Image and Vision Computing 27 (2009) 950–959

Contents lists available at ScienceDirect

Image and Vision Computing

journal homepage: www.elsevier.com/locate/imavis

Approximate graph edit distance computation by means of bipartite graph matching

Kaspar Riesen [*], Horst Bunke

Institute of Computer Science and Applied Mathematics, University of Bern, Neubrückstrasse 10, CH-3012 Bern, Switzerland

ARTICLE INFO

Article history:
Received 15 October 2007
Received in revised form 1 February 2008
Accepted 9 April 2008

Keywords:
Graph based representation
Graph edit distance
Bipartite graph matching

ABSTRACT

In recent years, the use of graph based object representation has gained popularity. Simultaneously, graph edit distance emerged as a powerful and flexible graph matching paradigm that can be used to address different tasks in pattern recognition, machine learning, and data mining. The key advantages of graph edit distance are its high degree of flexibility, which makes it applicable to any type of graph, and the fact that one can integrate domain specific knowledge about object similarity by means of specific edit cost functions. Its computational complexity, however, is exponential in the number of nodes of the involved graphs. Consequently, exact graph edit distance is feasible for graphs of rather small size only. In the present paper we introduce a novel algorithm which allows us to approximately, or suboptimally, compute edit distance in a substantially faster way. The proposed algorithm considers only local, rather than global, edge structure during the optimization process. In experiments on different datasets we demonstrate a substantial speed-up of our proposed method over two reference systems. Moreover, it is emprically verified that the accuracy of the suboptimal distance remains sufficiently accurate for various pattern recognition applications.

1. Introduction

Graph matching refers to the process of evaluating the structural similarity of graphs. A large number of methods for graph matching have been proposed in recent years [1]. The main advantage of a description of patterns by graphs instead of vectors is that graphs allow for a more powerful representation of structural relations. In the most general case, nodes and edges are labeled with arbitrary attributes. One of the most flexible methods for error-tolerant graph matching that is applicable to various kinds of graphs is based on the edit distance of graphs [2,3]. The idea of graph edit distance is to define the dissimilarity of graphs by the amount of distortion that is needed to transform one graph into another. Using the edit distance, an input graph to be classified can be analyzed by computing its dissimilarity to a number of training graphs. For classification, the resulting distance values may be fed, for instance, into a nearest-neighbor classifier. Alternatively, the edit distance of graphs can also be interpreted as a pattern similarity measure in the context of kernel machines, which makes a large number of powerful methods applicable to graphs [4], including support vector machines for classification and kernel principal component analysis for pattern analysis. There are various applications where the edit distance has proved to be suitable for error-tolerant graph matching [5,6].

Yet, the error-tolerant nature of edit distance potentially allows every node of a graph to be mapped to every node of another graph (unlike exact graph matching methods such as subgraph isomorphism or maximum common subgraph). The time and space complexity of edit distance computation is therefore very high. Consequently, the edit distance can be computed for graphs of a rather small size only.

In recent years, a number of methods addressing the high computational complexity of graph edit distance computation have been proposed. In some approaches, the basic idea is to perform a local search to solve the graph matching problem, that is, to optimize local criteria instead of global, or optimal ones [7,8]. In [9], a linear programming method for computing the edit distance of graphs with unlabeled edges is proposed. The method can be used to derive lower and upper edit distance bounds in polynomial time. Two fast but suboptimal algorithms for graph edit distance computation are proposed in [10]. The authors propose simple variants of an optimal edit distance algorithm that make the computation substantially faster. A number of graph matching methods based on genetic algorithms have been proposed [11]. Genetic algorithms offer an efficient way to cope with large search spaces, but are non-deterministic and suboptimal. A common way to make graph matching more efficient is to restrict considerations to special classes of graphs. Examples include the classes of planar graphs [12], bounded-valence graphs [13], trees [14], and graphs with unique vertex labels [15]. Recently, a suboptimal edit distance algorithm has been proposed [5] that requires the nodes of graphs to be

[*] Corresponding author. Tel./fax: +41 316318699.
E-mail addresses: riesen@iam.unibe.ch (K. Riesen), bunke@iam.unibe.ch (H. Bunke).

0262-8856/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.imavis.2008.04.004

planarly embedded, which is satisfied in many, but not all computer vision applications of graph matching.

In this paper, we propose a new efficient algorithm for edit distance computation for general graphs. The method is based on an (optimal) fast bipartite optimization procedure mapping nodes and their local structure of one graph to nodes and their local structure of another graph. This procedure is somewhat similar in spirit to the method proposed in [16,17]. However, rather than using dynamic programming for finding an optimal match between the sets of local structure, we make use of Munkres' algorithm [18]. Originally, this algorithm has been proposed to solve the assignment problem in polynomial time. However, in the present paper we generalize the original algorithm to the computation of graph edit distance. In experiments on semi-artificial and real-world data we demonstrate that the proposed method results in a substantial speed-up of the computation of graph edit distance, while at the same time the accuracy of the approximated distances is not much affected.

A preliminary version of the current paper appeared in [19]. The current paper has been significantly extended with respect to the underlying methodology and the experimental evaluation.

2. Graph edit distance computation

In this section we first define our basic notation and then introduce graph edit distance and its computation. Let L be a finite or infinite set of labels for nodes and edges.

Definition 1. (Graph) A graph g is a four-tuple $g = (V, E, \mu, \nu)$, where

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

This definition allows us to handle arbitrary graphs with unconstrained labeling functions. For example, the labels can be given by the set of integers, the vector space $\mathbb{R}^n$, or a set of symbolic labels $L = \{\alpha, \beta, \gamma, \ldots\}$. Moreover, unlabeled graphs are obtained as a special case by assigning the same label l to all nodes and 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. Undirected graphs can be modeled by inserting a reverse edge $(v, u) \in E$ for each edge $(u, v) \in E$ with $\nu(u, v) = \nu(v, u)$.

Graph matching refers to the task of evaluating the dissimilarity of graphs. One of the most flexible methods for measuring the dissimilarity of graphs is the edit distance [2,3]. Originally, the edit distance has been proposed in the context of string matching [20]. Procedures for edit distance computation aim at deriving a dissimilarity measure from the number of distortions one has to apply to transform one pattern into the other. The concept of edit distance has been extended from strings to trees and eventually to graphs [2,3]. Similarly to string edit distance, the key idea of graph edit distance is to define the dissimilarity, or distance, of graphs by the minimum amount of distortion that is needed to transform one graph into another. Compared to other approaches to graph matching, graph edit distance is known to be very flexible since it can handle arbitrary graphs and any type of node and edge labels. Furthermore, by defining costs for edit operations, the concept of edit distance can be tailored to specific applications.

A standard set of distortion operations 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 $(u \rightarrow \varepsilon)$, and the insertion of node v by $(\varepsilon \rightarrow v)$. For edges we use a similar notation. Other operations, such as merging and splitting of nodes [21], can be useful in certain applications but are not considered in this paper. Given two graphs, the source graph g_1 and the target graph g_2, the idea of graph edit distance is to delete some nodes and edges from g_1, relabel (substitute) some of the remaining nodes and edges, and insert some nodes and edges in g_2, such that g_1 is finally transformed into g_2. A sequence of edit operations $e_1, \ldots, e_k$ that transform g_1 completely into g_2 is called an edit path between g_1 and g_2. In Fig. 1 an example of an edit path between two graphs g_1 and g_2 is given. This edit path consists of three edge deletions, one node deletion, one node insertion, two edge insertions, and two node substitutions.

Obviously, for every pair of graphs (g_1, g_2), there exist a number of different edit paths transforming g_1 into g_2. Let $\Upsilon(g_1, g_2)$ denote the set of all such edit paths. To find the most suitable edit path out of $\Upsilon(g_1, g_2)$, one introduces a cost for each edit operation, measuring the strength of the corresponding operation. The idea of such a cost function is to define whether or not an edit operation represents a strong modification of the graph. Obviously, the cost function is defined with respect to the underlying node or edge labels.

Clearly, between two similar graphs, there should exist an inexpensive edit path, representing low cost operations, while for dissimilar graphs an edit path with high costs is needed. Consequently, the edit distance of two graphs is defined by the minimum cost edit path between two graphs.

Definition 2. (Graph Edit Distance) 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

$$d(g_1, g_2) = \min_{(e_1, \ldots, e_k) \in \Upsilon(g_1, g_2)} \sum_{i=1}^{k} c(e_i),$$

where $\Upsilon(g_1, g_2)$ denotes the set of edit paths transforming g_1 into g_2, and c denotes the cost function measuring the strength $c(e_i)$ of edit operation e_i.

The computation of the edit distance is usually carried out by means of a tree search algorithm which explores the space of all possible mappings of the nodes and edges of the first graph to the nodes and edges of the second graph. A widely used method is based on the A* algorithm [22] which is a best-first search algorithm. The basic idea is to organize the underlying search space as an ordered tree. The root node of the search tree represents the starting point of our search procedure, inner nodes of the search tree correspond to partial solutions, and leaf nodes represent complete – not necessarily optimal – solutions. Such a search tree is constructed dynamically at runtime by iteratively creating successor nodes linked by edges to the currently considered node in the search tree. In order to determine the most promising node in the current search tree, i.e. the node which will be used for further expansion of the desired mapping in the next iteration, a heuristic function is usually used. Formally, for a node p in the search tree, we use $g(p)$ to denote the cost of the optimal path from the root node to the current node p, i.e. $g(p)$ is set equal to the cost of the partial edit path accumulated so far, and we use $h(p)$ for denoting

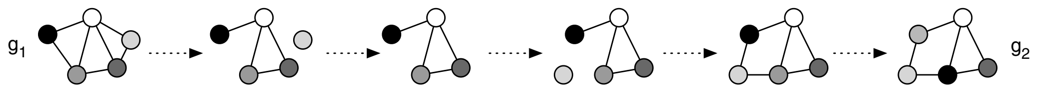

Fig. 1. A possible edit path between graph g_1 and g_2 (node labels are represented by different shades of grey).