New binary linear programming formulation to compute the graph edit distance

Julien Lerouge, Zeina Abu-Aisheh, Romain Raveaux, Pierre Héroux, Sébastien Adam

Abstract

In this paper, a new binary linear programming formulation for computing the exact Graph Edit Distance (GED) between two graphs is proposed. A fundamental strength of the formulations lies in their genericity since the GED can be computed between directed or undirected fully attributed graphs. Moreover, a continuous relaxation of the domain constraints in the formulation provides an efficient lower bound approximation of the GED. A complete experimental study that compares the proposed formulations with six state-of-the-art algorithms is provided. By considering both the accuracy of the proposed solution and the efficiency of the algorithms as performance criteria, the results show that none of the compared solutions dominate the others in the Pareto sense. In general, our formulation converges faster to optimality while being able to scale up to match the largest graphs in our experiments. The relaxed formulation leads to an accurate approach that is 12% more accurate than the best approximate method of our benchmark.

1. Introduction

Graphs are data structures that can describe complex entities through their elementary components (the vertices of the graph) and the relational properties between them (the edges of the graph). For attributed graphs, both vertices and edges can be characterized by attributes that can vary from nominal labels to more complex descriptions such as strings or feature vectors. This arrangement leads to very powerful representations that are used in many application domains such as computer vision, biology, chemistry or text processing. Computing the dissimilarity of such graphs is a crucial issue for graph-based pattern recognition. An enormous number of algorithms have been proposed in the literature to solve this problem. They can be categorized as embedding-based vs. matching-based methods.

In embedding-based methods, the key-idea is to project the input graphs to be compared into a vector space. Then, a norm is computed in this space. Thus, such methods bridge the gap between statistical and structural pattern recognition [1,2]. A natural way to perform this projection is to compute a feature vector for each graph to be compared [3–6]. Another type of graph-embedding approach consists of representing the graphs as vectors of distances to a number of graph prototypes [7], but the embedding of the graphs requires itself a dissimilarity computation method. Graph kernels [8–10] can also be considered as embedding-based approaches since they produce an implicit embedding of the graphs into a Hilbert space. All of these embedding-based methods are generally computationally effective since they do not involve a matching process. However, they do not take into account the complete relational properties and do not provide a matching between the graphs.

In matching-based methods, the similarity between two graphs requires the computation and the evaluation of the “best” matching between them. Since exact isomorphism rarely occurs in pattern analysis applications, the matching process must be error-tolerant, i.e., it must tolerate differences in the topology and/or in its labeling. To tackle this problem, spectral methods such as [11] have been studied. They are based on the eigendecomposition of the adjacency or Laplacian matrix of a graph. In this spectral framework, the graphs are unlabeled or only severely constrained label alphabets. Another well known error-tolerant matching-based method that can be used to evaluate the dissimilarity between two graphs is the Graph Edit Distance (GED) [12]. In this method, a set of graph edit operations is introduced black. Each edit operation is characterized by a cost, and the GED is the total cost of the least expensive sequence of operations that transforms one graph into the other. The GED is a dissimilarity measure for arbitrarily struc-
tured and/or labeled graphs. In contrast with other approaches, it does not suffer from any restriction and can be applied to any type of graph, including hypergraphs [13]. The GED has been used in many applications, e.g., malware detection [14], chemoinformatics [15], or document analysis [16].

A main usability limitation of the GED is its computational complexity since it is known to be NP-complete [17,18]. Computing the exact GED using \( A^* \) is exponential in the number of nodes and is only feasible for graphs of a rather small size (typically 10 nodes). To overcome this limitation, many contributions have been proposed over the last decade. Some are based on the proposition of new heuristics to improve the performance of exact approaches [19,20] whereas others have proposed faster but suboptimal methods that approximate the exact GED (e.g., [21–26]).

In this paper, we tackle the GED problem using Binary Linear Programming (BLP). Starting from a straightforward linear formulation of the GED, we derive a new exact BLP. This program is theoretically shown to be equivalent to the first approach (i.e., it computes the exact GED) and experimentally shown to be more effective. We also show that a relaxation of the domain constraints in this new formulation provides an efficient lower bound which can be used as an accurate approximation of the GED.

The performance of both the exact formulations and their approximations are compared with those of six exact and approximate approaches, including a previous BLP-based approach proposed in [27]. Each method is evaluated from both the precision and the efficiency point of view. For the sake of equality, all of the methods use the same edit operation cost values. These values are taken from reference work in the literature [7,15,28]. The experiments are performed on seven reference datasets which were carefully chosen to show the behaviors of the approaches on different types of graphs [28–30].

The results show that the new BLP formulation can compute an exact GED on larger graphs than the existing approaches and can compute the GED between richly attributed graphs (i.e., with attributes on both the vertices and edges), which cannot be handled using the BLP formulation proposed in [27]. They also show that our relaxed formulation is more accurate than recent approximation-based approaches, at the cost of extra computational time.

Section 2 presents the important definitions that are necessary for introducing our formulations of the GED. Then, Section 3 reviews the existing approaches for computing the GED with exact and approximate methods. Section 4 describes the proposed BLP formulations. Section 5 presents the experiments and analyzes the obtained results. Section 6 provides the study's conclusions.

2. Problem statement

Definition 1. An attributed graph \( G \) is a 4-tuple \( G = (V, E, \mu, \xi) \), where \( V \) is a set of vertices; \( E \) is a set of edges such that \( \forall e = (i, j) \in E, i \in V \) and \( j \in V \); \( \mu : V \rightarrow L_V \) is a vertex labeling function that associates a label \( \mu(v) \) to \( v \in V \), where \( L_V \) is the set labels for the vertices, and \( \xi : E \rightarrow L_E \) is an edge labeling function that associates a label \( \xi(e) \) to \( e \in E \), where \( L_E \) is the set of labels for the edges.

The vertices (resp. edges) label space \( L_V \) (resp. \( L_E \)) could be composed of any combination of numeric, symbolic or string attributes. A graph \( G \) is said to be simple if it has no loop (an edge that connects a vertex to itself) and no multiedge (several edges between the same vertices). In this case, \( E \subseteq \{(i, j) \in V \times V | i \neq j\} \) and an edge can be univocally designated by the pair of vertices that it connects. Otherwise, \( G \) is a multigraph and \( E \) is a multiset. A graph \( G \) is said to be undirected if the relation \( E \) is symmetric, i.e., if its edges have no orientation. In this case, \( \forall (i, j) \in E, (j, i) \in E \) and \( (i, i) \notin E \). Otherwise, \( G \) is a directed graph. Hence, Definition 1 allows us to handle arbitrarily structured graphs (directed or undirected, simple graphs or multigraphs) with unconstrained labeling.

The GED is commonly used to measure the dissimilarity between two graphs. The GED is an error-tolerant graph matching method. It defines the dissimilarity of two graphs by the minimum amount of distortion that is required to transform one graph into another [12].

Definition 2. The graph edit distance \( d(\ldots) \) is a function \( d : G \times G \rightarrow \mathbb{R}^+ \)

\[
(G_1, G_2) \rightarrow d(G_1, G_2) = \min_{(o_1, \ldots, o_k) \in \Gamma(G_1, G_2)} \sum_{i=1}^{k} c(o_i)
\]

where \( G_1 = (V_1, E_1, \mu_1, \xi_1) \) and \( G_2 = (V_2, E_2, \mu_2, \xi_2) \) are two graphs from the set \( G \) and \( \Gamma(G_1, G_2) \) is the set of all edit paths \( o = (o_1, \ldots, o_k) \) that allow transforming \( G_1 \) to \( G_2 \). An elementary edit operation \( o_i \) is one of vertex substitution \((v_1 \rightarrow v_2)\), edge substitution \((e_1 \rightarrow e_2)\), vertex deletion \((v_1 \rightarrow \epsilon)\), edge deletion \((e_1 \rightarrow \epsilon)\), vertex insertion \((\epsilon \rightarrow v_2)\) and edge insertion \((\epsilon \rightarrow e_2)\) with \( v_1 \in V_1, v_2 \in V_2, e_1 \in E_1 \) and \( e_2 \in E_2 \). Here, \( \epsilon \) is a dummy vertex or edge that is used to model the insertion or deletion. Additionally, \( c(.) \) is a function that associates a cost to each elementary edit operation \( o_i \).

The cost function \( c(.) \) is of primary interest for the GED computation and can change the problem that is being solved. In [31,32], a particular cost function for the GED is introduced, and it is shown that under this cost function, the GED computation is equivalent to the maximum common subgraph problem. Neuhaus and Bunke [33] have shown that if each elementary operation satisfies the criteria of a distance (separability, symmetry and triangular inequality) then the edit distance defines a metric between graphs. Recently, some methods have been proposed to learn the matching edit cost between graphs [34,35]. The discussion around the cost functions is beyond the topic of this paper, which focuses on GED computation for given costs.

When the GED is computed between the attributed graphs, the edit costs are usually defined as functions of the vertices (resp. edges) attributes. More precisely, substitution costs are defined as a function of the attributes of the substituted vertices (resp. edges), whereas insertion and deletion are penalized with a value that is linked to the attributes of the inserted/deleted vertex (resp. edge). In our experiments, we set the cost function to the values proposed in [7].

3. Related works

The GED has been the subject of many contributions in the literature, including some very complete surveys of existing approaches [36,37]. These reviews usually distinguish exact approaches from approximations.

3.1. Exact approaches

The first family of exact computation of the GED is based on the widely known \( A^* \) algorithm. This algorithm relies on the exploration of the tree of solutions. In this tree, each node corresponds to a partial edition of the graph. A leaf of the tree corresponds to an edit path that transforms one graph into the other. The exploration of the tree is guided by developing the most promising ways on the basis of an estimation of the GED. For each node, this estimation is the sum of the cost associated with the partial edit path and an estimation of the cost for the remaining path. The latter is given by a heuristic. Provided that the estimation of the future
دریافت فوری
متن کامل مقاله

امکان دانلود نسخه تمام متن مقالات انگلیسی
امکان دانلود نسخه ترجمه شده مقالات
پذیرش سفارش ترجمه تخصصی
امکان جستجو در آرشیو جامعی از صدها موضوع و هزاران مقاله
امکان دانلود رایگان ۲ صفحه اول هر مقاله
امکان پرداخت اینترنتی با کلیه کارت های عضو شتاب
دانلود فوری مقاله پس از پرداخت آنلاین
پشتیبانی کامل خرید با بهره مندی از سپرده هوشمند رهگیری سفارشات