seen in the last chapter we have two options: pruning and non pruning. This means that we consider four different categories of matching graphs:

- low cost, non pruned
- low cost, pruned
- high cost, non pruned
- high cost, pruned

For matching graphs build with low insertion and deletion costs and non pruned edges, there is only a slight visual resemblance to the original graphs. In general they do not provide convincing visual results and therefore do not resemble our intuition of core structures very much. Examples can be seen in Figure 4.4. Matching graphs from class I form an exception throughout, since I is structurally quite a simple letter (see Figures 4.4 (c), (d)).

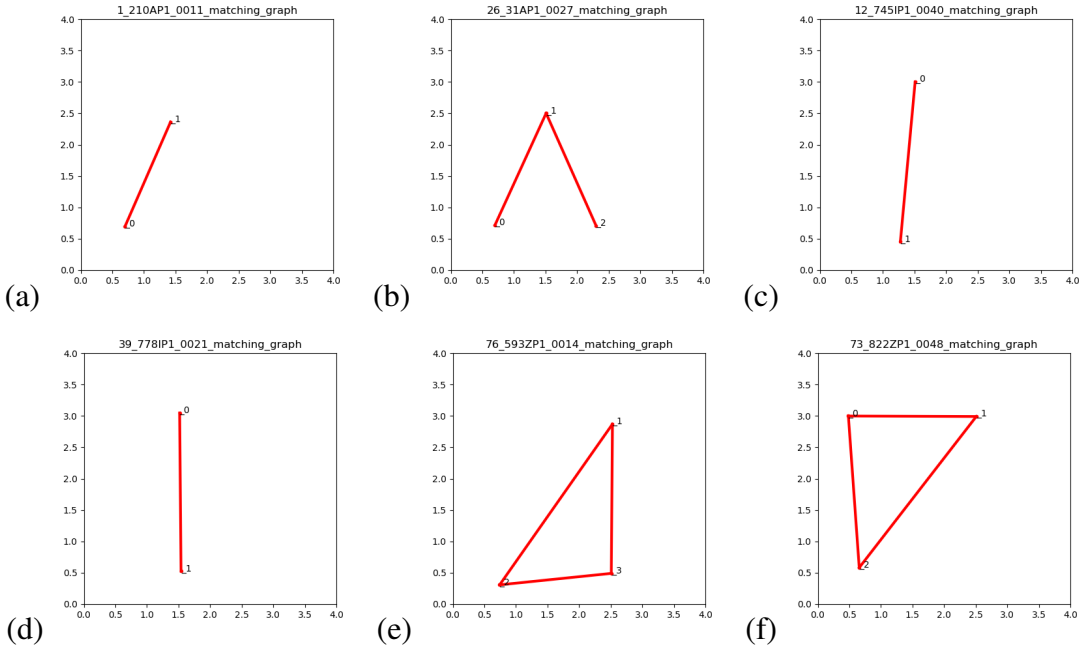


Figure 4.4: (a), (b) low cost, non pruned matching graphs from class A.
(c), (d) low cost, non pruned matching graphs from class I.
(e), (f) low cost, non pruned matching graphs from class Z.

Building matching graphs with low costs and pruned edges results in similar graphs as seen above. They only slightly resemble the original graphs. It is apparent that they are made up with less edges in general, since these matching graphs have been pruned. Examples hereof can be seen in Figure 4.5.

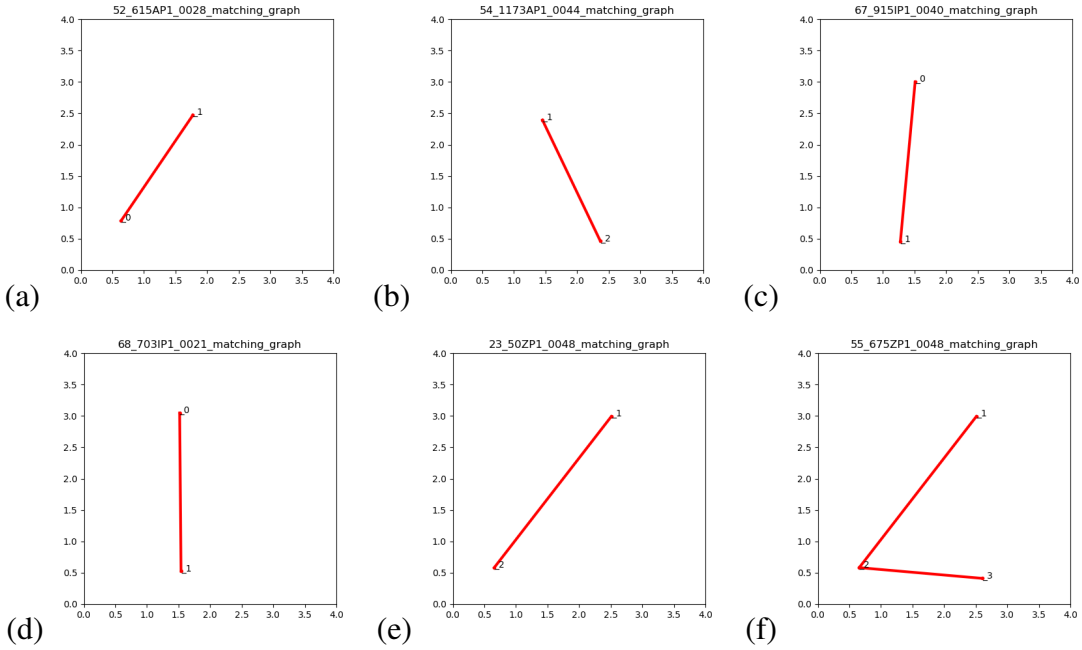


Figure 4.5: (a), (b) low cost, pruned matching graphs from class A.
(c), (d) low cost, pruned matching graphs from class I.
(e), (f) low cost, pruned matching graphs from class Z.

Matching graphs built with high costs and no pruning give a better visual result. Even though they do not exactly represent our intuition for core structures, it is possible to recognize the letters more easily than with the low cost matching graphs. Matching graphs from class I still form an exception in the sense, that the matching graphs represent the core structures pretty good because of the simplicity of the letter I. Examples can be seen in Figure 4.6.

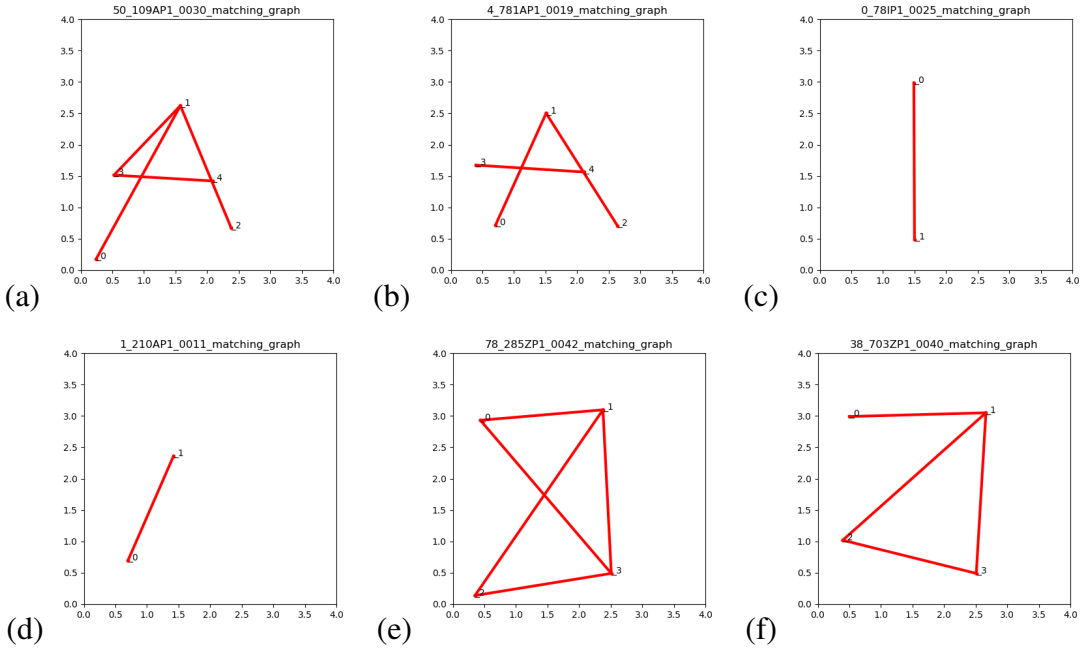


Figure 4.6: (a), (b) high cost, non pruned matching graphs from class A.
(c), (d) high cost, non pruned matching graphs from class I.
(e), (f) high cost, non pruned matching graphs from class Z.

Finally, matching graphs built with high insertion and deletion costs and pruned edges give the best visual results. They resemble our intuition of core structures very much. In comparison with the non pruned matching graphs built with high costs, we can see that they have less edges thus creating less visual distraction. Examples can be seen in Figure 4.7.

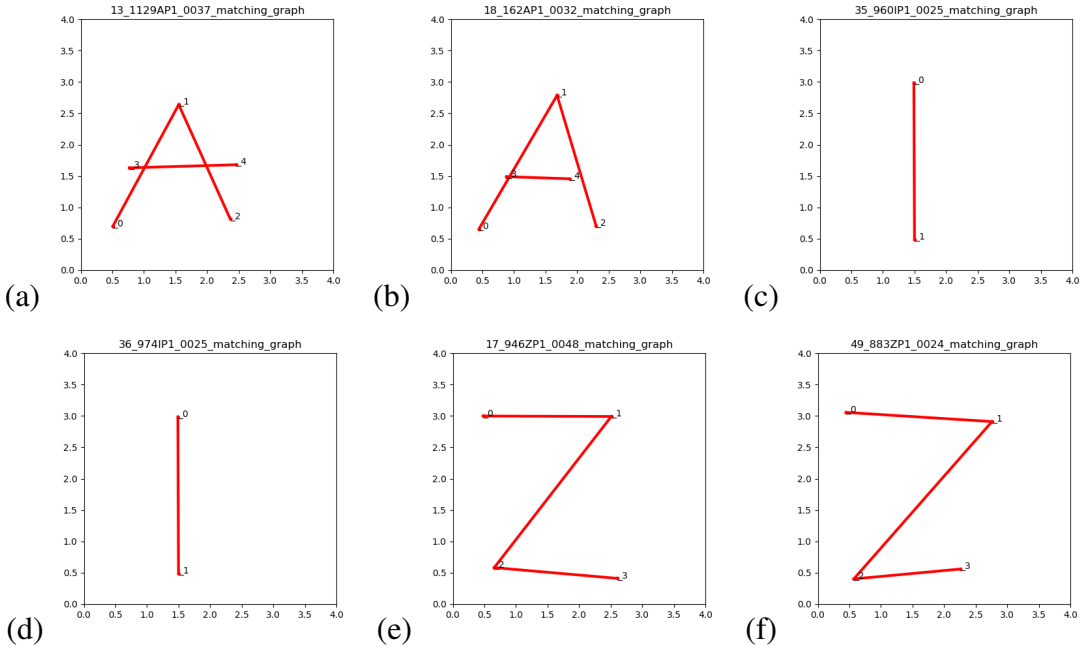


Figure 4.7: (a), (b) high cost, pruned matching graphs from class A.
(c), (d) high cost, pruned matching graphs from class I.
(e), (f) high cost, pruned matching graphs from class Z.

In conclusion we observe that for pruned matching graphs the higher the cost, the better the results. For unpruned matching graphs this trend is not as strong, since the highest cost matching graphs turned out slightly less clear. In general it can be said that pruned matching graphs resemble our intuition of a core structure more, than unpruned ones. Therefore we will focus only on pruned matching graphs in the next section.
A link to the repository with the complete data sets (and results) is available in the Appendix.

4.2 Results

In this section we discuss the outcomes of this thesis and answer the research question. Similar results for different data sets will be summarized in order to respect the scope of this thesis.

As we have seen in Chapter 2, the results of this thesis are based on the implementation of the Subgraph Isomorphism algorithm by Ullmann. The algorithm is designed to find all isomorphisms between two given graphs. As mentioned in the definition of subgraph isomorphism, in this thesis we work with graphs where the labels are coordinates.

Therefore we use the Eucledian distance between two nodes as part of determining if an isomorphism exists. By means of a visual inspection we decided to use 0.9 as constant in our heuristic (see Section 2.3). All of the following results are based on the distance threshold 0.9.

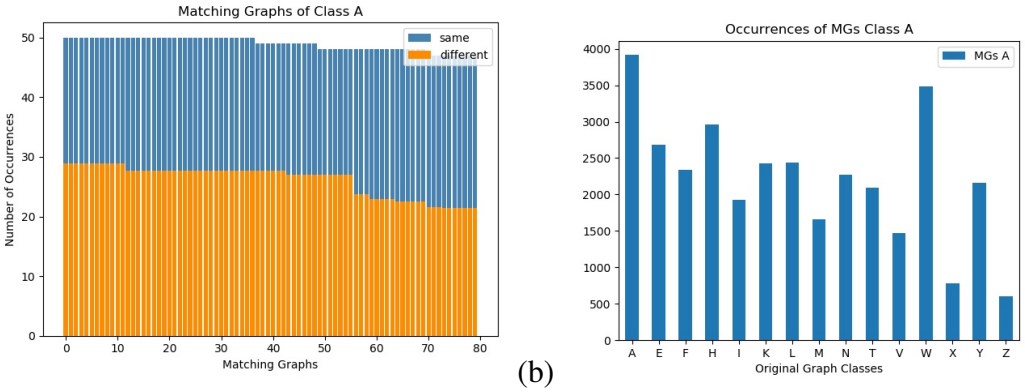
All data sets used in this thesis have 15 different classes (A-Z). So when considering the figures in order to answer the research question *"Do matching graphs created from a given class A appear more often in class A or in any of the other classes B-Z?"*, it is important to keep in mind, that the original category of "different classes" is 14 times bigger in size than the category "same class". To be able to compare the two classes more easily, the category of "different classes" has been normalized (divided by 14). Therefore it is possible that the number of occurrences in the "different classes" category is not necessarily an integer and can be in decimals. It is also important to note that for all data sets the results for the training, test and validation set were similar. Therefore all the examples are taken from the respective test sets. As noted in the last section, we will focus on pruned matching graphs only.

4.2.1 Results for Low Cost Matching Graphs

The first group of results includes all data sets with pruned matching graphs built with low insertion and deletion costs. The results show that for all the matching graphs no matter the class (A-Z), the matching graphs themselves appear more often in their own class than in different classes. It is apparent though that the matching graphs do not appear significantly more often in their own classes than in any of the other classes individually. This can be seen in the following examples.

In Figure 4.8 we see the results for the matching graphs from class A. Figure 4.8 (a) shows the number of occurrences for each matching graph in the whole set of original graphs. The x-axis represents the individual matching graphs from class A. The y-axis shows the number of times a matching graph appeared, either in original graphs from their own class (blue) or in original graphs from different classes (orange). It is important to note, that the matching graph occurrences have been sorted separately for these diagrams. That's to say, the first blue bar and the first orange bar do not necessarily represent the same matching graph. We can see that matching graphs from class A generally appear more often in the same class than in different classes. It is apparent that the number of times the matching graphs occur in their own class is 1.7 to 2.2 times higher than the number of times they occur in different classes.
Figure 4.8 (b) shows the distribution of the matching graphs from class A in all the original graph classes (A-Z). In this figure all the classes (A-Z) are listed in the x-axis and the y-axis shows the number of matching graphs from class A that have appeared in the respective classes. We can see that the matching graphs from class A appear in

all classes (A-Z). We observe that they appear slightly more often in their own class, compared to all other classes. However the number of times matching graphs form class A appear in other classes, especially classes E, F, H, K and W, is rather high as well.



(a) (b)

Figure 4.8: Occurrences of matching graphs from class A. Pruning, low costs, distance 0.9, test set.

In Figure 4.8 (a) (and in all the equivalent following figures) we can observe that there are groups of individual matching graphs that occur the same number of times in their own class and approximately the same number of times in the different classes category. As we have seen in chapter 3 the creation of matching graphs is based on the graph edit distance and therefore on the edit paths. All the matching graphs from one grouping are based on the same source graph in the edit path. The reason for this lies in the way of how we construct the matching graphs in the first place. Since the mapping in the edit path goes from source graph to target graph and we adopt the edges from the source graph, it is not surprising that the resulting matching graph strongly resembles it's source graph. Therefore the resemblance between all matching graphs with the same source graph is to be expected. As an example Figure 4.9 shows two matching graphs from class A built from the same source graph.

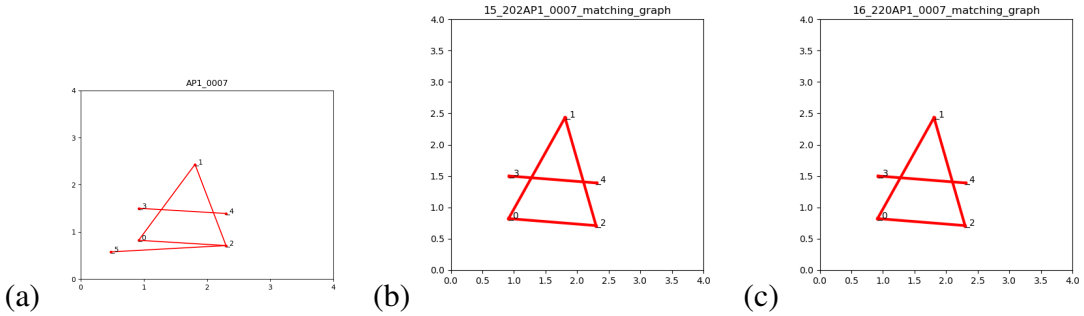


Figure 4.9: (a) Original graph from class A.
(b), (c) Two different matching graphs from class A built from the same source graph.

Next we consider the matching graphs from class I. As can be seen in Figure 4.10 (a) the matching graphs from class I also appear more often in their own class than in other classes. In this case we can see that the number of times the matching graphs occur in their own class is between 1.7 and 1.9 times higher than the number of times they appear in different classes. Figure 4.10 (b) shows that the matching graphs from class I appear the most in their own class, followed closely by occurrences in classes H, T and V. This is not surprising considering the shape of the letter I. It is to be expected that matching graphs from class I also appear in original graphs from other classes with similar structure. It is important to note that the results might be improved by changing the distance threshold used in the algorithm by Ullmann.

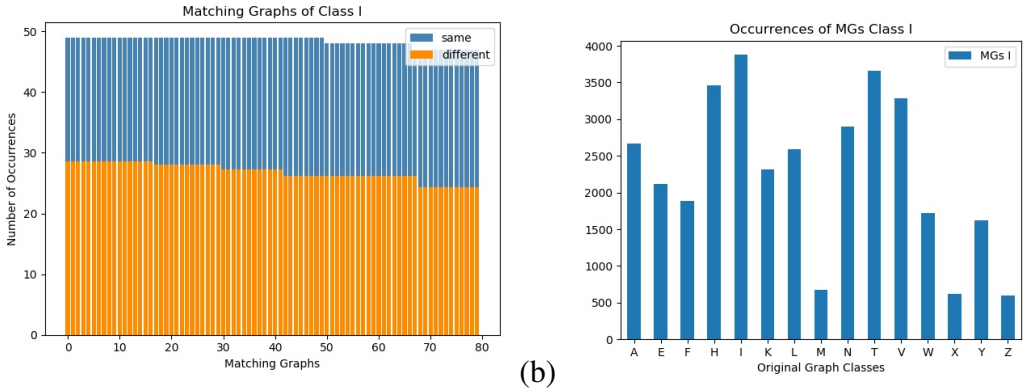
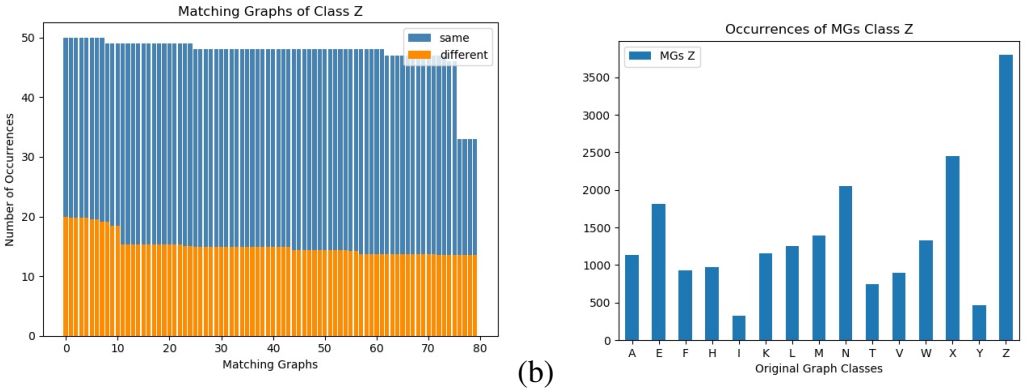


Figure 4.10: Occurrences of matching graphs from class I. Pruning, low costs, distance 0.9, test set.

Our last example for the low cost, pruned matching graphs are matching graphs from class Z. Figure 4.11 (a) shows that the matching graphs from class Z appear more often in their own class compared to different classes. We can see that the number of times

the matching graphs appear in their own class is between 2.3 and 2.5 times higher than the number of times they occur in different classes. In Figure 4.11 (b) we see that the matching graphs appear most often in their own class. However the number of times they appear in classes E, N and X is high as well.



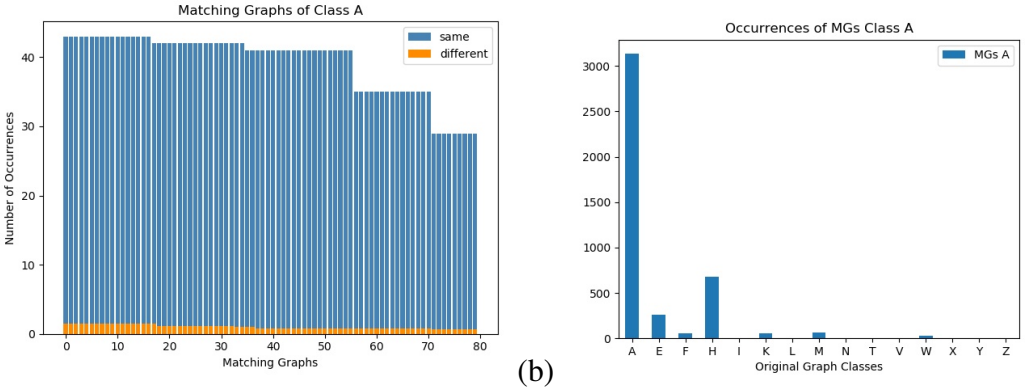
(a) (b)

Figure 4.11: Occurrences of matching graphs from class Z. Pruning, low costs, distance 0.9, test set.

4.2.2 Results for High Cost Matching Graphs

The second group of results contains all data sets with pruned matching graphs built with high insertion and deletion costs. Here we can observe that matching graphs appear considerably more often in their own class than in different classes compared to the results we have seen in the previous chapter. Examples follow below:

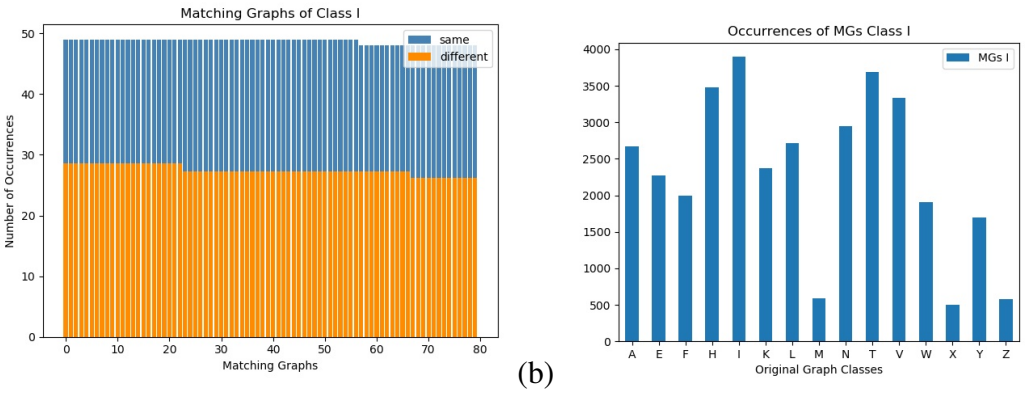
First, we consider the matching graphs from class A. They appear notably more often in their own class than in different classes. In Figure 4.12 (a) we can see that the number of times matching graphs from class A occur in their own class is between 29 and 41 times higher than the number of times they occur in different classes. Figure 4.12 (b) shows that almost all occurrences of the matching graphs are in class A, with a few being found in classes E, F, H, K, M and W. Here we can see that there exist classes, in which the matching graphs from class A do not appear at all, e.g. classes I, L and N.



(a) (b)

Figure 4.12: Occurrences of matching graphs from class A. Pruning, high costs, distance 0.9, test set.

Matching graphs from class I form somewhat an exception. In Figure 4.13 (a) we see that they do appear more often in their own class compared to different classes. However, similar to what we have seen in the previous section for class I, the number of times the matching graphs appear in their own class is only 1.6 to 1.8 times higher than the number of times the matching graphs appear in different classes. Similarly we observe in Figure 4.13 (b) that the matching graphs from class I appear most often in their own class, closely followed by occurrences in classes F, T and V. As we have seen with the low cost matching graphs from class I, this can be explained by the fact that I is structurally such a simple letter. Therefore it is to be expected that it occurs in other classes as well. It is important to note that here too the results might be improved by adapting the distance threshold used in the algorithm by Ullmann.

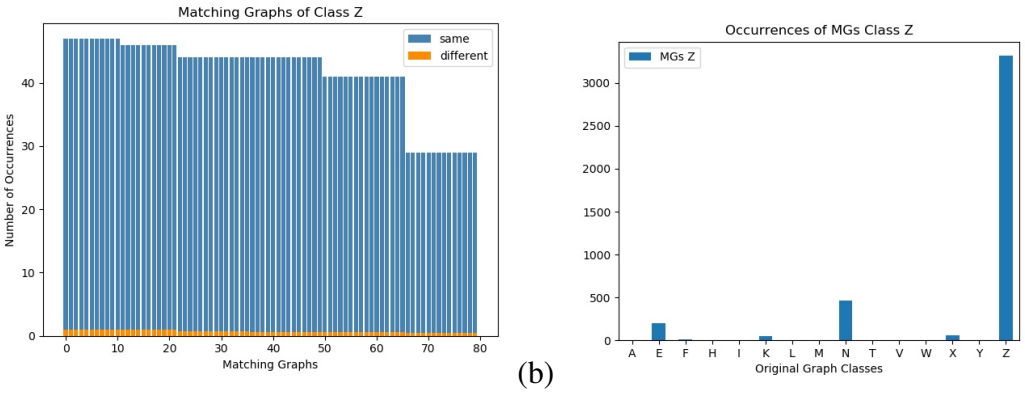


(a) (b)

Figure 4.13: Occurrences of matching graphs from class I. Pruning, high costs, distance 0.9, test set.

The last example are the matching graphs from class Z. Figure 4.14 (a) shows, that

the matching graphs appear more often in their own class than in different classes. The number of times the matching graphs appear in their own class is between 52 to 72.5 times higher compared to the number of times they occur in different classes. As can be seen in Figure 4.14 (b) most of the occurrences are indeed in the original graphs from class Z. However there are some appearances of matching graphs from class Z in classes E, K, N and X.



(a) (b)

Figure 4.14: Occurrences of matching graphs from class Z. Pruning, high costs, distance 0.9, test set.

4.2.3 Conclusions

All the matching graphs appear more often in their own class compared to different classes. However this is much more notable when considering matching graphs built with higher costs. The number of occurrences from the high cost matching graphs in their own class is up to 70 times higher that the number of occurrences in different classes. Compared to the low cost matching graphs, where the number of times the matching graphs appear in their own class is only between 1.7 and 2.5 times higher than the appearances in different classes.
In conclusion, consistent with what we have seen in chapter 3, pruned matching graphs generated with higher insertion and deletion costs generally perform better than pruned matching graphs, which have been built with lower costs. There are a few exceptions: matching graphs from classes I, V and sometimes L perform similarly, no matter the cost or the edge handling. But changing the distance threshold in the Ullmann algorithm might influence these results.

5
Conclusions and Future Work

Machine learning is the study of algorithms which improve automatically with experience. We have seen examples ranging from handwriting recognition to biometric pattern recognition and applications in chemistry. The focus in this thesis lies on pattern recognition. We have seen two different approaches: the statistical approach, which uses feature vectors and the structural approach, which focuses on symbolic data structures like for example graphs. In this thesis we focus on graph matching, which is part of the structural pattern recognition. Graph matching describes the process of finding similarities between two graphs. Based on a recent paper by M. Fuchs and K. Riesen [10] we have performed an analysis on a newly proposed data structure, the matching graphs. These matching graphs can store the information on similarity between two given graphs. They are being based on the concept of graph edit distance and can be used to solve graph classification problems. The analysis has been performed using the Subgraph Isomorphism algorithm by Ullmann [23]. While considering the research question

"Do matching graphs created from a given class A appear more often in class A or in any of the other classes B-Z?"

we were able to demonstrate, that all considered matching graphs appear more often in their own class, than in different classes. However, matching graphs generated with higher insertion and deletion costs generally perform better than matching graphs with lower costs. A few exceptions are matching graphs from classes I, V and sometimes L, where the costs did not notably influence the results.

Future work consists of the following:

- Rerun the *Algorithm for Subgraph Isomorphism* by Ullmann with a different distance threshold on the same data sets.
- Perform the analysis by running the *Algorithm for Subgraph Isomorphism* by Ullmann on different data sets.
- Identify which individual matching graphs perform best with a preliminary selection.

Acknowledgement

First and foremost I would like to thank Mathias Fuchs for being such an awesome supervisor. He always took time out of his busy schedule to answer my questions and give tips on how to improve my coding skills. His advice and patience during the project was invaluable. I would also like to thank PD Dr. Kaspar Riesen for the opportunity to write my thesis in Graph Theory.

Bibliography

[1] K. Borgwardt B. Rieck, C. Bock. A persistent weisfeiler-lehman procedure for graph classification. pages 5448–5458, 2019.

[2] S. Wold B. R.Kowalski. 31 pattern recognition in chemitry. *Handbook of Statistics*, 2:673–697, 1982.

[3] H. Bunke B. T. Messmer. Efficient subgraph isomorphism detection: A decomposition approach. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):307–323, 2000.

[4] Wikipedia contributors. Depth-first search. `https://en.wikipedia.org/w/index.php?title=Depth-first_search&oldid=999600850`. Online; accessed 08-January-2021.

[5] C. Sansone M. Vento D. Conte, P. Foggia. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298, 2004.

[6] R. Plamondon F. Leclerc. Automatic signature verification: the state of the art - 1989-1993. *International Journal of Pattern Recognition and Artificial Intelligence*, 8(3):643–660, 1994.

[7] P. Bane J. Kudase. A brief study of graph data structure. *International Journal of Advanced Research in Computer and Communication Engineering*, 5(6), 2016.

[8] R. A. Elschlager M. A. Fischler. The representation and matching of pictoral structures. *IEEE Transactions on Computers*, 22(1):67–92, 1973.

[9] M. Lozano E. Hancock M. Curado, F. Escolano. Early detection of alzheimer's disease: Detecting asymmetries with a return random walk link predictor, 2020.

[10] K. Riesen M. Fuchs. Matching of matching-graphs - a novel approach for graph classification, 2020.

[11] J. Kjell M. Kuhn. *Applied Predictive Modeling*. Springer, 2013.

[12] S. Venkatesh M. Lazarescu, H. Bunke. Graph matching: Fast candidate elimination using machine learning techniques. *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 236–245, 2000.

[13] B. D. McKay. Practical graph isomorphism, 1981.

[14] T. Mitchell. The discipline of machine learning, 2006.

[15] T. Matsumoto I. Yoshimura M. Yoshimura O. Henniger, D. Muramatsu. *Signature Recognition*. Springer, Boston, MA., 2009.

[16] Members of IAPR. Conference schedule. `https://iapr.org/conferences/`. Online; accessed 05-January-2021.

[17] Contributers of Techopedia. Handwriting recognition (hwr). `https://www.techopedia.com/definition/196/handwriting-recognition-hwr`. Online; accessed 04-January-2021.

[18] K. Riesen. *Structural Pattern Recognition with Graph Edit Distance*. Springer, 2015.

[19] K. Riesen. Structural pattern recognition, 2017.

[20] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[21] M. Santhi S. Hariprasath. Biomodal boimetric pattern recognition system based on fusion of iris and palprint using multi-resolution approach, 2019.

[22] Y. W. Tsay S. L. Peng. Comparison of protein quaternary structures by graph approaches. In *Pattern Recognition in Computational Molecular Biology: Techniques and Approaches*. John Wiley Sons, Inc., 2016.

[23] J. R. Ullmann. An algorithm for subgraph isomorphism, 1976.

[24] K. S. Fu W. H. Tsai. Error-correcting isomorphisms of attributed relational graphs for pattern analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(12):757–768, 1979.

A
Appendix

A

A.1 Refinement Procedure

Refinement Procedure described in *An Algorithm for Subgraph Isomorphism* by J. R. Ullmann [23].

```
Step 1   elim := 0,
         i := 1;
Step 2   k := 1,
         sc := 2^(p_α − 1);
         h := 1;
Step 3.  if sc & A_i = 0 then go to step 4;
         lst_k = h,
         k := k + 1;
Step 4.  sc := sc × 2^−1,
         h := h + 1;
         if k ≠ deg_i + 1 then go to step 3;
Step 5.  j = 1,
         sc = 2^(p_β − 1),
Step 6.  if M_i & sc = 0 then go to step 9;
         h := 1,
Step 7.  x := lst_h ,
         if M_x & B_j = 0 then go to step 8;
         h = h + 1;
         if h ≠ deg_i + 1 then go to step 7 else go to step 9;
Step 8.  M_i = M_i & NOT sc,
         elim := elim + 1;
         h := h + 1;
Step 9.  sc := sc × 2^−1;
         j = j + 1;
         if j ≠ p_β + 1 then go to step 6;
Step 10   if M_i = 0 then go to FAIL exit;
          i = i + 1,
          if i ≠ p_α + 1 then go to step 2;
          if elim ≠ 0 then go to step 1;
          go to SUCCEED exit;
```

Figure A.1: Refinemen Procedure

A.2 Repository

Link to the repository: https://github.com/v-WU/BachelorThesis.git