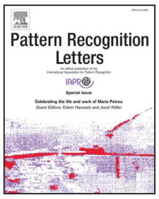
Contents lists available at ScienceDirect

Pattern Recognition Letters

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



Improving bipartite graph matching by assessing the assignment confidence[☆]

Miquel Ferrer[a], Francesc Serratosa[b], Kaspar Riesen[a,*]

[a] Institute for Information Systems, University of Applied Sciences and Arts Northwestern Switzerland, Riggenbachstrasse 16, Olten CH-4600, Switzerland
[b] Departament d'Enginyeria Informàtica i Matemàtiques, Universitat Rovira i Virgili, Avda. Països Catalans 26, Tarragona 43007, Spain

ARTICLE INFO

Article history:
Received 27 February 2015
Available online 22 July 2015

Keywords:
Graph matching
Graph edit distance
Beam search

ABSTRACT

Due to the ability of graphs to represent properties of entities and binary relations at the same time, a growing interest in this representation formalism can be observed in various fields of pattern recognition. The availability of a distance measure is a basic requirement for pattern recognition. For graphs, graph edit distance is still one of the most popular distance measures. In the present paper we substantially improve the distance accuracy of a recent framework for the approximation of graph edit distance. The basic idea of our novel approach is to manipulate the initial assignment returned by the approximation algorithm such that the individual assignments are ordered according to their individual confidence. Next, the individual assignments are post processed in this specific order. In an experimental evaluation we show that the order of the assignments plays a crucial role for the resulting distance accuracy. Moreover, we empirically verify that our novel generalization is able to generate approximations which are very near to the exact edit distance (in contrast with the original framework).

1. Introduction

The availability of a distance measure is a basic requirement for pattern recognition. If feature vectors are used for pattern representation, the Euclidean distance and similar measures can be used. If graphs are used instead, a large number of procedures for graph dissimilarity computation, commonly referred to as *graph matching*, exist (see ([7,29]) for exhaustive surveys). They include methods from *spectral graph theory* ([5,18]), *relaxation labeling* ([20,28,30]), and *graduated assignments* ([14,27]).

With the rise of graph kernels and graph embedding methods ([1,4,11,13]), the traditional gap between statistical and structural pattern recognition has been bridged. Both graph kernels and graph embedding provide a powerful vectorial description of the underlying graphs. Yet, both approaches crucially depend on similarity or dissimilarity computation on graphs and thus graph matching remains an important topic in the field of structural pattern recognition.

Graph edit distance ([3,25]), introduced about 30 years ago, is still one of the most flexible and versatile graph matching models available. Note that many graph matching models (e.g. *spectral methods* ([5,18])) are purely structural, in the sense that they are only applicable to unlabeled graphs, or they allow only severely constrained label alphabets. Yet, graph edit distance is able to cope with both directed and undirected, as well as with both labeled and unlabeled graphs. In addition, if there are labels on nodes, edges, or both, no constraints on the respective label alphabets have to be considered. Moreover, through the use of a cost function graph edit distance can be adapted and tailored to various problem specifications.

Due to this high flexibility, graph edit distance has found widespread applications. That is, the concept of graph edit distance has been successfully applied in the field of chemoinformatics for predicting or analyzing certain properties of molecular compounds ([2,12]). Malware Detection, i.e. the distinction between malicious and original binary executables ([17]). Fingerprint classification ([21]), or handwriting recognition ([10]) are other prominent examples where graph edit distance has proved to be suitable for error-tolerant graph matching.

The major drawback of graph edit distance is, however, its computational complexity which is exponential in the number of nodes of the involved graphs. Consequently, exact edit distance can be computed for graphs of rather small size only. Recently an algorithmic framework for the approximate computation of graph edit distance in cubic time has been presented ([23]). The substantial speed-up of this approximation is, however, at the expense of a general overestimation of the actual graph edit distance. The reason for this overestimation is that the core of the approximation framework is able to consider only local, rather than the global, edge structure of the underlying graphs.

[☆] This paper has been recommended for acceptance by M. Couprie.
[*] Corresponding author: Tel.: +41 79 688 7719; fax: +41 31 6318699
E-mail address: kaspar.riesen@fhnw.ch (K. Riesen).

http://dx.doi.org/10.1016/j.patrec.2015.07.010
0167-8655/© 2015 Elsevier B.V. All rights reserved.

In order to overcome this problem and reduce the overestimation a variation of the original framework has been recently proposed in ([24]). Given the initial matching found by the original framework, the main idea is to carry out a post processing step such that the number of incorrect assignments is decreased (which in turn reduces the overall overestimation). The proposed post processing varies the original node mapping by systematically swapping the target nodes of two individual node assignments. In order to search the space of assignment variations a *beam search* (i.e. a tree search with pruning) is used. One of the most important observations derived from ([24]) is that given an initial node assignment, one can substantially reduce the overestimation using this local search method.

Yet, to date the node mapping is varied without using any kind of heuristic information. In particular, it is not taken into account that certain nodes and/or local assignments are more evident than others, and should thus be considered first (or last) in the search process. In this paper we propose six different heuristics (plus inverse of them) that modify the initial node assignment returned by the original approximation framework. These heuristics are used to influence the order in which the assignments are eventually varied during the beam search. In other words, prior to run the beam search strategy proposed in ([24]), the order of the assignments is varied according to these heuristics.

The basics of the present paper have been lined up in preliminary papers ([8,9]). Yet, the present paper has been significantly extended with respect to the underlying methodology and the experimental evaluation. In particular, rather than using various randomized permutations of the assignments (as proposed in ([8])) we now use deterministic procedures for influencing the assignment order. Moreover, compared to ([9]) the number of sorting strategies as well as the number of data sets is substantially increased. Last but not least, we introduce a combination framework for using different sorting strategies simultaneously and thoroughly investigate the impact of such combinations.

2. Graph edit distance computation

In this section we start with our basic notation of graphs and then review the concept of graph edit distance. Eventually, the approximate graph edit distance algorithms (which build the basis of the present work) are briefly described[1].

2.1. Graph edit distance

A graph g is a four-tuple $g = (V, E, \mu, \kappa)$, where V is the finite set of nodes, $E \subseteq V \times V$ is the set of edges, $\mu : V \longrightarrow L_V$ is the node labeling function, and $\kappa : E \longrightarrow L_E$ is the edge labeling function. The labels for both nodes and edges can be given, for instance, by 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. Unlabeled graphs are a special case by assigning the same (empty) label $\varnothing$ to all nodes and edges.

Given two graphs, $g_1 = (V_1, E_1, \mu_1, \kappa_1)$ and $g_2 = (V_2, E_2, \mu_2, \kappa_2)$, the basic idea of graph edit distance is to transform g_1 into g_2 using edit operations, namely, *insertions*, *deletions*, and *substitutions* of both nodes and edges. The substitution of two nodes u and v is denoted by $(u \rightarrow v)$, the deletion of node u by $(u \rightarrow \epsilon)$, and the insertion of node v by $(\epsilon \rightarrow v)$[2].

A sequence $(e_1, \ldots, e_k)$ of k edit operations e_i that transform g_1 completely into g_2 is called an edit path $\lambda(g_1, g_2)$ between g_1 and g_2. Let $\Upsilon(g_1, g_2)$ denote the set of all edit paths between two graphs g_1 and g_2. To find the most suitable edit path out of $\Upsilon(g_1, g_2)$, one commonly introduces a cost function $c(e)$ for every edit operation e, measuring the strength of the corresponding operation. Hence, cost functions allow the integration of domain specific knowledge about object similarity in the matching process (note that automatic procedures for learning the edit costs from a set of sample graphs are also available ([6,26])).

Clearly, between two similar graphs, there should exist an inexpensive edit path, representing low cost edit operations, while for dissimilar graphs an edit path with high cost is needed. Consequently, the edit distance $d_{\lambda_{\min}}(g_1, g_2)$, or $d_{\lambda_{\min}}$ for short, between two graphs g_1 and g_2 is defined by

$$d_{\lambda_{\min}}(g_1, g_2) = \min_{\lambda \in \Upsilon(g_1, g_2)} \sum_{e_i \in \lambda} c(e_i). \qquad (1)$$

2.2. Bipartite graph edit distance approximation

Exact computation of graph edit distance is usually carried out by means of a tree search algorithm (e.g. A* ([15])) 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 major drawback of such an exhaustive search is its computational complexity which is exponential in the number of nodes. In fact, the problem of graph edit distance can be reformulated as an instance of a *Quadratic Assignment Problem* (*QAP*) which is known to be $\mathcal{NP}$*-complete*.

The approximate graph matching framework introduced in ([23]) reduces the problem of graph edit distance computation to an instance of a *Linear Sum Assignment Problem* (*LSAP*) which can be – in contrast with QAPs – efficiently solved. The reformulation of graph edit distance to an instance of an LSAP is carried out by means of the following three major steps.

First Step. Assume that the graphs to be matched consist of node sets $V_1 = \{u_1, \ldots, u_n\}$ and $V_2 = \{v_1, \ldots, v_m\}$, respectively. A cost matrix $\mathbf{C}$ is then defined as follows:

$$\mathbf{C} = \left[\begin{array}{cccc|cccc}
c_{11} & c_{12} & \cdots & c_{1m} & c_{1\varepsilon} & \infty & \cdots & \infty \\
c_{21} & c_{22} & \cdots & c_{2m} & \infty & c_{2\varepsilon} & \ddots & \vdots \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \infty \\
c_{n1} & c_{n2} & \cdots & c_{nm} & \infty & \cdots & \infty & c_{n\varepsilon} \\
\hline
c_{\varepsilon 1} & \infty & \cdots & \infty & 0 & 0 & \cdots & 0 \\
\infty & c_{\varepsilon 2} & \ddots & \vdots & 0 & 0 & \ddots & \vdots \\
\vdots & \ddots & \ddots & \infty & \vdots & \ddots & \ddots & 0 \\
\infty & \cdots & \infty & c_{\varepsilon m} & 0 & \cdots & 0 & 0
\end{array}\right]$$

Entries c_{ij} (defined in the left upper part of $\mathbf{C}$) denote the cost of node substitutions $(u_i \rightarrow v_j)$, $c_{i\varepsilon}$ (defined in the right upper part of $\mathbf{C}$) denote the cost of node deletions $(u_i \rightarrow \varepsilon)$, and $c_{\varepsilon j}$ (defined in the left lower part of $\mathbf{C}$) denote the cost of node insertions $(\varepsilon \rightarrow v_j)$. Note that every node can be deleted or inserted at most once. Therefore any non-diagonal element of the right-upper and left-lower part is set to ∞. The bottom right part of the cost matrix is set to zero since substitutions of the form $(\varepsilon \rightarrow \varepsilon)$ should not cause any cost.

Second Step. Given the cost matrix $\mathbf{C} = (c_{ij})$, the LSAP optimization consists in finding a permutation $(\varphi_1, \ldots, \varphi_{n+m})$ of the integers $(1, 2, \ldots, (n+m))$ that minimizes the overall assignment cost $\sum_{i=1}^{(n+m)} c_{i\varphi_i}$. In fact, this problem can be solved in polynomial time by means of Munkres' algorithm ([19]), the algorithm of Volgenant–Jonker ([16]), or many others.

The resulting permutation corresponds to the mapping

$$\psi = ((u_1 \rightarrow v_{\varphi_1}), (u_2 \rightarrow v_{\varphi_2}), \ldots, (u_{m+n} \rightarrow v_{\varphi_{m+n}}))$$

[1] In Table 4 at the end of the present paper the different notations/symbols that are used in the paper are summarized.

[2] Similar notation is used for edges.