Advances in Computer Vision and Pattern Recognition

Kaspar Riesen

Structural Pattern Recognition with Graph Edit Distance

Approximation Algorithms and Applications

Springer

Advances in Computer Vision and Pattern Recognition

More information about this series at http://www.springer.com/series/4205

Kaspar Riesen

Structural Pattern Recognition with Graph Edit Distance

Approximation Algorithms and Applications

Springer

Kaspar Riesen
Institut für Wirtschaftsinformatik
Fachhochschule Nordwestschweiz
Olten
Switzerland

ISSN 2191-6586 ISSN 2191-6594 (electronic)
Advances in Computer Vision and Pattern Recognition
ISBN 978-3-319-27251-1 ISBN 978-3-319-27252-8 (eBook)
DOI 10.1007/978-3-319-27252-8

Library of Congress Control Number: 2015958905

Printed on acid-free paper

This Springer imprint is published by SpringerNature
The registered company is Springer International Publishing AG Switzerland

To Madeleine, Emilie, and Maxime

Preface

The present book is concerned with the use of graphs in the field of structural pattern recognition. In fact, graphs are recognized as versatile alternative to feature vectors and thus, they found widespread application in pattern recognition and related fields (yet, the present book is actually focused on the field of pattern recognition only). In the last four decades, a huge number of procedures for graph distance computation, which is actually a basic requirement for pattern recognition, have been proposed in the literature. *Graph edit distance*, introduced about 30 years ago, is still one of the most flexible graph distance models available and subject of various recent research activities.

The objective of the present book is twofold. First, it gives a general and thorough introduction into the field of structural pattern recognition with a particular focus on graph edit distance (including a survey of graph edit distance applications that emerged during the last decade). Second, it presents a comprehensive compilation of diverse novel methods related to graph edit distance that have been developed and researched in the course of a recent research project that has been conducted under my supervision. In particular, the second part of the present book summarizes and consolidates the results of the following articles.[1]

1. Kaspar Riesen, Horst Bunke: Improving Bipartite Graph Edit Distance Approximation using Various Search Strategies. Pattern Recognition 48(4): 1349–1363 (2015).
2. Kaspar Riesen, Andreas Fischer, Horst Bunke: Estimating Graph Edit Distance Using Lower and Upper Bounds of Bipartite Approximations. IJPRAI 29(2) (2015).
3. Miquel Ferrer, Francesc Serratosa, Kaspar Riesen: Improving Bipartite Graph Matching by Assessing the Assignment Confidence. Pattern Recognition Letters, 2015. Accepted for Publication.

[1]We reuse several text excerpts, tables, and figures from the corresponding original publications with permission from *Elsevier*, *World Scientific*, and *IEEE*.

4. Kaspar Riesen, Miquel Ferrer: Predicting the Correctness of Node Assignments in Bipartite Graph Matching. Pattern Recognition Letters, 2015. Accepted for Publication.
5. Kaspar Riesen, Miquel Ferrer, Horst Bunke: Approximate Graph Edit Distance in Quadratic Time. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), 2015. Accepted for Publication.

Bern Kaspar Riesen
September 2015

Acknowledgments

I am deeply grateful to Horst Bunke for introducing me to the world of graphs and pattern recognition about 10 years ago. His excellent advice and open attitude towards new ideas have strongly influenced my scientific development.

Although only one author is mentioned on the front cover of this book, various colleagues contributed invaluable parts to this work. In particular, I would like to thank the co-authors of the papers, which actually build the basis of this book, viz., Andreas Fischer, Miquel Ferrer, Rolf Dornberger, Francesc Serratosa, and Horst Bunke.

Moreover, I would like to thank Thomas Strahm for his support in all administrative concerns as well as for encouraging me for writing the present book.

I would also like to acknowledge the funding by the Hasler Foundation Switzerland and the Swiss National Science Foundation (Project 200021 153249).

Furthermore, I would like to thank Simon Rees and Wayne Wheeler from Springer for their guidance and support.

Last but not least, I am very grateful to my family for their understanding and encouragement.

Contents

Part I
Foundations and Applications of Graph Edit Distance

Chapter 1
Introduction and Basic Concepts

Abstract In this chapter we first introduce pattern recognition as a computer science discipline and then outline the major differences between statistical and structural pattern recognition. In particular, we discuss the advantages and drawbacks of both approaches. Eventually, graph-based pattern representation is formally introduced and complemented by a list of applications where graphs are actually employed. The remaining parts of this chapter are then dedicated to formal introductions of diverse graph matching definitions. We particularly delve into the difference between exact and inexact graph matching. Last but not least, we give a brief survey of existing graph matching methodologies that somehow differ from the approach that is actually pursued in the present book.

1.1 Pattern Recognition

The ability of recognizing patterns has been essential for our survival and thus, evolution has led to highly sophisticated neural and cognitive systems in humans for solving pattern recognition tasks [1]. In fact, humans are faced with a great diversity of pattern recognition problems in their everyday life. Examples of pattern recognition tasks—which are in the majority of cases intuitively solved—include the recognition of a written or a spoken word, the face of a friend, an object on the table, a traffic sign on the road, and many others. These simple examples illustrate the essence of pattern recognition. In the world there exist classes of patterns which are recognized by humans according to certain knowledge learned before [2].

The terminology *pattern* refers to any observation in the real world (e.g., an image, an object, a symbol, or a word, to name just a few). The overall aim of *pattern recognition* as a computer science discipline is to develop methods that are able to (partially) imitate the human capacity of perception and intelligence. In other words, pattern recognition aims at defining algorithms that automate or (at least) support the process of recognizing patterns stemming from the real world.

However, pattern recognition refers to a highly complex process which cannot be solved by means of explicitly specified algorithms in general. For instance, to date one is not able to write an analytic algorithm to recognize, say, a face in a photo [3].

K. Riesen, *Structural Pattern Recognition with Graph Edit Distance*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-3-319-27252-8_1

In order to overcome this problem, machine-based pattern recognition is based on the so-called *learning paradigm*. That is, pattern recognition algorithms are commonly designed such that they can be trained on labeled data (referred to as *training data*). By means of this training data, a pattern recognition system is able to derive a model, which in turn can be used to solve the given pattern recognition task. Hence, the purpose of any pattern recognition system is to learn from examples such that it becomes able to make predictions about new, i.e., unseen, data.

Pattern recognition emerged as a very important and active discipline in computer science. This becomes also evident by the high number of scientific journals that are concerned with this research area (e.g., *Pattern Analysis and Applications* (Springer), *Pattern Recognition* and *Pattern Recognition Letters* (both Elsevier), and the *IEEE Transactions on Pattern Analysis and Machine Intelligence*, to name just a few examples). Moreover, pattern recognition methodologies are nowadays employed in various areas of science and industry. Handwriting recognition [4–9] and document analysis [10–15] are well-known examples of pattern recognition applications. Other prominent examples of pattern recognition applications include (biometric) person identification and authentication [16–24], activity predictions for molecular compounds [25–28], and function predictions for proteins [29–33].

1.1.1 Statistical and Structural Pattern Recognition

The question how to represent patterns in a formal way such that they can automatically be processed by machine is a key issue in pattern recognition and related fields. In general, there are two major ways to tackle this crucial step, viz., the *statistical* and the *structural approach*.

In the statistical approach, feature vectors are employed for representing the underlying patterns. That is, a pattern is formally represented as a vector $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ of n numerical features. Representing patterns by feature vectors offers a number of useful properties, in particular, the mathematical wealth of efficient operations available in a vector space, which has eventually resulted in a rich repository of algorithmic tools for statistical pattern recognition [1, 3, 34].

However, the use of feature vectors implicates the two following limitations.

1. As vectors always represent a predefined set of features, all vectors in a given application have to preserve the same length regardless the size or complexity of the corresponding pattern.
2. Vectors do not provide a direct possibility to describe binary (or higher order) relationships that might exist among different parts of a pattern.

These two drawbacks are severe, particularly when the patterns under consideration are characterized by complex structural relationships rather than the statistical distribution of a fixed set of pattern features.

Both limitations can be overcome by *graph-based pattern representation*, which is commonly used in a structural pattern recognition. Basically, graphs consist of finite sets of (labeled) nodes and edges, where the edges connect pairs of nodes.

Table 1.1 Complementary properties of feature vectors and graphs

	Vectors	Graphs
Representational power	Low	High
Efficiency	High	Low

Hence, graphs are able to not only describe properties of a pattern, but also (binary) relationships among different parts of the underlying pattern, or arrangement of patterns. Moreover, graphs are not constrained to a fixed size, i.e., the number of nodes and edges is not limited a priori but can be adapted to the size and the complexity of each individual pattern under consideration.

However, one drawback of graphs, when compared to feature vectors, is the significant increase of the complexity of many algorithms. Regard, for instance, the algorithmic comparison of two patterns. In a statistical pattern recognition, every vector has equal dimension, and moreover, the ith entry in any vector describes the same numerical property of the underlying pattern. Due to this homogeneous nature, comparison of two patterns is straightforward and can be accomplished in linear time with respect to the length of the two vectors (e.g., by means of the Euclidean distance). The same task for graphs, however, is much more complex, as the sets of nodes and edges are generally unordered and of arbitrary size. More formally, when comparing two graphs with each other, one has to identify common parts of the graphs by considering all of their subsets of nodes. Regarding that there are $O(2^n)$ subsets of nodes in a graph with n nodes, the inherent difficulty of graph comparison becomes obvious.

We have to conclude that the flexibility of graphs handicap their general applicability in pattern recognition. In summary and from a high-level perspective, we observe quite complementary properties of feature vectors and graphs (cf. Table 1.1). The present book is concerned with the use of graphs in pattern recognition and particularly addresses the efficiency problem of graph comparison.

Readers who are aware of the recent rise of *graph kernels* [35–37] and *graph embedding methods* [38–40] might interject that the traditional gap between statistical and structural pattern recognition has been bridged. In fact, both graph kernels and graph embeddings provide a powerful vectorial description of the underlying graphs. While graph kernels produce an implicit embedding of graphs into a Hilbert space, graph embeddings result in an explicit feature vector in a real vector space. Yet, both approaches crucially depend on similarity or dissimilarity computation on graphs. That is, in spite of the recent paradigm shift in structural pattern recognition, the topic of (efficient) graph comparison is still of high importance.

1.2 Graph and Subgraph

The following definition allows us to handle arbitrarily structured graphs with unconstrained labeling functions.

Definition 1.1 (*Graph*) Let L_V and L_E be finite or infinite label sets for nodes and edges, respectively. A *graph* g is a four-tuple $g = (V, E, \mu, \nu)$, where

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

The set of all graphs over the label alphabets L_V and L_E (also referred to as *graph domain*) is denoted by $\mathcal{G}$. The size of a graph g is denoted by $|g|$ and is defined as the number of nodes, i.e., $|g| = |V|$.

The labels for both nodes and edges can be given by the set of integers $L = \{1, 2, 3, \ldots\}$, the vector space $L = \mathbb{R}^n$, a set of symbolic labels $L = \{\alpha, \beta, \gamma, \ldots\}$, or a combination of various label alphabets from different domains. Given that the nodes and/or the edges are labeled, the graphs are referred to as *labeled graphs*.[1] *Unlabeled graphs* are obtained as a special case by assigning the same (empty) label $\varnothing$ to all nodes and edges, i.e., $L_V = L_E = \{\varnothing\}$. In some algorithms and applications it is necessary to include *empty "nodes"* and/or *empty "edges"* (also referred to as *null nodes* and *null edges*). We denote both empty nodes and empty edges by ε. Note the difference between an unlabeled and an empty node. An empty node ε is non-existing (i.e., *null*) while an unlabeled node u is an existing node from V with empty label $\mu(u) = \varnothing$ (the same accounts for the edges).

Edges are given by pairs of nodes (u, v), where $u \in V$ denotes the source node and $v \in V$ the target node of a directed edge. Commonly, the two nodes u and v connected by an edge (u, v) are referred to as *adjacent*. A graph is termed *complete* if all pairs of nodes are adjacent. The *degree* of a node $u \in V$ is the number of adjacent nodes to u, i.e., the number of *incident* edges of u.

Directed graphs directly correspond to the above-stated definition of a graph. However, by inserting a reverse edge $(v, u) \in E$ for every edge $(u, v) \in E$ with an identical label, i.e., $\nu(u, v) = \nu(v, u)$, the class of *undirected graphs* can be modeled as well. Since there are always edges in both directions in this case, the direction of an edge can be safely ignored.

A common approach to describe the edge structure of a graph $g = (V, E, \mu, \nu)$ is to define the adjacency matrix of g.

Definition 1.2 (*Adjacency Matrix*) Let $g = (V, E, \mu, \nu)$ be a graph with $|g| = n$. The $n \times n$ *adjacency matrix* $\mathbf{A} = (a_{ij})_{n \times n}$ of graph g is defined by

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

where $v_i, v_j \in V$ and ε refers to the empty edge.

[1] *Attributes* and *attributed graphs* are sometimes synonymously used for *labels* and *labeled graphs*, respectively.