Pattern Recognition 80 (2018) 210–224

Contents lists available at ScienceDirect

Pattern Recognition

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

A novel hierarchical-based framework for upper bound computation of graph edit distance

Karam Gouda[a,b,*], Mona Arafa[a], Toon Calders[b,c]

[a] Faculty of Computers & Informatics, Benha University, Egypt
[b] Department of Computer & Decision Engineering, Université Libre de Bruxelles, Belgium
[c] Department of Mathematics & Computer Sciences, University of Antwerp, Belgium

ARTICLE INFO

Article history:
Received 12 May 2017
Revised 18 November 2017
Accepted 20 March 2018
Available online 21 March 2018

Keywords:
Graph similarity
Graph edit distance
Upper bound

ABSTRACT

Graph similarity is an important notion with many applications. Graph edit distance is one of the most flexible graph similarity measure available. The main problem with this measure is that in practice it can only be computed for small graphs due to its exponential time complexity. The present paper is concerned with efficient solutions with high quality approximation of graph edit distance. In particular, we present a novel upper bound computation framework for graph edit distance. It is based on breadth-first hierarchical views of the graphs and a novel hierarchical traversing and matching method to build a graph mapping. The main advantage of this framework is that it combines map construction with edit counting in easy and straightforward manner. It also allows to compare the graphs from different hierarchical views to improve the bound. Furthermore, to avoid the complexity of multi-view comparisons and preserve distance accuracy, two new view-selection methods, based on the vertex and edge star structures, are introduced to scale the computations. Contrasting our approach with the state-of-the-art overestimation methods, experiments show that it delivers comparable upper bounds with over three orders of magnitude speedup on real data graphs. Experiments also show that this approach improves the classification accuracy of the KNN classifiers by over 15 percent when compared with the state-of-the-art overestimation methods.

1. Introduction

Powerful data structures such as graphs are currently used to represent complex entities and their relationships in many application areas. These areas include Pattern Recognition [1], Social Networks [2], Software Engineering [3], Bio-informatics [4], Semantic Web [5], Chem-informatics [6], etc. Yet, the expressive power and flexibility of the graph data model comes at the cost of high computational complexity of many basic operations. Graph edit distance is one of such operations which has recently drawn lots of interest in the research community. Given two labeled graphs, graph edit distance measures the minimum cost of graph edits to be performed on one of them to get the other. A graph edit operation is usually one of vertex insertion/deletion, edge insertion/deletion or a change of a vertex'/edge's label in the graph.

Due to the rich information provided by its associated edit sequence as well as its ability to cope with any kind of graph structures and labeling scheme, graph edit distance is considered as one of the most flexible graph similarity measure available for labeled graphs. Today, graph edit similarity plays a significant role in managing graph data [7–10], and is employed in a variety of analysis tasks such as graph classification and clustering [11,12], object recognition in computer vision [1], etc. Unfortunately, the main problem with this measure is that it can only be computed for small graphs due to its exponential time complexity [13–16]. Computing graph edit distance is known to be an NP-hard problem [8]. In practice, to be able to compare large graphs, algorithms seeking suboptimal solutions have been proposed [1,8,12,17–24]. Most of these solutions deliver bounded values.

Many recently proposed interesting upper bounds of graph edit distance, as well as the upper bound introduced in this paper, are obtained using graph mapping methods. Given a graph mapping one can define a graph editing and vice versa. The intuition behind these methods is that the better the mapping between graphs, the better the upper bound on their edit distance. In [8,25] a graph mapping approach is introduced. The idea at the heart of this ap-

* Corresponding author at: Department of Computer & Decision Engineering, Université Libre de Bruxelles, Belgium.
E-mail addresses: karam.gouda@fci.bu.edu.eg, karam.gouda@ulb.ac.be (K. Gouda), mona.arafa@fci.bu.edu.eg (M. Arafa), toon.calders@ulb.ac.be (T. Calders).

https://doi.org/10.1016/j.patcog.2018.03.019
0031-3203/

proach is based on the observation that matching vertices with similar neighborhoods often yields graph mapping with low edit cost. In this approach, dummy vertices are first inserted into the smaller graph to equalize the number of vertices at both graphs. A cost matrix between the vertices of the two graphs is then built, where each entry holds the matching edit cost between the neighbourhoods of the corresponding vertices. Using this matrix, a cubic-time bipartite assignment algorithm such as the Hungarian Algorithm [26] is used to optimally match the vertices and build a vertex map. Finally, the edit cost of this map is calculated and returned as an upper bound of graph edit distance. The main problem with this approach, however, is that the pairwise vertex cost considers the graph structure only locally. Hence, in cases where neighborhoods do not differentiate the vertices, such as in unlabeled graphs, this approach performs poorly. Recent improvements consider modifying the initial graph mapping by deploying additional search strategies in a post processing phase. For example, an exhaustive vertex swapping procedure is used in [8], a greedy vertex swapping is used in [27,28], and a randomized vertex swapping based on simulated annealing is used in [24]. Despite the computation overhead of these post-processing steps (from $O(|V|^2)$ to $O(|V|^6)$ additional time), the resulting graph mappings are still prone to local optima, and depend on the initialization. On the other hand, many attempts have been made towards improving the efficiency of the basic vertex assignment. In [22], the cubic-time complexity is avoided by using a quadratic-time greedy solution when computing the vertex assignment. Recently, in [23], a new quadratic-time approach is developed to reduce the loss in accuracy of the previous greedy vertex assignment by allowing bidirectional assignments between corresponding vertices.

In this paper, we presents a novel linear-time upper bound computation framework for graph edit distance, named Breadth First Search Tree based Edit Distance (BFST_ED), which pursues a completely different graph mapping construction strategy. The basic idea of our approach is to picture the graphs to be compared as hierarchical structures. This view facilitates comparison and graph mapping construction. BFST_ED adopts the breadth-first hierarchical view of graphs, where each graph is represented as a breadth first search tree (BFST for short) and a set of backward edges. A *concurrent pre-order* traversing and matching method of the corresponding BFSTs is then developed in order to build a vertex map between graphs. Using this map, the edit costs on backward edges are calculated and then added to the tree mapping edit cost to produce an upper bound of graph edit distance. The BFSTs mapping method is improved as follows: First, instead of matching the vertices in a strict preorder traversal of BFSTs, it is allowed for a vertex to find a suitable match among different options by comparing the children of both the visited vertex and its match before visiting any of these children in the preorder. Second, the idea of spare trees is introduced to reduce the number of insertions/deletions incurred by the mapping, and a local bipartite matching based on lookahead is used to enhance the vertex matching process.

This novel framework allows to explore a quadratic space of graph mappings to improve the upper bound, where for each two corresponding vertices it is possible to run the BFSTs mapping method on the distinct hierarchical views imposed by these vertices. To avoid the complexity of multi-view comparisons and preserve distance accuracy, two new view-selection methods are introduced to scale the computations. These methods are based on the vertex and edge star structures [8,16,25] and their edit similarities. Another solution would be the possibility of carrying out view comparisons in parallel, a feature which is not offered by any of the state-of-the-art distance approximation methods. Contrasting our approach with the state-of-the-art overestimation methods, experiments show that it delivers comparable upper bounds with over three orders of magnitude speedup on real data graphs. Experiments also show that this approach improves the classification accuracy of the KNN classifiers by over 15 percent when compared with the state-of-the-art overestimation methods. A preliminary version of this paper is introduced in [29]. The work in the current paper has significantly extended the idea with respect to the underlying methodology and the experimental evaluation.

The remainder of this paper is organized as follows. Preliminary concepts are presented in Section 2. Section 3 presents the framework at higher level, the different BFSTs matching methods, and finally the complexity analysis. Section 4 is devoted to the upper bound improvements. The experimental results are reported in Section 5. Section 6 concludes the paper.

2. Preliminaries

2.1. Graphs

Let Σ be a set of discrete-valued labels. A labeled, undirected graph G is a triple $G = (V, E, l)$, where $V = \{v_1, v_2, \ldots, v_{|V|}\}$ is a set of vertices, $E = \{e_1, \ldots, e_{|E|}\} \subset V \times V$ is a set of undirected edges, and l is a labeling function $l: V \cup E \to \Sigma$, assigning for each vertex $v \in V$ or edge $e \in E$ an alphabet character $l(v) \in \Sigma$ or $l(e) \in \Sigma$. $|V|$ and $|E|$ are called the *order* and *size* of G, respectively. For each vertex $u \in V$, the *neighborhood* of u in G is given as $N(u) = \{v : (u, v) \in E\}$, and the degree of u in G as $deg(u) = |N(u)|$. A labeled, undirected graph G is said to be *connected* if each pair of vertices u_i, $u_j \in G$, $i \neq j$, are directly or indirectly connected. An undirected graph is *simple* if it neither contains self-loops nor multiple edges. This paper focuses on simple and connected graphs with only labeled vertices. An extension to edge-labeled graphs is straightforward.

A graph $G = (V, E, l)$ is a *subgraph* of another graph $G' = (V', E', l')$ (or G' is a supergraph of G), denoted $G \subseteq G'$, if there exists a *subgraph isomorphism* from G to G'. We may simply say that G' contains G.

Definition 1 (Sub-)graph isomorphism**.** A subgraph isomorphism is an *injective* function $f: V \to V'$, such that (1) $\forall\ u \in V$, $l(u) = l'(f(u))$. (2) $\forall\ (u, v) \in E$, $(f(u), f(v)) \in E'$. If $G \subseteq G'$ and $G' \subseteq G$, then we say that G and G' are graph isomorphic to each other, denoted by $G \cong G'$.

Definition 2 (Maximum) common sub-graph**.** Given two graphs G_1 and G_2. A graph $G = (V, E)$ is said to be a common sub-graph of G_1 and G_2 if $\exists\ H_1 \subseteq G_1$ and $H_2 \subseteq G_2$ such that $G \cong H_1 \cong H_2$. A common sub-graph G is a maximum common edge (resp. vertex) sub-graph if there exists no other common sub-graph $G' = (V', E')$ such that $|E'| > |E|$ (resp. $|V'| > |V|$).

2.2. Graph editing and graph edit distance

A graph G can be transformed into another graph by elementary edit operations consisting of inserting or deleting a vertex or an edge, or changing a vertex label. Notice that a vertex can be deleted only if its incident edges have been previously deleted. Each elementary edit operation p is associated to a application-dependent cost $c(p)$, measuring the strength of the corresponding operation. Given two graphs G_1 and G_2, the sequence of edit operations performed on one of them to get the other is called a *graph editing*. Formally, let p_i be an edit operation, a graph editing $Gedit = \langle p_i \rangle_{i=1}^{k}$ is a sequence of edit operations $\langle p_1, p_2, \ldots, p_k \rangle$ that transform G_1 into G_2, that is, $Gedit(G_1) = G_1 \to^{p_1} G^1 \to^{p_2} G^2 \ldots \to^{p_k} G^k \cong G_2$. The cost of a graph editing is the sum of its edit operation's costs, i.e., $\mathcal{C}(G_1, G_2, Gedit) = \sum_{i=1}^{k} c(p_i)$.

Given two graphs G_1 and G_2, clearly, there could be multiple edit sequences that turn G_1 into G_2. An optimal graph editing is