INTERNATIONAL MASTER OF RESEARCH IN
COMPUTER SCIENCE: COMPUTER AIDED DECISION
SUPPORT

Graph for Pattern Recognition

Author:
Romain Raveaux
Zeina ABU-AISHEH

in the
RFAI groups
at the University of Tours

October 2013
Abstract

Attributed graphs are powerful data structures for the representation of complex entities. In a graph-based representation, vertices and their attributes describe objects (or part of objects) while edges represent interrelationships between the objects. Due to the inherent genericity of graph-based representations, and thanks to the improvement of computer capacities, structural representations have become more and more popular in the field of Pattern Recognition (PR). PR problems can take advantage of graph in two ways:

- through Graph Matching
- through Graph Embedding

Keywords Graph matching, Graph embedding, Pattern Recognition
## Contents

Abstract

List of Figures and Tables

List of Tables

Abbreviations

Symbols

1 Introduction

1.1 Introduction to Graphs

1.1.1 Graph Definition

1.1.2 Graph Categories

1.1.2.1 Directed and Undirected Graphs

1.1.2.2 Attributed and Non-attributed Graphs

1.1.2.3 Attributed Subgraph

1.1.2.4 Special Cases

Planar Graph

Weighted Graph

Directed Acyclic Graph

Bipartite Graph

Labelled Graph

1.2 Graph Matching Problems

1.2.1 Exact Matching

1.2.1.1 Graph isomorphism

1.2.1.2 Induced Subgraph Isomorphism (SGI)

1.2.1.3 Monomorphism

1.2.1.4 Maximum Common Subgraph (MCS)

1.2.1.5 Complexity of exact matching problems

1.2.2 Inexact Graph Matching (IGM)

1.2.2.1 Problem transformation : from inexact to exact

1.2.2.2 Substitution-Tolerant Subgraph Isomorphism

1.2.2.3 Inexact Subgraph Isomorphism

1.2.2.4 Inexact Graph Isomorphism
1.2.3 Multivalent Matching ................................................. 11
1.3 Graph Matching for Pattern Recognition Problems ................. 11
  1.3.1 Full graph search ......................................................... 12
  1.3.2 Subgraph search ....................................................... 12
  1.3.3 Subgraph spotting ...................................................... 12
  1.3.4 Similarity search ...................................................... 12
  1.3.5 Graph mining .......................................................... 12
1.4 Conclusion ............................................................... 13

Chapter 2 .............................................................. 13
2.1 Graph Matching in Pattern Recognition ............................... 14
  2.1.1 Graph Representation in Pattern Recognition .................. 14
  2.1.2 Inexact Graph Matching in Pattern Recognition ............... 17
       Distance and Graph Classification ............................... 18
2.2 Criteria and Synthesis ................................................. 21
2.3 Related Works .......................................................... 23
  2.3.1 Error Tolerant Methods ............................................. 23
  2.3.2 Focusing on Graph Edit Distance ................................. 24
  2.3.3 Graph Edit Distance Optimal Methods ......................... 25
       2.3.3.1 Graph Edit Distance ........................................... 25
       2.3.3.2 A-Star .......................................................... 26
  2.3.4 Graph Edit Distance sub-optimal Methods ...................... 27
       2.3.4.1 Beam-Search .................................................... 27
       2.3.4.2 Munkres ......................................................... 28
2.4 Application to graph classification .................................. 29
2.5 Scientific Deadlocks .................................................. 30
2.6 Conclusion ............................................................. 30

Bibliography .............................................................. 32
List of Figures

1.1 Graph isomorphism problem ........................................ 5
1.2 Three different types of morphisms ................................. 7
1.3 Substitution-tolerant subgraph isomorphism problem ............ 9
1.4 Inexact Subgraph Isomorphism problem ............................ 10
1.5 Inexact graph isomorphism problem ............................... 11
1.6 Graph isomorphism constraints ..................................... 12

2.1 Graph of pixels ...................................................... 15
2.2 House represented as graph of characteristic points [74] ........ 15
2.3 Region adjacency graph ............................................. 16
2.4 Combination of structural and statistical features ............... 16
2.5 Transforming $g_1$ into $g_2$ by means of edit operations. Note that nodes labels are represented in different gray scales. ............ 25
2.6 The numbers indicate in which order the nodes were expanded: the node with the lowest estimated cost $f$ among all currently known nodes is always picked first. Eventually node $n$ is reached and recognized as the optimal solution because the estimated remaining costs are 0 and no unexpanded node has a lower estimated total cost. ............................... 27
List of Tables

2.1 Comparison between Classical Graph-Matching Methods in Terms of Their Computational Complexity and the Ability to Perform an Inexact Matching 23
2.2 Comparison between Error-Tolerant Graph-Matching Methods in Terms of their Optimality, Popularity and whether if they are metric or not. . . 24
2.3 Symbols used in this chapter and their methods’ names . . . . . . . . . . 31
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG</td>
<td>Attributed Graphs</td>
</tr>
<tr>
<td>GED</td>
<td>Graph Edit Distance</td>
</tr>
<tr>
<td>BM</td>
<td>Beam Search</td>
</tr>
</tbody>
</table>

...
Symbols

\[ d \quad \text{distance} \]

\[ \ldots \]
Chapter 1

Introduction

In this chapter we present the basic definitions, establish notations and we give a brief overview about the main aspects that the report focuses on.

1.1 Introduction to Graphs

1.1.1 Graph Definition

Graphs are an efficient data structure and the most general formalism for object representation in structural pattern recognition. They are basically composed of a finite or infinite set of vertices $V$, that represents parts of objects, connected by a set of edges $E \subseteq V \times V$, that represents the relations between these two parts of objects, where each edge connects two nodes in the graph ($e=(u,v)$ such that both $u$, $v$ are nodes that belong to the set $V$).

Definition 1.1. Graph $G=(V,E)$

$V$ is a set of vertices

$E$ is a set of edges such as $E \subseteq V \times V$

The graph size $|G|$ refers to the number of nodes of the given graph $G$.

Strings and trees are considered as special cases of graphs.

1.1.1.1 Subgraph

A subgraph $\bar{G}$ is a graph whose set of vertices $\bar{E}$ and set of edges $\bar{V}$ form subsets of the sets $V$ and $E$ of the graph $G$. A subgraph $\bar{G}$ of a graph $G$ is said to be induced (or full)
if, for any pair of vertices \( v_i \) and \( v_j \) of \( \bar{G} \), \( e(v_i, v_j) \) is an edge of \( \bar{G} \) if and only if \( e(v_i, v_j) \) is an edge of \( G \). In other words, \( \bar{G} \) is an induced subgraph of \( G \) if it has exactly the edges that appear in \( G \) over the same vertex set.

**Definition 1.2. Subgraph**

\[
V(\bar{G}) \cup V(G) \\
E(\bar{G}) \cup E(G)
\]

1.1.2 Graph Categories

Graphs are divided into two main categories according their edge orientation and their attributes.

1.1.2.1 Directed and Undirected Graphs

A graph \( G \) is said to be **undirected** when all the edges \( e_i \)'s of the set \( E \) have no direction. Consequently, the edges \( e_1=(u,v) \) and \( e_2=(v,u) \) are identical as the graph \( G \) is undirected. In contrast to the **directed** graphs which respect the direction that is assigned to each edge \( e_i \). Thus, for the directed graphs \( e_1 \neq e_2 \).

1.1.2.2 Attributed and Non-attributed Graphs

Non-attributed graphs are only based on their topological structures, e.g. molecular graphs where the structural formula is considered as the representation of a chemical substance. Thus, no attributes can be found on neither the edges nor the nodes of the graph. Whereas in the attributed graphs (AG), significant attributes can be found on edges, nodes or both of them which efficiently describe objects (in terms of shape, color, coordinate, size, etc.) and their relations.

In the AGs, two extra parameters have been added(\( \mu, \zeta \)) where nodes’ labels and edges’ labels are represented successively.

Mathematically speaking, AG is considered as a set of 4 tuples \( (V, E, \mu, \zeta) \) such that:

**Definition 1.3. Attributed Graph**

\( G = (V, E, \mu, \zeta) \)

\( V \) is a set of vertices

\( E \) is a set of edges such as \( E \subseteq V \times V \)
µ : V → L_V. µ is a vertex labeling function which associates the label l_V to a vertex v_i.
ζ : E → L_E. ζ is an edge labeling function which associates the label l_E to an edge e_i.

Note that node/edge attributes can be neither continuous (in \( \mathbb{R}^n \)) or discrete values. For notational convenience directed attributed relational graphs are simply referred to as graphs in the rest of the paper.

In this work, we consider simple graphs which does not contain self-loops or multi-edges.

1.1.2.3 Attributed Subgraph

Now that we have defined the notions of subgraph 1.2 and attributed graph 1.3, we can define the notion of attributed subgraph. It relies on the definition of subgraph 1.2 with some extends to consider constraints due to attributes.

Definition 1.4. Attributed Subgraph

Given a graph G = (V,E,µ,ζ), a subgraph of G is a graph S = (V_S,E_S,µ_S,ζ_S) such that V_S ⊆ V, E_S ⊆ E, \( \forall e = (v_1,v_2) \in E_S \) \( v_1 \in V_S \), \( v_2 \in V_S \) and \( µ_S \) and \( ζ_S \) are the restrictions of \( µ \) and \( ζ \) to \( V_S \) and \( E_S \), i.e. \( µ_S(v) = µ(v) \) and \( ζ_S(e) = ζ(e) \).

1.1.2.4 Special Cases

Planar Graph A planar graph is a graph that can be embedded in the plane, i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints.

Weighted Graph A graph is a weighted graph if a number (weight) is assigned to each edge.

Directed Acyclic Graph A directed acyclic graph is a directed graph with no directed cycles, such that there is no way to start at some vertex v and follow a sequence of edges that eventually loops back to v again.

Bipartite Graph A bipartite graph is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V; that is, U and V are each independent sets.

Labelled Graph A Labelled Graph is a special case of Attributed Graph labelled with discrete values (symbolic or nominal data). It means that the label set is a finite set.
1.2 Graph Matching Problems

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. Matching methods are divided into two broad categories: the first contains exact matching methods that require a strict correspondence among the two objects being matched or at least among their subparts. The second category defines inexact matching methods, where a matching can occur even if the two graphs being compared are structurally different to some extent.

1.2.1 Exact Matching

In this type of problems and at the aim of matching two graphs, significant part of the topology together with the corresponding node and edge labels in g1 and g2 have to identical. Exact matching methods can only be applied on labelled graphs ?? or non-attributed graph.

1.2.1.1 Graph isomorphism

The mapping between the nodes of the two graphs must be edge-preserving in the sense that if two nodes in the first graph are linked by an edge, they are mapped to two nodes in the second graph that are linked by an edge as well. This condition must hold in both directions, and the mapping must be bijective. That is, a one-to-one correspondence must be found between each node of the first graph and each node of the second graph. More formally, when comparing two graphs $G = (V, E, \mu, \zeta)$ and $G' = (V', E', \mu', \zeta')$ we are looking for a bijective function $f : V \rightarrow V'$ which maps each vertex $v \in V$ onto a vertex $v' \in V'$ such that certain conditions are fulfilled:

**Definition 1.5.** Graph isomorphism

A function $f : V \rightarrow V'$ is a graph isomorphism from $G$ to $G'$ if

1. $\forall v \in V, \ \mu(v) = \mu'(f(v))$
2. $\forall e \in E$ and $\forall e' \in E', \ \zeta(e(v_i, v_j)) = \zeta'(e'(f(v_i), f(v_j)))$.
3. $\forall v' \in V', \ \mu'(v') = \mu(f^{-1}(v'))$
4. $\forall e'(v'_i, v'_j) \in E'$ exists an edge $e = (f^{-1}(v'_i), f^{-1}(v'_j)) \in E$ with $\zeta'(e') = \zeta(e)$

Figure 1.1 depicts the graph isomorphism problem.
1.2.1.2 Induced Subgraph Isomorphism (SGI)

It requires that an isomorphism holds between one of the two graphs and a node-induced subgraph of the other. More formally, when comparing two graphs \( G = (V, E, \mu, \zeta) \) and \( G' = (V', E', \mu', \zeta') \) we are looking for a function \( f : V \to V' \) which maps each vertex \( v \in V \) onto a vertex \( v' \in V' \) such that certain conditions are fulfilled:

**Definition 1.6.** Induced subgraph isomorphism

A function \( f : V \to V' \) is a subgraph isomorphism from \( G \) to \( G' \) if

1. \( \forall v \in V, \mu(v) = \mu'(f(v)) \)
2. \( \forall e \in E \) and \( \forall e' \in E', e(v_i, v_j) = e'(f(v_i), f(v_j)). \)
3. \( \forall e'(v'_i, v'_j) \in E' \cap f(V) \times f(V) \) exists an edge \( e = (f^{-1}(v'_i), f^{-1}(v'_j)) \in E \) with \( \zeta'(e') = \zeta(e) \)

In its exact formulation, the subgraph isomorphism must preserve the labelling, i.e. \( \mu(v) = \mu'(v') \) and \( \zeta(e) = \zeta'(e') \).

1.2.1.3 Monomorphism

Monomorphism is a light form of subgraph isomorphism. It drops also the condition that the mapping should be edge-preserving in both directions. It requires that each node of the first graph is mapped to a distinct node of the second one, and each edge of the first graph has a corresponding edge in the second one; the second graph, however, may have both extra nodes and extra edges.
It relies on the definition of subgraph 1.4. Using this definition, the subgraph isomorphism problem between a pattern graph $G$ and a target graph $G'$ is defined by

**Definition 1.7. Monomorphism**

An injective function $f : V \rightarrow V'$ is a subgraph isomorphism from a graph $G = (V, E, \mu, \zeta)$ to a graph $G' = (V', E', \mu', \zeta')$ if there exists a subgraph $S$ of $G'$ such that $f$ is a graph isomorphism from $G$ to $S$:

1. $\forall v \in V, f(v) = v' \in V', f^{-1}(v') = v$
2. $\forall e = (v_i, v_j) \in E$, there exists a distinct edge $e' = (f(v_i), f(v_j)) \in E'$

In its exact formulation, the subgraph isomorphism must preserve the labelling, i.e. $\mu(v) = \mu'(v')$ and $\zeta(e) = \zeta'(e')$.

### 1.2.1.4 Maximum Common Subgraph (MCS)

It maps a subgraph of the first graph to an isomorphic subgraph of the second one; since such a mapping is not uniquely defined, usually the goal of the algorithm is to find the largest subgraph for which such a mapping exists. Actually, there are two possible definitions of the problem, depending on whether node-induced subgraphs or plain subgraphs are used. In the first case, the maximality of the common subgraph is referred to the number of nodes, while in the second it is the number of edges that is maximized.

### 1.2.1.5 Complexity of exact matching problems

The matching problems mentioned above are all NP-complete except for graph isomorphism, for which it has not yet been demonstrated if it belongs to NP or not. Polynomial isomorphism algorithms have been developed for special kinds of graphs (e.g. for trees by Aho et al.1 in 1974, for planar graphs by Hopcroft and Wong68 in 1974, for bounded valence graphs by Luks97 in 1982) but no polynomial algorithms are known for the general case.

Hence, exact graph matching has exponential time complexity in the worst case. However, in many PR applications the actual computation time can be still acceptable, because of two factors: first, the kinds of graphs encountered in practice are usually different from the worst cases for the algorithms. Second, node and edge attributes can be used very often to reduce dramatically the search time. Of the above-mentioned
matching problems, exact isomorphism is very seldom used in PR, since more often than not the graphs being compared are obtained as the result of a (sometimes very complex) description process that is inevitably subject to some form of noise, and so missing or extra nodes and edges can appear, hampering the isomorphism. Subgraph isomorphism and monomorphism, instead, albeit more demanding from a computational viewpoint, can be effectively used in many contexts, and several algorithms for these problems have been proposed. Finally, the MCS problem is receiving much attention, albeit exact methods known up to now are only able to deal with graphs with a small number of nodes.

Figure 1.2 illustrates the different types of morphism. There exists a monomorphism between model A and the input graph but no subgraph isomorphism. Between model B and the input graph, there is a subgraph isomorphism but no isomorphism while model C and the input graph are isomorphic.

1.2.2 Inexact Graph Matching (IGM)

The stringent constraints imposed by exact matching are in some circumstances too rigid for the comparison of two graphs. So the matching process must be tolerant: it must accommodate the differences by relaxing, to some extent, the constraints that define the matching type. Usually, in these algorithms the matching between two nodes that do
not satisfy the edge-preservation requirements of the matching type is not forbidden. Instead, it is penalized by assigning to it a cost that may take into account other differences (e.g. among the corresponding node/edge attributes). So the algorithm must found a mapping that minimizes the matching cost. The definition of an error model is strongly application-dependent.

1.2.2.1 Problem transformation : from inexact to exact

An inexact matching is generally needed when no exact mapping between vertex and/or edge labels can be found, but when the mapping can be associated to a cost. For example, this case occurs when vertex and edge labels are numerical values (scalar or vectorial). The cost for the mapping can then be defined as the sum of the distances between label values. A first solution to tackle such problems relies on a discretization or a classification procedure to transform the numerical values into nominal/symbolic labels. The main drawback of such approaches is their sensitivity to frontier effects of the discretization or misclassification. A subsequent exact matching algorithm would then be unsuccessful. A second solution consists in using exact matching algorithms and to customize the compatibility function for pairing vertices and edges. The main drawback of such approaches is the need to define thresholds for these compatibilities. A last way consists in using an inexact matching procedure that overcomes this drawback by integrating the numerical values during the mapping search. In this case, the matching problem turns from a decision one to an optimization one.

1.2.2.2 Substitution-Tolerant Subgraph Isomorphism

This problem aims at finding a subgraph isomorphism of a pattern graph $S$ in a target graph $G$. This isomorphism only considers label substitutions and forbids vertex and edge insertion in $G$. This kind of subgraph isomorphism is often needed in pattern recognition problems when graphs are attributed with real values and no exact matching can be found between attributes due to noise. A subgraph isomorphism is said to be substitution-tolerant when the mapping does not affect the topology, i.e. each vertex and each edge of the pattern graph has a one-to-one mapping into the target graph, but when editing operations between vertex and edge labels are allowed. A substitution-tolerant mapping is generally needed when no exact mapping between vertex and/or edge labels can be found, but when the mapping can be associated to a cost. For example, this case occurs when vertex and edge labels are numerical values (scalar or vectorial) resulting from a feature extraction step as often in pattern analysis.
Definition 1.8. Substitution-Tolerant Subgraph Isomorphism
An injective function $f : V \rightarrow V'$ is a subgraph isomorphism from a graph $G = (V, E, \mu, \zeta)$ to a graph $G' = (V', E', \mu', \zeta')$ if there exists a subgraph $S$ of $G'$ such that $f$ is a graph isomorphism from $G$ to $S$:

1. $\forall v \in V, f(v) = v' \in V', f^{-1}(v') = v$
2. $\forall e = (v_i, v_j) \in E$, there exists a distinct edge $e' = (f(v_i), f(v_j)) \in E'$
3. $\mu(v) \approx \mu'(v')$ and $\zeta(e) \approx \zeta'(e')$

In pattern recognition applications, where vertices and edges are labeled with measures which may be affected by noise, a substitution-tolerant formulation which allows differences between labels of mapped vertices and edges is mandatory. However, these differences are associated to costs and one is interested in finding the mapping corresponding to the minimal global cost, if one exists. i.e. $\mu(v) \approx \mu'(v')$ and $\zeta(e) \approx \zeta'(e')$.

Figure 1.3 depicts the substitution-tolerant subgraph isomorphism problem.

1.2.2.3 Inexact Subgraph Isomorphism

It requires that each node/edge of the first graph is mapped to a distinct node/edge of the second graph or to a dummy node/edge. This dummy element can absorb structural modifications between the two graphs. It drops also the condition that the penalty costs should take into account extra nodes and extra edges of the second graph.

Definition 1.9. Inexact Subgraph Isomorphism
An injective function $f : V \rightarrow V'$ is a subgraph isomorphism from a graph $G = (V, E, \mu, \zeta)$ to a graph $G' = (V', E', \mu', \zeta')$ if there exists a subgraph $S$ of $G'$ such that $f$ is a graph isomorphism from $G$ to $S$:

1. $\forall v \in V, f(v) = v' \in V', f^{-1}(v') = v$
2. $\forall e = (v_i, v_j) \in E$, there exists a distinct edge $e' = (f(v_i), f(v_j)) \in E'$
3. $\mu(v) \approx \mu'(v')$ and $\zeta(e) \approx \zeta'(e')$
1.2.2.4 Inexact Graph Isomorphism

A significant number of inexact graph matching algorithms base the definition of the matching cost on an explicit model of the errors (deformations) that may occur (i.e. missing nodes, etc.), assigning a possibly different cost to each kind of error. These algorithms are often denoted as error-correcting or error-tolerant.

**Definition 1.10.** Inexact Graph Isomorphism

A bijective function $f : V \rightarrow V'$ is an inexact graph isomorphism from a graph $G = (V, E, \mu, \zeta)$ to a graph $G' = (V', E', \mu', \zeta')$ such that certain conditions are fulfilled:

1. $\Delta_v'$ is a set of dummy nodes
2. $\Delta_e'$ is a set of dummy edges
3. $\forall v \in V, f(v) = v' \in \{V' \cup \Delta_v'\}$
4. $\forall e = (v_i, v_j) \in E$, there exists a distinct edge $e' = (f(v_i), f(v_j)) \in \{E' \cup \Delta_e'\}$
5. $\mu(v) \approx \mu'(v')$ and $\zeta(e) \approx \zeta'(e')$

Figure 1.4 depicts the inexact subgraph isomorphism problem.

$$(V, E, \mu, \zeta)$$ to a graph $G' = (V', E', \mu', \zeta')$ if there exists a subgraph $S$ of $G'$ such that $f$ is a graph isomorphism from $G$ to $S$:

1. $\Delta_v'$ is a set of dummy nodes
2. $\Delta_e'$ is a set of dummy edges
3. $\forall v \in V, f(v) = v' \in \{V' \cup \Delta_v'\}$
4. $\forall e = (v_i, v_j) \in E$, there exists a distinct edge $e' = (f(v_i), f(v_j)) \in \{E' \cup \Delta_e'\}$
5. $\mu(v) \approx \mu'(v')$ and $\zeta(e) \approx \zeta'(e')$
4. $\forall e = (v_i, v_j) \in \{E \cup \Delta_e\}$, there exists a distinct edge $e' = (f(v_i), f(v_j)) \in \{E' \cup \Delta'_e\}$

5. $\forall e' = (v'_i, v'_j) \in \{E' \cup \Delta'_e\}$, there exists a distinct edge $e = (f^{-1}(v'_i), f^{-1}(v'_j)) \in \{E \cup \Delta_e\}$

6. $\mu(v) \approx \mu'(v')$ and $\zeta(e) \approx \zeta'(e')$

Figure 1.5 depicts the inexact graph isomorphism problem.

### 1.2.3 Multivalent Matching

Multivalent matching drops the condition that nodes in the first graph are to be mapped to distinct nodes of the other; hence, the correspondence can be many-to-one. Note multivalent matching can be whether exact or inexact. The possible kind of multivalent mapping are the following:

1. One to many
2. Many to one
3. Many to many

Figure 1.6 depicts the level of constraint of each graph isomorphism problem.

### 1.3 Graph Matching for Pattern Recognition Problems

Given a graph database consisting of $n$ graphs, $D = g_1, g_2, ..., g_n$, and a query graph $q$, almost all existing algorithms of processing graph search can be classified into the
following four categories: Full graph search, Subgraph search, Similarity search and
Graph mining.

1.3.1 Full graph search

Find all graphs $g_i$ in $D$ s.t. $g_i$ is the same as $q$.

1.3.2 Subgraph search

Find all graphs $g_i$ in D containing $q$ or contained by $q$.

1.3.3 Subgraph spotting

Find all occurrences of $q$ in D.

1.3.4 Similarity search

Find all graphs $g_i$ in $D$ s.t. $g_i$ is similar to $q$ within a user-specified threshold based
on some similarity measures. Similarity search is a generalization of the graph and
subgraph search.

1.3.5 Graph mining

Graph mining problem gathers similar graph or subgraph of $D$ in order to find clusters
or prototypes. No query is provided by the user.
1.4 Conclusion

This report deals with the inexact graph matching problem in Pattern Recognition context.

In this chapter, we have defined the two main families of graph matching: exact and inexact graph matching families. We have also given an overview about three types of graphs: non-attributed, labeled and continuous attributed graphs.

GI, SGI, monomorphism and MCS problems are restricted to non-attributed and labeled graphs. The scope of inexact morphisms is less limited and so graphs labelled with continuous attributes can be involved into the matching procedure. Cost penalties are assigned to approximately matched nodes and edges in order to consider structure and attribute differences.

The organization of this report is in three chapters:

Chapter 2 A literature review is presented in order to trace its evolution. An overview of several exact and inexact graph matching techniques is provided. This chapter ends up by addressing the scientific deadlocks that the thesis handles in order to come up with some feasible solutions.
Chapter 2

State of the Art

2.1 Graph Matching in Pattern Recognition

2.1.1 Graph Representation in Pattern Recognition

Graph matching is considered as a fundamental problem and an efficient data structure for object representation in PR. Patterns can be represented by graphs where the problem of object detection can be easily transformed into a graph matching problem. It is possible to identify at least 3 application areas where graph matching techniques have been successfully used. They are:

1. Biomedical and biological applications
2. Document processing
3. Image/Video analysis

For instance, image content can be represented as a graph in different ways. Here are listed five well-known representations in the image recognition topic:

1. Interest point graph
2. Region adjacency graph
3. Neighbourhood graph
4. Spatial relationship graph
5. Skeleton graph
In this section, we show the most common graph representation methods in PR.

Images can be depicted as pixels where each pixel represents a node and the adjacent nodes are connected by edges. Figure 2.1 illustrates an example of graph of pixels:

An extension of the previous graph representation method has been come up in order to get rid of many pixels on the graph, an extraction of some characteristic points can be done where each end point on the image is represented as a vertex on the graph. Figure 2.2 shows an example of graph of characteristic points.

Region adjacency graph is another way of representing images where the image is segmented into a certain number of regions, each region represents its pixels.

Figure 2.3 illustrates a square that is segmented into regions and converted to a graph afterwards.

Recently, some hybrid approaches have come up with combination of structural and statistical features as a way to represent patterns as graphs. Images are segmented into primitives which are considered as vertices of graphs and their (topological or spatial)
relationships between them as edges. Primitive length and their relative angles are associated to nodes and edges respectively. Figure 2.4 presents an example of such a hybrid method.

For a graph system to be general, one has to choose the graph representation method before extracting features.

In PR, the attributes of both nodes and edges play an important role for representing graphs. The choice of the way of representing such attributes influences the performance of the representation system and the graph matching technique used to match those structural attributed graphs.

For the used graphs in PR, combining both symbolic and numeric attributes on nodes and edges makes the extracted nodes and edges more meaningful and highly representative. Unlike the other graphs used in other fields (e.g. shortest path) where the combination of symbolic and numeric attributes is not necessarily needed.
One of the great interests of graph matching in PR is to build a Graph Database Repository for the community where a collection of various graphs which are deformed by noise is included in addition to involving thousands of graphs with different sizes (from tens of nodes to hundreds of nodes). IAM [62] and SESYD [18] are examples of such repositories.

For the performance evaluation of our proposed graph matching techniques, we focus on the repository IAM for five reasons. First, it has a variety of graphs extracted from different fields (e.g. letters and symbols), images, fingerprints, molecules, and proteins). Second, each dataset of this repository includes thousands of graphs that are affected by deformation and noise which is an important interest for error-tolerant graph matching problems. Third, the diversity of graphs' sizes where the smallest graphs are involved in the letter dataset (i.e. graphs of 8 nodes) and the largest graphs are included in the website dataset (i.e. graphs of 834 nodes). Fourth, important information is provided with each data set (e.g. number of available graphs, number of graphs categories, maximum number of nodes, maximum number of edges and edge probability). Fifth, it was used as an evaluation repository in [63] with which our proposed methods are compared. For further information about IAM, see sub-section ?? and appendix ??.

The challenging part in PR is to find a scalable inexact graph matching method that can match large attributed graphs (i.e. attributed graphs that hold thousands of nodes and edges).

Pattern Recognition is a special case study for the graph matching problems. It stands out from the crowd (computer vision, cybernetics, chemistry, text processing ...) by the very particular data under concerned. Here are three main graph features:

1. Graph size (do not exceed thousand nodes)
2. Attribute diversity (mixed of symbolic, vectorial, continuous, ...)
3. Graph topology (often planar or poorly connected)

2.1.2 Inexact Graph Matching in Pattern Recognition

[10] has divided the literature of graph matching into two periods: (the classical period and the modern period).

The aim of the classical period was to transfer mathematical models from the vector space to the graph domain. This period consists of three main big families: graph distance, median graph and weighted graphs.
In the other period, referred to as the modern period, instead of working in the graph domain, the graphs are mapped into a vector or a dot product space, such a transformation is referred to as graph embedding. As a result of this transformation, all the operations are carried out in the vector space, rather than the graph domain. This period is modern as classification tools, applied for vectors or dot product spaces, can be also applicable on graphs thanks to the transformation provided from the graph domain to the vector or dot space.

In this report, we focus on the classical period, specifically on the first family (i.e. graph distance).

**Distance and Graph Classification** In the literature, procedures have been proposed for evaluating the similarity of two graphs. Such a task, referred to as graph matching 1.2, has been drastically studied in Pattern Recognition, aiming at finding the similarity or dissimilarity of graphs [14]. Based on this similarity/dissimilarity, a distance score can be calculated by mapping (sub)structures of one graph to similar (sub)structures of another graph.

Restricting applications to exact graph matching is obviously not recommended. In reality, objects suffer from the presence of both noise and distortions, due to the graph extraction process, and thus exact graph matching algorithms fail to answer whether two graphs $g_1$ and $g_2$ are, not identical, but similar. In addition, when describing non-discrete properties of an object, nodes and edges of the source graph are labeled using continuous labels (i.e. $L \subseteq \mathbb{R}$). Such objects (i.e. with non-discrete labels) are likely to be nonidentical. For these mentioned arguments, we focus on inexact graph matching methods taking into account their applicability in the real world.

Inexact graph matching techniques have been widely proposed in the literature. The inexact graph matching, aims at relaxing, to some extent, the constraints of the exact matching process in such a way that a similarity answer/score is given for matching an attributed source graph with an attributed target graph while penalizing the structure of one or both them.

Graph Edit Distance is one of the famous error-tolerant approaches that has been firstly reported in [77] where a search tree is constructed in order to find the best path (i.e. mappings) allowing to find the best error-correcting graph isomorphism. In [77], label substitution was the only allowed operation for matching graphs which restricted their framework to graphs that are structurally identical and only have differences or variance in their associated labels. Later, in the enhanced versions of Graph Edit Distance [76], two operations were added: deletions and insertions of nodes and edges, in addition to
substitutions operation, which allows both graphs to have both different structures and labels. Graph Edit Distance will be detailed in 2.3.3.1.

Graph Edit Distance is computationally complex or expensive, it is said to be an NP-hard problem where its complexity is exponential in the number of nodes of the involved graphs, and thus such a fact limits the graph distances algorithm to work on relatively small graphs only. Using heuristic methods, like A* [57], can prune off the underlying search tree by decreasing the number of explored nodes and edges. However, the problem is still known to be NP-hard. In order to overcome this drawback, one has to think about an approximated or a sub-optimal version of graph matching which finds a local minimum rather than a global one [14]. However, while these algorithms do not guarantee to find an optimal solution of a given graph matching problem, their proposed solutions are not often far from the optimal ones. Such algorithms are extremely faster than the optimal ones (i.e. can be run in polynomial time). Approximated versions were reported in [25]; citeBoeresRB04; and [69]. Those authors have come up with local optimization criteria which cope with the computational complexity of graph matching. [56] has proposed a sub-optimal version of A*.

Moreover, another approximate graph edit distance is proposed in [63] which solves graph matching by means of bipartite graph matching. Other possibility for speeding up the computation of graph edit distance has been reported in [20], its basic idea is to find an optimal match between the sets of subgraphs by means of dynamic programming.

A linear programming formulation of Graph Edit Distance has been reported in [37], such a formulation works on unlabeled graphs.

**Genetic algorithms.** The formulation of complex graph matching problems as combinatorial optimization has been proposed in the literature. Genetic algorithms are considered as examples of such a formulation.

Matchings are formalized as states (chromosomes) of a search space with a corresponding fitness in [2], [83]; [67]; and [71]. Genetic algorithms start with an initial pool of chromosomes, considered as matchings, which evolves iteratively into other generations of matchings. Such algorithms are random in the sense that they contain randomized operations where the pool of node-to node correspondences is iteratively and randomly modified. Hence, the search space is randomly explored. Despite the randomness, two advantages are recorded for genetic algorithms. First, they are able to efficiently overcome the problem of both huge search spaces and the local minima proposed for approximating Graph Edit Distance. Second, during the randomized steps, low cost matchings are prioritized which guarantees to have, though not optimal, low cost matchings.
Relaxation labeling. Formalization by means of relaxation labeling is another type of graph matching formalization that has been proposed in the literature. The very first work has been proposed in [25]. Labels of target graphs are presented as a discrete set, each node of the source graph is assigned to one label of the target graph set. The selection of a label candidate is done using Gaussian Probability Distribution. The node-matching procedure is iteratively conducted. In the beginning, an initial labeling is selected which is dependent on the node attributes, node connectivity, etc. Afterwards, the labeling is enhanced until a satisfying labeling is found. Such a satisfying labeling is considered as the best matching of two graphs.

Some enhancements have been come up in the relaxation labeling domain. [32]; and [39] are the first works applying probability theory to graph matching where an iterative approach is conducted using a method called probabilistic relaxation. These works only take into account binary relations and assuming a Gaussian error and the use of binary relations is justified to be enough for fully defining the whole structure. The main drawback of the initial formulation of this technique, namely the fact that node and edge labels are used only in the initialization of the matching process. Such a drawback was overcome in [82]. A Bayesian perspective is used for both unary and binary node attributes in [82]; [31]; and [84].

In [33] this method is further improved by also taking edge labels into account in the evaluation of the consistency measure for matching hierarchical relational models. Bayesian graph [52] has been built up on the idea of probabilistic relaxation. The concept of Bayesian graph has also been successfully applied to trees [75].

Artificial neural networks. Neural networks have also extensively been applied to many graph matching problems. In [28]; and [70], neural networks are used to classify directed acyclic graphs. Another type of neural network, dedicated to acyclic graphs, was reported in [16] where acyclic graphs are projected on a vector space and then by means of euclidean distances, the similarity is measured. Neural networks, in their general way, are based on an energy minimization framework, and use some kind of Hopfeld network (e.g. clique detection method proposed by [66] and another method proposed by [72]). A neural gas network, derived from Kohonen’s Self-Organizing Maps (SOM), has been adopted in [71] where an unsupervised learning is applied to learn the correspondences of nodes of a source graph and the nodes of a target graph.

Different types of neural networks have been tested trying to find the most suitable type for each specific graph matching problem (e.g. Frontal face authentication problems [41], overlapped shape recognition [73], recognition of hand printed Chinese characters [72], and a Bayesian development implemented as a neural network architecture [13]).
EM algorithms.

Another important approach that was employed for solving inexact graph matching is the EM algorithm. [4] and [23] are examples of this approach, in which two similar EM frameworks are proposed for two different graph matching problems. The algorithm proposed in [49] includes with both graphs’ structures and their nodes/edges’ labels.

2.2 Criteria and Synthesis

We propose a survey concerning the state of the art of the graph matching problem, conceived as the most important element in the definition of inductive inference engines in graph-based pattern recognition applications. We review both methodological and algorithmic results, focusing on inexact graph matching procedures. A selection of inexact graph matching algorithms is proposed and synthetically described, aiming at explaining some significant instances of each graph matching methodology mainly considered in the technical literature. From definition established in chapter 1, we propose to categorize methods from the literature into five problem types:

1. Exact Subgraph Isomorphism (ESGI)
2. Substitution-Tolerant Subgraph Isomorphism (STSI)
3. Inexact Subgraph Isomorphism (ISGI)
4. Inexact Graph Isomorphism (IGI)
5. Multivalent Inexact Graph Isomorphism (MIGI)

In addition, each problem can be modelled exactly or in an approximated manner. Some methods from the literature reformulate the graph matching problems into simpler problems in order to solve them optimally. One may note that solving (even optimally) an approximated formulation may not lead to an optimal solution of the original problem. We consider the following formulations:

1. Approximate Formulation
   (a) Weighted bipartite graph
   (b) Neural networks
   (c) Weighted adjacency matrix
   (d) Strings
2. Exact Formulation

(a) Integer linear programming
(b) Constraint satisfaction programming
(c) Edit path
(d) ...

We analyse the methods from the literature through three criteria:

1. Optimal or sub-optimal
2. Complexity class of the methods (Polynomial or not)
3. Deterministic or not

Finally, we group the methods according to the optimization algorithms involved.

1. Optimal

(a) Algebraic algorithm (such as eigen decomposition, spectral graph theory)
(b) Mathematical solver (Integer Linear Programming, Constraint Satisfaction Programming)
(c) Tree search (Dijkstra, A*, Branch and Bound, ...)
(d) Hungarian algorithm
(e) Indexed Search

2. Sub-Optimal

(a) Meta-heuristic optimization (such as genetic algorithm, tabu search, particle swarm, ...)
(b) Tree search (Beam Search, A*, ...)
(c) Expectation-Maximization (EM)

Let us now overview the main approaches, summarized in Table 2.2. We also report some of the most representative references in the field of computer vision. See references [14], [47] for a further study.


<table>
<thead>
<tr>
<th>Name</th>
<th>Graph matching problem</th>
<th>Objective Formulation</th>
<th>Optimality</th>
<th>Complexity</th>
<th>Deterministic</th>
<th>Optimality Method</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node Assignment</td>
<td>IGI</td>
<td>Approximate (Bipartite Graph)</td>
<td>Sub-optimal</td>
<td>Polynomial</td>
<td>Deterministic</td>
<td>Hungarian algorithm [60], [65]</td>
<td></td>
</tr>
<tr>
<td>ISGI</td>
<td>IGI</td>
<td>Exact (Bipartite Graph)</td>
<td>Optimal</td>
<td>NP-Complete</td>
<td>Deterministic</td>
<td>Tree search</td>
<td>[11], [7]</td>
</tr>
<tr>
<td>ISGI ILP</td>
<td>IGI</td>
<td>Exact (Bipartite Graph)</td>
<td>Optimal</td>
<td>NP-Complete</td>
<td>Deterministic</td>
<td>Mathematical solver [9]</td>
<td></td>
</tr>
<tr>
<td>Approximated GED</td>
<td>IGI</td>
<td>Exact (Bipartite Graph)</td>
<td>Sub-optimal</td>
<td>Polynomial</td>
<td>Deterministic</td>
<td>Hungarian algorithm [60]</td>
<td></td>
</tr>
<tr>
<td>MIGI</td>
<td>IGI</td>
<td>Exact (Bipartite Graph)</td>
<td>Optimal</td>
<td>NP-Complete</td>
<td>Deterministic</td>
<td>Mathematical solver [9]</td>
<td></td>
</tr>
<tr>
<td>EM graph matching</td>
<td>IGI</td>
<td>Approximate (Bipartite matching)</td>
<td>Sub-optimal</td>
<td>Polynomial</td>
<td>Deterministic</td>
<td>EM Algorithm [1], [24]</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2.1:** Comparison between Classical Graph-Matching Methods in Terms of Their Computational Complexity and the Ability to Perform an Inexact Matching

### 2.3 Related Works

In this sub-section, we firstly review the literature of the error tolerant graph matching methods. Then, we focus on one of the famous error tolerant graph matching methods (Graph Edit Distance: GED) giving some arguments for which we have selected this method. At last, we give an overview about the optimal and the sub-optimal versions of GED with which we will compare our proposed methods.

#### 2.3.1 Error Tolerant Methods

As illustrated in chapter 1, inexact error tolerant graph matching gives a similarity measure between two graphs instead of answering whether those two graphs are identical or not. Its task is to find the minimum mapping cost between two graphs (i.e. partial mapping). In this subsection, we synthesize error-tolerant graph matching methods, presented in the literature, taking into account Optimality, Popularity and whether the method is a metric or not as criteria of synthesizing. Table 2.2 presents error-tolerant graph matching methods with their references, grouped according to their optimality.
Table 2.2: Comparison between Error-Tolerant Graph-Matching Methods in Terms of their Optimality, Popularity and whether if they are metric or not.

<table>
<thead>
<tr>
<th>Name</th>
<th>Optimality</th>
<th>Popularity</th>
<th>Metric</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>GED</td>
<td>Optimal</td>
<td>++</td>
<td>YES</td>
<td>[11]</td>
</tr>
<tr>
<td>GED ILP</td>
<td>Optimal</td>
<td>++</td>
<td>YES</td>
<td>[37]</td>
</tr>
<tr>
<td>Approx GED</td>
<td>Sub-optimal</td>
<td>++</td>
<td>NO</td>
<td>[56]</td>
</tr>
<tr>
<td>ILP</td>
<td>Optimal</td>
<td>++</td>
<td>YES</td>
<td>[48]</td>
</tr>
<tr>
<td>Indexing method</td>
<td>Optimal</td>
<td>++</td>
<td>YES</td>
<td>[59]</td>
</tr>
<tr>
<td>Neural Network</td>
<td>Sub-optimal</td>
<td>+</td>
<td>YES</td>
<td>[71]</td>
</tr>
<tr>
<td>String based methods</td>
<td>Sub-optimal</td>
<td>+</td>
<td>YES</td>
<td>[54], [5]</td>
</tr>
<tr>
<td>Spectral Theory</td>
<td>Optimal</td>
<td>++</td>
<td>YES</td>
<td>[80], [22], [79]</td>
</tr>
<tr>
<td>Node Assignment</td>
<td>Sub-optimal</td>
<td>++</td>
<td>NO</td>
<td>[62], [81]</td>
</tr>
<tr>
<td>Genetic algorithm</td>
<td>Sub-optimal</td>
<td>-</td>
<td>NO</td>
<td>[2], [27], [35]</td>
</tr>
<tr>
<td>Probabilistic Relaxation</td>
<td>Sub-optimal</td>
<td>?</td>
<td>NO</td>
<td>[62], [24], [30]</td>
</tr>
</tbody>
</table>

2.3.2 Focusing on Graph Edit Distance

In this sub-section, we give some arguments for which we have given a focus on Graph Edit Distance.

First, Graph Edit Distance \((d_{\text{plain}})\) is an error-tolerant technique that has been widely studied and largely applied to pattern recognition. Its flexibility comes from its generality as it can be applicable on unconstrained attributed graphs. Moreover, it can be dedicated to various applications by means of specific edit cost functions. Recently, a novel \(d_{\text{plain}}\) toolkit software has been designed by [64] which integrates both optimal and approximated versions of \(d_{\text{plain}}\).

Second, it has been shown by Neuhaus and Bunke [53] that \(d_{\text{plain}}\) can be a metric if each of its elementary operations satisfies the three properties of metric spaces (i.e. positivity, symmetry and triangular inequality ). \(d_{\text{plain}}\) can be turned into a metric by putting the metric constraints on the cost functions of edit operations. For graph edit distance to satisfy the triangular property any node substitution operation having higher costs than than a deletion followed by an insertion operation is replaced by the latter. However, graph edit distance has a drawback behind its complexity which is exponential in the number of vertices. Such a drawback prevents this technique to be scalable to large graphs.

Third, few research papers have discussed the direct relation between GED and MCS ([8],[7]). [7] has studied some restrictions under which GED can pass through a maximum unlabeled common subgraph of two graphs \((G_1\) and \(G_2\)). These restrictions are put on the cost of the elementary costs of edit paths where each edit operations must avoid as much as possible vertex and edge substitutions. In other words, a vertex substitution that is more expensive than the removal of this vertex followed by its insertion with a new label is replaced by the latter. Hence leading to MCS of \(G_1\) and \(G_2\) that
is constructed without any substitution. \(MCS(G_1, G_2)\) is not necessarily unique for the two graphs \(G_1\) and \(G_2\).

For all the arguments mentioned above, we focus on \(d_{plain}\) as a basis of this dissertation.

### 2.3.3 Graph Edit Distance Optimal Methods

#### 2.3.3.1 Graph Edit Distance

\(d_{plain}\) is a similarity graph matching method that is based on Dijkstra’s algorithm. The concept of graph edit distance was first reported in (Sanfeliu and Fu, 1983; Bunke and Allermann, 1983). Its basic idea of \(d_{plain}\) is to find the best series of distortions that can transform one graph into another. In other words, graph edit distance tries to find the minimum cost among all generated costs. The included edit operations are: insertion, deletion and substitution operations. These operations include both edges and nodes.

For two graphs (a model graph and a data graph), \(d_{plain}\) transforms the data graph into the model graph by performing some distortions on the data graph (i.e. inserting, deleting or substituting nodes and their corresponding edges.). The computations of \(d_{plain}\) are carried out by means of a tree search algorithm. For the case of attributed graphs, \(d_{plain}\) is computed based on the edit distance methods of strings and numbers where nodes and edges are represented as strings and numbers. In some methods (e.g [56], [55], [54] and [21]), attributes are handled on both edges and nodes while in [37] only nodes attributes are considered.

A example of an edit path between two graphs \(g_1\) and \(g_2\) is shown in Figure 2.5. The Following operations have been applied in order to transform \(g_1\) into \(g_2\): three edge deletions, one node deletion, one node insertion, two edge insertions and two node substitutions.

For further reading, we refer the reader to [29] and [86].

**Definition 2.1.** Graph Edit Distance

Let \(G_1 = (V_1, E_1, \mu_1, \zeta_1)\) be the model graph and \(G_2 = (V_2, E_2, \mu_2, \zeta_2)\) be the target graph The graph edit distance between \(G_1\) and \(G_2\) is defined by:
\[ d_{\text{plain}}(G_1, G_2) = \min_{e_1, \ldots, e_k \in EP} \sum_{i=1}^{k} c(e_i) \] (2.1)

Where \( EP \) denotes the sequence of edit operations \( e_1; \ldots; e_k \) transforming \( G_1 \) into \( G_2 \), also called an edit path between \( G_1 \) and \( G_2 \). \( EP \) can be also defined by: \( EP = \{ e_i \}_{i=1}^{k} \), and \( c \) denotes the cost function measuring the strength \( c(e_i) \) of edit operation \( e_i \).

A standard set of distortion operations, called edit path, is given by insertions, deletions and substitutions of both nodes and edges. We denote the substitution of two nodes \( u \) and \( v \) by \( (u \rightarrow v) \), the deletion of node \( u \) by \( (u \rightarrow \epsilon) \), and the insertion of node \( v \) by \( (\epsilon \rightarrow v) \). For edges (e.g. \( u \) and \( v \)), we use a similar notation:

\[
E_i = \begin{cases} 
  u \rightarrow v & \text{u-v Substitution} \\
  u \rightarrow \epsilon & \text{Deletion of node u} \\
  \epsilon \rightarrow v & \text{Insertion of node v}
\end{cases} 
\] (2.2)

\( d_{\text{plain}} \) ends up finding the optimal edit path between two graphs. However, its bottleneck comes from its computational complexity where the size of the search space increases exponentially with the number of nodes of the involved graph restricts it to work on only small graphs.

### 2.3.3.2 A-Star

A few papers have proposed some approaches that make the \( d_{\text{plain}} \) problem less complex and accordingly allow \( d_{\text{plain}} \) to efficiently work on highly large graphs. [57] proposed to use heuristics (referred to as \( d_{A^*} \)). Such a heuristic method puts a restriction on the lower bound of the exact cost which also guarantees to end up finding an optimal solution of \( d_{\text{plain}} \). The heuristic solution is a best-first search algorithm (Hart et al., 1968) which always finds an optimal solution starting from a current node \( n \) and ending with a leaf node if there exists one. In \( d_{A^*} \), the substitutions between unprocessed nodes \( n_1 \) of a Graph \( G_1 \) an unprocessed nodes \( n_2 \) of a graph \( G_2 \) is performed. In order to obtain a lower bound of the edit cost, a combination of \( \min(n_1, n_2), \max(0, n_1 - n_2) \) and \( \max(0, n_2 - n_1) \) is taken into account where \( \min(n_1, n_2) \) represents the least expensive substitution between \( n_1 \) and \( n_2 \) while \( \max(0, n_1 - n_2) \) and \( \max(0, n_2 - n_1) \) represent the costs of nodes deletion and nodes insertions, respectively. At each stage, the most promising partial path is always selected first. Such heuristics choose the least expensive cost between a substitution operation and a deletion operation followed by an insertion operation.
Chapter 2. State of the Art

Figure 2.6: The numbers indicate in which order the nodes were expanded: the node with the lowest estimated cost $f$ among all currently known nodes is always picked first. Eventually node $n$ is reached and recognized as the optimal solution because the estimated remaining costs are 0 and no unexpanded node has a lower estimated total cost.

Although this heuristic technique guarantees to find an optimal edit path, the running time remains a bottleneck.

An example of a search tree that is constructed by the $A^*$ algorithm is showed in figure 2.6.

A widely used method for edit distance computation is based on the $A^*$ algorithm 1.

2.3.4 Graph Edit Distance sub-optimal Methods

Suboptimal algorithms have been proposed by the literature in order to cope with the problem of $d_{plain}$ (i.e. the heuristic and the non-heuristic versions). The aim of these algorithms is to overcome with the problem of running time and space complexity from which $d_{plain}$ suffers.

2.3.4.1 Beam-Search

A modification of $d_{A^*}$, called Beam-Search, has been proposed by [56]. The purpose of Beam-Search (referred to as $d_{Beam(s)}$) is to prune the search tree while searching an
Algorithm 1 Graph Edit Distance Algorithm

Require: Non-empty attributed graphs $g_1 = (V_1, E_1, \mu_1, v_1)$ and $g_2 = (V_2, E_2, \mu_2, v_2)$ where $V_1 = \{u_1, ..., u_{|V_1|}\}$ and $V_2 = \{u_2, ..., u_{|V_2|}\}$

Ensure: A minimum cost edit path from $g_1$ to $g_2$ e.g. $p_{min} = \{u_1 \rightarrow v_3, u_2 \rightarrow \epsilon, ...\}$

1: initialize OPEN to the empty set
2: For each node $w \in V_2$, insert the substitution $\{u_1 \rightarrow w\}$ into OPEN
3: Insert the deletion $\{u_1 \rightarrow \epsilon\}$ into OPEN
4: loop
5: Remove $p_{min} = \text{argmin}_{p \in \text{OPEN}}\{g(p) + h(p)\}$ from OPEN
6: if $p_{min}$ is a complete edit path then
7: Return $p_{min}$ as the solution
8: else
9: Let $p_{min} = \{u_1 \rightarrow v_{i1}, ..., u_k \rightarrow v_{ik}\}$
10: if $k < |V_1|$ then
11: For each $w \in V_2 \setminus \{v_{i1}, ..., v_{ik}\}$, insert $p_{min} \cup \{u_{k+1} \rightarrow w\}$ into OPEN
12: Insert $p_{min} \cup \{u_{k+1} \rightarrow \epsilon\}$ into OPEN
13: else
14: Insert $p_{min} \cup \bigcup_{w \in V_2 \setminus \{v_{i1}, ..., v_{ik}\}}\{\epsilon \rightarrow w\}$ into OPEN
15: end if
16: end if
17: end loop

optimal edit path. Instead of exploring all edit paths in the search tree, a parameter $s$ is set to an integer $n$ which is in charge of keeping the $n$ most promising partial edit paths in the OPEN set all the times. Such an algorithm leads to a sub-optimal solution.

2.3.4.2 Munkres

Recently, graph edit distance was extended to perform well on large graphs by speeding up its computations. One of the important works in this scope is the work of [63] who has proposed an approximate approach of $d_{plain}$ which finds an approximate edit path between two graphs using a bipartite matching procedure. In their work, Munkres algorithm [51] (referred to as $d_{BP}$) has been adapted to solve the bipartite matching problem in polynomial time. This algorithm has no restrictions regarding graphs’ size.

d_{BP} has reformulated the assignment problem as a problem of finding an optimal matching in a complete bipartite graph matching. A quadratic cost matrix is constructed between two graphs, each element $C_{ij}$ in the matrix $C$ corresponds to the cost of assigning the $i^{th}$ element of the graph $G_1$ to the $j^{th}$ element of the graph $G_2$. Thus the problem of graph matching is reduced to finding the minimum assignment cost $C_{ip}$ such that $p = p_1, ..., p_2$ is a matrix permutation. The left upper corner of the matrix contains all possible node substitutions, the diagonal of the right upper matrix represents the
The cost of all possible node deletions and the diagonal of the bottom left corner contains all possible node insertions. The bottom right corner elements cost is set to zero which concerns the substitution of $\epsilon - \epsilon$. Edges substitution, insertion and deletion are added to nodes substitution, nodes deletion and nodes insertion, respectively. $d_{BP}$ finds a permutation $p = p_1, \ldots, p_2$ that reduces the edit cost of $d_{plain}(C_i p)$.

The experimental part of this work was divided into two parts: running time and classification accuracy. Their proposed sub-optimal algorithm $d_{BP}$ was tested on different data-sets (Letter, COIL-100, GREC Fingerprint, Molecule and Proteins data-sets) and compared to other three graph edit distance algorithms ($d_{plain}$, $d_{A^*}$ and $d_{Beam(s)}$). Results showed that the classification accuracy was not affected by using such an approximate $d_{plain}$ algorithm besides to the fact that this algorithm performs remarkably faster than the optimal $d_{plain}$, $d_{A^*}$ and the suboptimal $d_{Beam(s)}$ algorithm as it performs in polynomial time. However, the evaluation study has also showed that there is a relatively large divergence when drawing a scatter plot of $d_{BP}$ versus $d_{plain}$. Such a result shows that $d_{BP}$ does not really permit to perform matching of two graphs which can theoretically be seen from reducing the graph matching problem to finding the minimum assignment cost.

2.4 Application to graph classification

Structural representations have become more and more popular in many application domains such as computer vision, image understanding, biology, chemistry, text processing or pattern recognition. As a consequence of the emergence of graph-based representations, new computing issues such as graph mining [43] [15], graph clustering [85] [60] or supervised graph classification [6] [48] provoked a growing interest.

This report deals with the inexact graph matching problem. One important application of graph matching methods is the graph classification. A graph classification algorithm aims at assigning a class to an unknown graph using a mapping function $f$. This function usually can be defined as follows:

**Definition 2.2.** Let $\chi$ be the set of the labeled graphs. Given a graph learning dataset $L = \{< g_i, c_i >\}_{i=1}^M$ where $g_i \in \chi$ is a labelled graph and $c_i \in C$ is a class of the graph among the $N$ classes. A graph classifier consists in inducing from $L$ a mapping function $f(g) : \chi \rightarrow C$ which assigns a class to an unknown graph.

In the literature, this problem is generally tackled using two kinds of approaches. The first one consists in using kernel based algorithms such as Support Vector Machines
(SVM) or Kernel Principal Component Analysis (KPCA) [38] [81]. Using such methods, the graph is embedded in a feature space composed of label sequences which are obtained through a graph traversal. The kernel values are then computed by measuring the similarity between label sequences. Such approaches have proven to achieve high performance but they are computationally expensive when the dataset is large. The second family consists in using a k-Nearest Neighbors (k-NN) rule in a dissimilarity space, using a given dissimilarity measure. This kind of approach is the most frequently chosen for its simplicity to implement and its good asymptotic behavior. However, it suffers from one major drawback: its combinatorial complexity. This report deals with the inexact matching problem with an application to graph classification problem where the dissimilarity space is directly linked to the cost induced by the inexact matching. It is the authors belief that a suitable matching would lead to an accurate graph distance.

2.5 Scientific Deadlocks

Challenges of graph matching in PR lie in the existence of very large graphs. Solving graph matching problems in large graphs requires either coming up with an approximate approach (i.e. suboptimal inexact graph matching technique).

However, one has to know these following facts:

- Lots of approximate graph matching techniques can be run in polynomial time. However, they do not guarantee to find the best or the optimal solutions of the original graph matching problem.
- There is always a tradeoff between optimality and rapidity thus to speed computations up while being optimal.

2.6 Conclusion

In PR, significant attention is given to IGM thanks to the continuous attributed graphs on which they can be applied. In this chapter, we first have provided an overview of the most common graph representational methods in PR. Second, we have reviewed graph matching methods focusing on IGM and taking some important criteria into account for synthesizing these methods.
Graph matching is a CPU consuming and a memory consuming task, for that reason, a vast number of algorithms have been proposed either to approximate this task (e.g. To prune off the search tree as soon as possible).

The table below lists the symbols used to refer to the related works, presented in Subsection 2.3.1, and their original methods’ names:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_{plain}$</td>
<td>Graph Edit Distance</td>
</tr>
<tr>
<td>$d_{A^*}$</td>
<td>A-star</td>
</tr>
<tr>
<td>$d_{Beam(s)}$</td>
<td>Beam Search</td>
</tr>
<tr>
<td>$d_{BP}$</td>
<td>Munkres’s algorithm</td>
</tr>
</tbody>
</table>

Table 2.3: Symbols used in this chapter and their methods’ names

At the end of this chapter, some scientific deadlocks of graph matching have been discussed.
Bibliography


Chapter 2. State of the Art


Chapter 2. State of the Art


[85] Hugo Zanghi, Christophe Ambroise, Vincent Miele, Hugo Zanghi, Christophe Ambroise, and Vincent Miele. Fast online Graph Clustering via Erd
denyi Mixture. (8), 2007.