New Binary Linear Programming Formulation to Compute the Graph Edit Distance

Julien Lerouge[a], Zeina Abu-Aisheh[b], Romain Raveaux[b], Pierre Héroux[a], Sébastien Adam[a]

[a] *Normandie Univ, UNIROUEN, UNIHAVRE, INSA Rouen, LITIS, 76000 Rouen, France*
[b] *LI, Université François Rabelais de Tours, 37200 Tours, France*

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 methods 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.

Keywords: Graph Edit Distance, Integer Linear Programming, Graph Matching, Pattern Matching.

Email addresses: Julien.Lerouge@univ-rouen.fr (Julien Lerouge), zeina.abu-aisheh@univ-tours.fr (Zeina Abu-Aisheh), romain.raveaux@univ-tours.fr (Romain Raveaux), pierre.heroux@univ-rouen.fr (Pierre Héroux), sebastien.adam@univ-rouen.fr (Sébastien Adam)

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 eigen decomposition 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 structured 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], chemioinformatics [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 unambiguously 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 $(i,j) = (j,i)$. 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 $d(.,.)$ is a function*

$$\begin{aligned} d \quad : \quad & \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}^+ \\ & (G_1, G_2) \mapsto d(G_1, G_2) = \min_{(o_1,\ldots,o_k)\in\Gamma(G_1,G_2)} \sum_{i=1}^{k} c(o_i) \end{aligned}$$

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 $\mathcal{G}$ 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, ϵ 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] and [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 cost is lower than or equal to the real cost, an optimal path from the root node to a leaf node is guaranteed to be found [38]. A simple way to fulfill this constraint would be to set the estimation of the future cost to zero, but this setting could lead to exploring the whole tree of solutions. The other extreme consists of computing the real cost for the remaining edit

path and would require an exponential amount of time. Indeed, the smaller the difference between the estimation and the real future cost is, the smaller the number of nodes that will be expanded by the A* algorithm. The different A*-based methods published in the literature mainly differ in the implemented heuristics for the future cost estimation which correspond to different tradeoffs between the approximation quality and their computation time [19, 20].

In another family of algorithms, the GED is computed by solving a BLP. To the best of our knowledge, Almohamad and Duffuaa [39] proposed the first BLP formulation of the weighted graph matching problem. It consists of determining the permutation matrix that minimizes the L_1 norm of the difference between the adjacency matrix of the input graph and the permuted adjacency matrix of the target graph. More recently, Justice and Hero [27] also proposed a BLP formulation of the GED problem. The proposed program searches for the permutation matrix that minimizes the cost of transforming G_1 into G_2, with G_1 and G_2 two unweighted and undirected graphs. The criterion to be minimized (eq. 1) accounts for the costs for the matching vertices, but the formulation does not integrate the ability to process graphs with labels on their edges.

$$d(G_1, G_2) \;=\; \min_{P} \sum_{i=1}^{n} \sum_{j=1}^{n} c(l(A_1^i), l(A_2^j)) P_{i,j} \;+\; \frac{1}{2} c(0,1) |A_1 \;-\; P A_2 P^T|^{ij} \quad (1)$$

where A_k is the adjacency matrix of G_k, and P is an orthogonal permutation matrix such that $PP^T = P^T P = I$, $l(A_k^i)$ is the label of the i^{th} vertex in G_k, while c is a cost function. A mathematical transformation is used to transform this non linear optimization problem into a linear problem.

In the proposed formulation, the number of constraints and variables grows quadratically with the number of vertices in the graphs, and thus, it could impact considerably the memory consumption of the program. Moreover, the modeling of the graphs by means of an adjacency matrix restricts the formulation to the processing of simple graphs.

3.2. Approximations

Considering that exact computation of the GED can be performed in a reasonable amount of time only for small graphs, and many researchers have focused their effort on the computation of the GED approximations in polynomial time. For example, in their paper [27], Justice and Hero proposed a lower bound of the GED that can be computed in $\mathcal{O}((n_1 + n_2)^7)$, where $n_1 = |V_1|$ and $n_2 = |V_2|$, by extending the domain of variables in P from $\{0, 1\}$ to $[0, 1]$. In the same paper, an upper bound was proposed by reducing the GED problem to the Linear Sum Assignment Problem (LSAP). The LSAP can be solved by the Hungarian method (also called Munkres assignment algorithm [40]) in polynomial time $\mathcal{O}(n^3)$ where n is the dimension of the vertex assignment cost matrix. The cost matrix of dimension $n_1 + n_2$ was filled due to the first term of eq. 1. In the same direction, Riesen *et al.* [23] proposed to enrich the assignment cost matrix with the edge edit costs. Finally, the vertex assignment was used to derive an edit path, and its associated cost is an upper bound of the GED.

In [24], Neuhaus *et al.* proposed two other approximations based on A^* method. The first one, called A^*-BeamSearch, proposed to prune the tree of solutions by limiting the number of concurrent partial solutions to the q most promising solutions. At the end of the algorithm, a valid edit path and its associated cost are provided, but there is no guarantee that it corresponds to the optimal path, since the latter could have been eliminated in earlier steps of the algorithm. The parameter q manages the trade-off between the combinatorial cost and the quality of the approximation. This method provides an upper bound of the exact GED. In the same paper, a method called A^*-Pathlength is proposed to speed up the access to a leaf node in the tree of solutions by giving a higher exploration priority to long partial edit paths. This strategy is motivated by the observation that first assignments are the most computationally expensive and that they are rarely called into question.

More recently, in [41], the vertex assignment computed by means of bipartite graph matching is used as an initialization step for a genetic algorithm that

attempts to improve the quality of the approximation. Indeed, from any vertex assignment, it is possible to derive an edit path and finally compute its cost [23]. The vertex assignment that is optimal in terms of vertex substitution is not always optimal for the whole edit path. The initial population is generated by deriving mappings that are mutated versions of the mapping that has been determined by the Hungarian algorithm. The probability of a vertex mapping to be selected is linked to the vertex substitution cost. The lower the corresponding edit distance is, the better the individual fits the objective function. The genetic algorithm iterates by selecting and mixing several mappings.

Fischer *et al.* [20] proposed to integrate in the A* algorithm a heuristic based on a modifed Hausdorff distance. This distance is computed in a time complexity of $\mathcal{O}(n_1.n_2)$. This heuristic has also been used on its own without the A* algorithm in [21]. GED approximations have also been proposed in a probabilistic framework [25]. Thus, the objective is to find the vertex assignment that maximizes the a posteriori probability while considering the vertex attributes. However, the corresponding heuristics are unbounded and cannot be exploited by branch and bound algorithms to prune the tree of solutions or to efficiently prioritize its exploration in the A* algorithm.

4. Graph Edit Distance Using Binary Linear Programming

In this article, the GED problem is modeled by a BLP which is a restriction of integer linear programming (ILP) in which the variables are binary. Hence, its general form is the following:

$$\min_{x} c^T x \tag{2a}$$

$$\text{subject to } Ax \leq b \tag{2b}$$

$$x \in \{0,1\}^n \tag{2c}$$

where $c \in \mathbb{R}^n, A \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^m$ are data of the problem. A feasible solution is a vector x of n binary variables (2c) that respects linear inequality constraints (2b). If the program has at least one feasible solution, then the

optimal solutions are the solutions that minimize the objective function (2a), which is a linear combination of variables of x weighted by the components of the vector c.

In this section, we present some formulations that were written for GED computation. The first is a straightforward formulation deduced from Definition 2 whereas the second is more refined, with fewer variables and constraints. The third concerns undirected graphs. Then, we discuss how the formulations are solved, and the impact of their differences on performance issues. We also show that relaxing formulations can provide a lower bound of the GED.

4.1. Modelling the GED problem as a BLP

4.1.1. Variable and cost functions definitions

Our goal is to compute the GED between two graphs $G_1 = (V_1, E_1, \mu_1, \xi_1)$ and $G_2 = (V_2, E_2, \mu_2, \xi_2)$. In this section, for the sake of simplicity of notations, G_1 and G_2 are simple directed graphs. However, the formulations can be applied without modification to multigraphs, and with some slight modifications in the undirected case (these modifications are explained in 4.3).

In Definition 2, the edit operations that are allowed to transform the graphs G_1 and G_2 are (i) the substitution of a vertex (respectively, an edge) of G_1 with a vertex (resp. an edge) of G_2, (ii) the deletion of a vertex (or an edge) from G_1 and (iii) the insertion of a vertex (or an edge) in G_1. In Table 1, we define a set of binary variables for each of these edit operations. For example, we are given two sets of vertices V_1 and V_2, where i a vertex from V_1, which is substituted with k a vertex from V_2 has an edit cost of $c_{i,k}$. Variable $x_{i,k} = 1$ if a vertex i is substituted with a vertex k and it is 0 otherwise. The concept holds true for the other variables.

Using these notations, an edit path between G_1 and G_2 is defined as a 6-tuple $(\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v}, \mathbf{e}, \mathbf{f})$ where $\mathbf{x} = (x_{i,k})_{(i,k)\in V_1\times V_2}$, $\mathbf{y} = (y_{ij,kl})_{(ij,kl)\in E_1\times E_2}$, $\mathbf{u} = (u_i)_{i\in V_1}$, $\mathbf{e} = (e_{ij})_{ij\in E_1}$, $\mathbf{v} = (v_k)_{k\in V_2}$ and $\mathbf{f} = (f_{kl})_{kl\in E_2}$ (see Table 1 for the definition of each variable).

To evaluate the global cost of an edit path, the elementary costs for each edit operation must be defined. The notations used for these costs are given in Table 1. These cost functions traditionally depend on the labels of the vertices and edges. As stated before, defining these cost functions is out of the scope of our contributions and reference definitions are used in our experiments.

Table 1: Notations for the GED framework

Edit operation	Variable	Cost
Substitution of vertex i by vertex k	$x_{i,k}$	$c_{i,k}$
Deletion of vertex i	u_i	$c_{i,\epsilon}$
Insertion of vetex k	v_k	$c_{\epsilon,k}$
Substitution of edge ij by edge kl	$y_{ij,kl}$	$c_{ij,kl}$
Deletion of edge ij	e_{ij}	$c_{ij,\epsilon}$
Insertion of edge kl	f_{kl}	$c_{\epsilon,kl}$

4.1.2. Objective function

The objective function (3) to be minimized is the overall cost induced by an edit path $(\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v}, \mathbf{e}, \mathbf{f})$ that transforms graph G_1 into graph G_2.

$$C(\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v}, \mathbf{e}, \mathbf{f}) = \sum_{i \in V_1} \sum_{k \in V_2} c_{i,k} \cdot x_{i,k} + \sum_{ij \in E_1} \sum_{kl \in E_2} c_{ij,kl} \cdot y_{ij,kl} + \sum_{i \in V_1} c_{i,\epsilon} \cdot u_i + \sum_{k \in V_2} c_{\epsilon,k} \cdot v_k + \sum_{ij \in E_1} c_{ij,\epsilon} \cdot e_{ij} + \sum_{kl \in E_2} c_{\epsilon,kl} \cdot f_{kl} \tag{3}$$

4.1.3. Constraints

The constraints are designed to guarantee that the admissible solutions of the BLP are edit paths that transform G_1 in G_2. An edit path is considered to be admissible if and only if the following conditions are respected. (i) It provides a one-to-one mapping between a subset of the vertices of G_1 and a subset of the vertices of G_2. This one-to-one mapping is equivalent to vertex substitution. The remaining vertices are either deleted or inserted. (ii) It provides

a one-to-one mapping between a subset of the edges of G_1 and a subset of the edges of G_2. This one-to-one mapping is equivalent to edge substitution. The remaining edges are either deleted or inserted. (iii) The vertex mappings and edge mappings are consistent, i.e., the graph topology is respected.

(i) Vertex mapping constraints The constraint (4) ensures that each vertex of G_1 is either mapped to exactly one vertex of G_2 or deleted from G_1, while the constraint (5) ensures that each vertex of G_2 is either mapped to exactly one vertex of G_1 or inserted in G_1:

$$u_i + \sum_{k \in V_2} x_{i,k} = 1 \quad \forall i \in V_1 \tag{4}$$

$$v_k + \sum_{i \in V_1} x_{i,k} = 1 \quad \forall k \in V_2 \tag{5}$$

(ii) Edges mapping constraints The constraints (6) and (7) guarantee a valid mapping between the edges:

$$e_{ij} + \sum_{kl \in E_2} y_{ij,kl} = 1 \quad \forall ij \in E_1 \tag{6}$$

$$f_{kl} + \sum_{ij \in E_1} y_{ij,kl} = 1 \quad \forall kl \in E_2 \tag{7}$$

(iii) Topological constraints The respect of the graph topology in the mapping is described in the following proposition :

Proposition 1. *An edge $ij \in E_1$ can be mapped to an edge $kl \in E_2$ only if the head vertices $i \in V_1$ and $k \in V_2$, and tail vertices $j \in V_1$ and $l \in V_2$, are respectively mapped.*

This quadratic constraint is expressed linearly with constraints (8) and (9):

- ij and kl can be mapped only if their head vertices are mapped:

$$y_{ij,kl} \leq x_{i,k} \quad \forall (ij, kl) \in E_1 \times E_2 \tag{8}$$

- ij and kl can be mapped only if their tail vertices are mapped:

$$y_{ij,kl} \leq x_{j,l} \quad \forall (ij, kl) \in E_1 \times E_2 \tag{9}$$

Equations 8 and 9 contribute not only to edge substitutions, but also to edge deletions and insertions. Let $i \in V_1$ such that i is deleted from G_1 ($u_i = 1$). Using Equation 4, we deduce that $\forall k \in V_2, x_{i,k} = 0$. Then, using Equation 8, $\forall j \in V_1$ such that $ij \in E_1$ and $\forall kl \in E_2$, $y_{ij,kl} \leq x_{i,k} = 0$ and, using Equation 6, $e_{ij} = 1$, which deletes edge ij. Consequently, if $i \in V_1$ is deleted from G_1, then all of the edges $ij \in E_1$ are deleted from G_1. Similarly, if $k \in V_2$ is inserted into G_1, then all of the edges $kl \in E_2$ are inserted into G_1. These two properties also hold true for the tail vertices, which ensures a consistent edge mapping between the two sets $E_1 \cup \epsilon$ and $E_2 \cup \epsilon$.

4.1.4. Straightforward formulation F1

Placing Equations 3 to 9 together with domain constraints that ensure that the solution is made of binary variables leads to a straightforward version of the BLP formulation called F1. This formulation F1 has $|V_1| + |V_2| + |E_1| + |E_2| + |V_1| \cdot |V_2| + |E_1| \cdot |E_2|$ variables and $|V_1| + |V_2| + |E_1| + |E_2| + 2 \cdot |E_1| \cdot |E_2|$ constraints (without the domain constraints).

4.2. Reducing the size of the formulation

In this subsection, we present a second exact formulation of the GED called F2, which has been derived from the formulation F1. We show that this formulation is theoretically equivalent to F1 and that it reduces the number of variables and the number of constraints.

4.2.1. Reducing the number of variables

The mapping constraints (4), (5),(6) and (7) can be transformed into inequality constraints, without changing their role in the program:

$$\sum_{k \in V_2} x_{i,k} \leq 1 \quad \forall i \in V_1 \tag{10}$$

$$\sum_{i \in V_1} x_{i,k} \leq 1 \quad \forall k \in V_2 \tag{11}$$

$$\sum_{kl \in E_2} y_{ij,kl} \leq 1 \quad \forall ij \in E_1 \tag{12}$$

$$\sum_{ij \in E_1} y_{ij,kl} \leq 1 \quad \forall kl \in E_2 \tag{13}$$

Replacing in Equation 3 the variables $\mathbf{u}, \mathbf{v}, \mathbf{e}$ and $\mathbf{f}$ by their expressions deduced from equations (4),(5), (6) and (7), we get a new objective function:

$$C'(\mathbf{x}, \mathbf{y}) = \sum_{i \in V_1} \sum_{k \in V_2} (c_{i,k} - c_{i,\epsilon} - c_{\epsilon,k}) \cdot x_{i,k} + \sum_{ij \in E_1} \sum_{kl \in E_2} (c_{ij,kl} - c_{ij,\epsilon} - c_{\epsilon,kl}) \cdot y_{ij,kl} + \gamma$$
$$\left(\text{with } \gamma = \sum_{i \in V_1} c_{i,\epsilon} + \sum_{k \in V_2} c_{\epsilon,k} + \sum_{ij \in E_1} c_{ij,\epsilon} + \sum_{kl \in E_2} c_{\epsilon,kl} \right) \tag{14}$$

Equation (14) shows that the GED can be obtained without explicitly computing the variables $\mathbf{u}, \mathbf{v}, \mathbf{e}$ and $\mathbf{f}$. Once the formulation solved, all insertion and deletion variables can be *a posteriori* deduced from the substitution variables.

4.2.2. Reducing the number of constraints

In the formulation F1, the number of topological constraints, (8) and (9), is $|E_1| \cdot |E_2|$. Therefore, on average, the number of constraints grows quadratically with the density of the graphs. We show that it is possible to formulate the GED problem with potentially less constraints, leaving the set of solutions unchanged. To this end, we mathematically express Proposition 1 in another way. We replace the constraints (8) and (9) by the following ones:

- Given an edge $ij \in E_1$ and a vertex $k \in V_2$, there is at most one edge whose initial vertex is k that can be mapped with ij:

$$\sum_{kl \in E_2} y_{ij,kl} \leq x_{i,k} \quad \forall k \in V_2, \forall ij \in E_1 \tag{15}$$

- Given an edge $ij \in E_1$ and a vertex $l \in V_2$, there is at most one edge whose terminal vertex is l that can be mapped with ij:

$$\sum_{kl \in E_2} y_{ij,kl} \leq x_{j,l} \quad \forall l \in V_2, \forall ij \in E_1 \tag{16}$$

Proposition 2. *Let Γ_1 be the set of edit paths (between G_1 and G_2) implied by the set of admissible solutions of F1, and let Γ_2 be the set of edit paths obtained*

similarly by replacing in F1 the constraints (8) *and* (9) *by the constraints* (15) *and* (16). *Then,* $\Gamma_1 = \Gamma_2$.

Proof.

$\Gamma_2 \subseteq \Gamma_1$: Let $ij \in E_1$ and $kl \in E_2$, and let us suppose that (15) is satisfied.

$$x_{i,k} \geq \sum_{kl' \in E_2} y_{ij,kl'} \Rightarrow \quad x_{i,k} \geq y_{ij,kl} + \sum_{kl' \in E_2, kl' \neq kl} y_{ij,kl'} \Rightarrow \quad x_{i,k} \geq y_{ij,kl}$$

Thus, constraint (8) is satisfied for all $ij \in E_1$ and for all $kl \in E_2$. Similarly, we deduce that (9) is satisfied using the constraint (16).

$\Gamma_1 \subseteq \Gamma_2$: Let $ij \in E_1$ and $k \in V_2$. If $\{l \in V_2 : kl \in E_2\} = \emptyset$, then $\sum_{kl \in E_2} y_{ij,kl} = 0$, and (15) is satisfied. Otherwise, using constraint (8), we have the following:

$$\forall kl \in E_2, x_{i,k} \geq y_{ij,kl} \;\Rightarrow\; x_{i,k} \geq \max_{kl \in E_2}(y_{ij,kl})$$

(6) ensures that $\text{card}\{l' \in V_2 : y_{ij,kl} = 1\} \leq 1$, and thus, $\max_{kl' \in E_2}(y_{ij,kl'}) = \sum_{kl' \in E_2} y_{ij,kl'} \Rightarrow \quad x_{i,k} \geq \sum_{kl' \in E_2} y_{ij,kl'}$ and (15) is still satisfied.

Thus, the constraint (15) is satisfied for all $ij \in E_1$ and for all $k \in V_2$. Similarly, we prove that (16) is satisfied using (9) and (7). □

The number of topological constraints, (15) and (16), is now $|E_1| \cdot |V_2|$. On average, it grows linearly with the density of the graphs. This relationship leads to substantially shorter formulations of the GED as the numbers of graph vertices and edges grow. In addition, we prove that the constraints (12) and (13) are not necessary to the formulation of the GED problem, since they are implied by other constraints of the BLP.

Proposition 3. *Constraint* (12) *is implied by* (10) *and* (15)

Proof. Let $ij \in E_1$. Given (15), we have

$$\sum_{kl \in E_2} y_{ij,kl} \leq x_{i,k} \quad \forall k \in V_2 \Rightarrow \sum_{k \in V_2} \sum_{kl \in E_2} y_{ij,kl} \leq \sum_{k \in V_2} x_{i,k}$$

We reduce the left term of this inequation and use (10):

$$\sum_{kl \in E_2} y_{ij,kl} \leq \sum_{k \in V_2} x_{i,k} \leq 1$$

Thus, (12) is implied by (10) and (15). Similarly, we prove that (13) is implied by (11) and (16). □

4.2.3. Simplified formulation F2

The results obtained in 4.2.1 and 4.2.2 show that the GED problem can be solved using (14) as the objective function and (10), (11), (15) and (16) as the constraints. We have finally arrived at a new formulation F2 of the GED problem, as follows:

$$\min_{\mathbf{x},\mathbf{y}} \left(\sum_{i \in V_1} \sum_{k \in V_2} \left(c_{i,k} - c_{i,\epsilon} - c_{\epsilon,k} \right) \cdot x_{i,k} + \sum_{ij \in E_1} \sum_{kl \in E_2} \left(c_{ij,kl} - c_{ij,\epsilon} - c_{\epsilon,kl} \right) \cdot y_{ij,kl} \right) + \gamma \tag{17a}$$

$$\text{subject to} \quad \sum_{k \in V_2} x_{i,k} \leq 1 \quad \forall i \in V_1 \tag{17b}$$

$$\sum_{i \in V_1} x_{i,k} \leq 1 \quad \forall k \in V_2 \tag{17c}$$

$$\sum_{kl \in E_2} y_{ij,kl} \leq x_{i,k} \quad \forall k \in V_2, \forall ij \in E_1 \tag{17d}$$

$$\sum_{kl \in E_2} y_{ij,kl} \leq x_{j,l} \quad \forall l \in V_2, \forall ij \in E_1 \tag{17e}$$

Here, γ is not a function of $\mathbf{x}$ and $\mathbf{y}$. It does not impact the minimization problem. However, γ is mandatory to obtain the GED value.

The formulation F2 has $|V_1| \cdot |V_2| + |E_1| \cdot |E_2|$ variables and $|V_1| + |V_2| + 2|V_2| \cdot |E_1|$ constraints. The domain constraints have not been account for because they are, by design, part of the BLP framework.

4.3. Extension to undirected graphs

Suppose that G_1 and G_2 are undirected graphs, i.e., their edges have no orientation. The notations ij and ji refer to the same edge of E_1 and kl and lk refer to the same edge of E_2. This new assumption leads to revise the sets of constraints (17d) and (17e) into the following single constraint:

$$\sum_{kl \in E_2} y_{ij,kl} \leq (x_{i,k} + x_{j,k}) \quad \forall k \in V_2, \forall ij \in E_1 \tag{18}$$

Indeed, given an edge $ij \in E_1$ and a vertex $k \in V_2$, there is at most one edge that is incident to k that can be matched to ij. Moreover, $x_{i,k}$ and $x_{j,k}$ cannot be simultaneously equal to 1, and thus, the sum $x_{i,k} + x_{j,k}$ is at most equal to 1.

4.4. Comparing the solving of F1 and F2 and obtaining lower bounds

In the previous subsection, the objective functions of F1 and F2 are equal. Furthermore, propositions 1 and 2 demonstrated that the set of constraints of F1 and F2 describes the exact same set of admissible solutions. Since F1 and F2 are two minimization problems with the same objective function over the same set of admissible solutions, their optimal solution is the same, and when optimality is reached, the value of their objective functions is the GED. Thus, they are strictly equivalent. However, we have also shown that F2 uses fewer variables than F1, and depending on the density of the graphs, it potentially uses fewer constraints to solve the same problem. Thus, in terms of the solution time and the used memory, they might not behave the same.

Solving an ILP that is defined using Equations (2a) to (2c) is NP-hard [18], and thus, exploring the entire solution tree would take an exponential amount of time. However, dedicated solvers have been developed to reduce the number of explored solutions and the solution time, by using a branch-and-cut algorithm along with some heuristics [42]. Given an instance of the problem, the solver explores the tree of solutions with the branch-and-bound algorithm and finds the best feasible solution, in terms of the objective function optimization.

The continuous relaxation of an ILP is a linear program (LP) in which the constraints are unmodified but the variables are now continuous. The LP solution is a lower bound for the solution in the initial problem. It can be reached in polynomial time $\mathcal{O}(n^{3.5})$ with the interior point method [43] where n is the number of variables in the model. This lower bound helps the ILP in finding the solution by pruning the exploration of the solution tree.

The continuous relaxation complexity is related to the number of variables and constraints in the model. Since F2 holds fewer variables and constraints

than F1, the continuous relaxation of F2 is faster to compute. A straightforward effect is that solving the ILP of F2 might also be faster than F1, since the LP relaxation is used by the ILP solving methods to cut the unpromising branches.

An important aspect to be highlighted here is that continuous relaxation can also be used to approximate the optimal objective value in polynomial time. We explore this opportunity in the experimental part of the paper by calling F1LP (resp. F2LP) the continuous relaxation of F1 (resp. F2). To accomplish this goal, we only substitute the discrete space $\{0, 1\}$ by the continuous space $[0, 1]$ in the domain constraints. In the experiments of Section 5, F1LP and F2LP are compared to approximate methods from the literature.

5. Experiments

This section aims at evaluating the proposed contributions through a robust experimental study that compares ten methods on reference datasets. We first describe the methods that have been studied, the datasets and the protocol. Then, the results are presented and discussed.

5.1. Studied methods

In this experimental part, our proposals (F1, F2, F1LP and F2LP) are compared against six GED algorithms from the literature. From the related works, we chose two exact methods and four approximate methods. On the exact method side, the A* algorithm applied to the GED problem [19] is a foundational work. In our tests, the heuristic is computed using the bipartite approximation [19]. A* with a bipartite heuristic is the most well-known exact GED method and is often used to evaluate the accuracy of the approximate methods. The second exact method is the BLP proposed by Justice and Hero in [27]. This method, called JH in the paper, is directly linked to our proposal. Since this method cannot address edge attributes, we could not perform JH on all of our datasets.

On the approximate method side, we can distinguish three families of methods in the literature: tree-based methods, assignment-based methods and set-based methods. For the tree-based methods, a truncated version of A* called A*-BeamSearch was chosen (BS) using a bipartite heuristic. This method is known to be one of the most accurate approximations from the literature [24]. Among the assignment-based methods, we selected the bipartite graph matching (BP) that is described in [23]. This upper bound was shown to be a good compromise between the speed and accuracy. We also added the Fast Bipartite method (FBP) [44] since it is an extension of BP. FBP has been used to question the community about the accuracy and speed-up [45]. Finally, we picked a set-based method proposed in [21]. This method (H) provides a lower bound of GED that is obtained using Hausdorff matching.

5.2. Datasets

In this paper, GED algorithms are evaluated on six different real world graph datasets and a synthetic dataset. These datasets have been chosen by carefully reviewing all of the publicly available datasets that have been used in the reference works mentioned in section 3 (LETTER, GREC, COIL, Alkane, FINGERPRINT, PAH, MUTA, PROTEIN and AIDS to name the most frequent datasets). On the basis of this review, a subset of these datasets has been chosen to obtain a good representativeness of the different graph features that can affect the GED computation (size, labeling, directed/undirected):

GREC [29] is composed of undirected graphs that have rather small size (i.e., up to 20 vertices). In addition, continuous attributes on vertices and edges play an important role in the matching procedure. Such graphs are representative of pattern recognition problems.

MUTA of molecules [29] is representative of exact matching problems because a significant part of the topology together with the corresponding vertex and edge labels in G_1 and G_2 are required to be identical. In addition, this set of graphs gathers large instances with up to 70 vertices.

PROTEIN [29] is a molecule dataset similar to MUTA or PROTEIN. How-

ever, the stringent constraints imposed by exact vertex matching are relaxed due to the string edit distance. Thus, the matching process can be tolerant and accommodate to differences in the labels.

ILPISO [28] stands apart from the others in the sense that this dataset holds directed graphs. The aim is to illustrate the flexibility of our proposal in that it can handle different types of graphs.

LETTER [29] is broken down into three parts (LOW, MED, HIGH) which correspond to distortion levels. Assessing methods according to the noise level is an interesting viewpoint when addressing pattern recognition problems. The LETTER dataset is useful because it holds graphs that have a rather small size, which makes feasible the computations of the GED methods.

PAH and Alkane[1] are purely structural databases with no labels at all.

All of these datasets are publicly available on the IAPR TC15 website[2]. A synthesis that concerns those data is given in Table 2. To evaluate the algorithms' behaviors when the size of the problem grows, we have built subsets in which all of the graphs have the same number of vertices for GREC, MUTA, PROTEIN and ILPISO. The details that concern the subset sizes are given in Table 2. The cost functions have a parameter $\alpha \in [0, 1]$ that is domain-dependent. α corresponds to the weighting parameter that controls whether the edit operation cost on the vertices or on the edges is more important. We borrow the settings from [7] for the GREC, PROT, MUTA and LETTER databases and from [15] for the Alkane and PAH datasets. Elementary operation costs are reported in Table 2. With the goal of reproducibility, all of the graph subsets and the code of the cost function are available at `https://sites.google.com/site/blpged/`.

[1] `https://brunl01.users.greyc.fr/CHEMISTRY/index.html`

[2] `https://iapr-tc15.greyc.fr/links.html`

Table 2: Summary of the graph datasets' characteristics, subsets decomposition and cost function of the datasets – $\delta_{a,b}$ is the Kronecker Delta function

Dataset	#Graphs	Vertex labels	Edge labels	$\|V\|$	$\|E\|$	Directed	$c_{i,\epsilon} = c_{\epsilon,k}$	$c_{ij,\epsilon} = c_{\epsilon,kl}$	α	c_{ik}	$c_{ij,kl}$	#vertices (#graphs)
PAH	70	Chemical symbol	Valence	40	40.8	False	90	15	0.5	Extended Euclidean distance	$1 - \delta_{\xi(ij),\xi(kl)}$	5 (41) 10 (74) 15 (34) 20 (39)
PROT	188	x, y coordinates	Line type	16.6	17.2	False	11	1	0.75	Extended string edit distance	$1 - \delta_{\xi(ij),\xi(kl)}$	20 (15) 30 (13) 40 (22)
MUTA	50	Type and aa-sequence	Type and distance	30	62.1	False	11	11	0.25	$1 - \delta_{\mu(i),\mu(k)}$	$1 - \delta_{\xi(ij),\xi(kl)}$	10 (10) 20 (10) 70 (10)
ILIPSO	36	Scalar value	Scalar	28.3	54	True	66.6	66.6	0.5	L1 norm	L1 norm	10 (12) 25 (12) 50 (12)
LETTER	75	None	None	8.8	7.8	False	0.3	0.5	0.75	L2 norm	$1 - \delta_{\xi(ij),\xi(kl)}$	No subsets
Alkane	10	None	None	20.8	24.5	False	3	3	0.5	$1 - \delta_{\mu(i),\mu(k)}$	$1 - \delta_{\xi(ij),\xi(kl)}$	No subsets
GREC	750	x, y coordinates	None	4.7	3.9	False	3	3	0.5	$1 - \delta_{\mu(i),\mu(k)}$	$1 - \delta_{\xi(ij),\xi(kl)}$	No subsets

5.3. Experimental protocol

Our experiments were conducted in the context of graph comparisons. Let $\mathcal{S}$ be a graph dataset that consists of m graphs, $\mathcal{S} = \{G_1, G_2, ..., G_m\}$. Let $\mathcal{P} = \mathcal{P}_e \cup \mathcal{P}_a$ be the set of all of the GED methods listed in subsection 5.1, with $\mathcal{P}_e$ = {A*, JH, F1, F2} the set of exact methods and $\mathcal{P}_a$ = {BP, BS, FBP, H, F1LP, F2LP} the set of approximate methods. The parameter q of BS was set to 10. Given a method $p \in \mathcal{P}$, we computed the square distance matrix $M^p \in \mathcal{M}_{m \times m}(\mathbb{R}^+)$, which holds every pairwise comparison $M^p_{i,j} = d_p(G_i, G_j)$, where the distance $d_p(G_i, G_j)$ is the value returned by the method p on the graph pair (G_i, G_j) within a certain time limit, while using the cost metaparameters defined in Table 2. Due to the large number of matchings to be computed and the exponential complexity of the algorithms tested, we allowed a maximum of **300 seconds** for any distance computation. When the time limit was over, the best approximation found thus far was outputted by the given method. This time constraint was sufficiently large to allow the methods to search deeply into the solution space and to ensure that many tree nodes were explored. The key idea was to reach optimality whenever it was possible or at least to get as close as possible to the *Graal*, the optimal solution that corresponds to the exact distance.

Based on this context of pairwise graph comparisons, a set of metrics is defined to measure the accuracy and speed of all of the methods.

5.3.1. Accuracy metrics

To quantify the error in the approximate methods, we compute an index called the deviation. This index relies on a reference matrix that holds either the optimal GED whenever it is possible to compute it, or the lowest approximation found among all the methods. In the latter case, the lower bounds (H, F1LP and F2LP) are removed from the eligible reference methods since they do not represent feasible solutions and can not represent real sequences of edit

operations. Hence, the deviation is computed on a subset n using equation 19.

$$deviation(i,j)^p = \frac{|M_{i,j}^p - R_{i,j}|}{R_{i,j}}, \forall (i,j) \in [\![1,m]\!]^2, \forall p \in \mathcal{P} \tag{19}$$

where $R_{i,j}$ is defined in equation 20.

$$R_{i,j} = \min_{p \in \mathcal{P} \setminus \{F1LP, F2LP, H\}} \{M_{i,j}^p\},\ \forall (i,j) \in [\![1,m]\!]^2 \tag{20}$$

$R_{i,j}$ is either the best upper bound or the optimal solution when the time available allows its computation. For a given method, the deviation can express the error made by a suboptimal solution in terms of the percentage of the best solution. For each subset indexed by n, the mean deviation is derived as follows in equation 21 :

$$\overline{deviation_n^p} = \frac{1}{m \times m} \sum_{i=1}^{m} \sum_{j=1}^{m} deviation(i,j)_n^p \tag{21}$$

where $p \in \mathcal{P}$, $n \in [1, \#subsets]$ and $deviation(i,j)_n^p$ stands for the deviation computed on subset n. To obtain comparable results between the datasets, the mean deviations are normalized between $[0,1]$ using equation 22:

$$\overline{deviation\ score^p} = \frac{1}{\#subsets} \sum_{n=1}^{\#subsets} \frac{\overline{deviation_n^p}}{maxdev_n} \tag{22}$$

where $maxdev_n = \max \overline{deviation_n^p}\ \forall p \in \mathcal{P}$

5.3.2. Speed metrics

To evaluate the convergence speeds of algorithms, the mean time for each dataset is derived as follows in equation 23 :

$$\overline{time_n^p} = \frac{1}{m \times m} \sum_{i=1}^{m} \sum_{j=1}^{m} time(G_i, G_j)_n^p \tag{23}$$

with $(i,j) \in [\![1,m]\!]^2$, $n \in [1, \#subsets]$ and $time(i,j)_n^p$ stands for the computing time on subset n.

Finally, we introduce a last metric called the speed score where the running time is normalized between $[0,1]$ as follows in equation 24 :

$$\overline{speed\ score^p} = \frac{1}{\#subsets} \sum_{n=1}^{\#subsets} \frac{\overline{time_n^p}}{maxtime_n} \tag{24}$$

5.3.3. Experimental settings

In this practical work, BP was provided by the Institute of Computer Science and Applied Mathematics of Bern[3], while the other methods were re-implemented. All of the methods are implemented in Java 1.7 except for the F1, F2 and JH models which were implemented in $C\#$ using CPLEX Concert Technology. CPLEX 12.6 was chosen since it is known to be one of the best mathematical programming solvers. All of the methods were run on a 2.6 GHz quad-core computer with 8 GB RAM. For the sake of comparison, none of the methods were parallelized and CPLEX was set up in a deterministic manner.

5.4. Results

The discussion that concerns the obtained results is organized into three parts. First, we compare only exact methods. Then, we include approximate approaches in the discussion. Finally, we draw a synthesis by comparing all of the methods from a multi-objective point of view.

5.4.1. Comparing exact methods

From a theoretical point of view, the main criterion that characterizes an exact GED computation method is its computation time, i.e., its $\overline{\textit{speed score}^p}$. From a practical point of view, solving for the exact GED can be infeasible in a reasonable amount of time and can overstep the memory capacity (e.g., using A*). For the execution time, it is well admitted that a time limit can be set. When this time limit is reached, the solver returns the best solution found thus far, which is not necessarily the optimal solution. Thus, the deviation $\overline{\textit{deviation score}^p}$ is computed to characterize the accuracy of the proposed solution. When memory problems occur, the solver can not return any solution and the deviation can not be computed.

The obtained results when comparing only exact methods are presented in columns 1-4 of Table 3. Three values are given to qualify each method of $\mathcal{P}_e$

[3] `http://www.iam.unibe.ch/fki/`

on each dataset: the percentage of instances solved to optimality (without time or memory problems), the mean deviations (different from 0 when optimality is not reached for all the instances) and the running time. If the time limit of **300** seconds is reached for some instances, the first value is less than 100. If the memory limit is reached, then "OM" appears in the table. The values "NA" means that the algorithm can not be applied to the dataset. It occurs for JH on GREC, PROT and ILPISO, since JH formulation does not account for the edge labels. As expected, A* is the worst method in Table 3. It reaches the optimal solution only for datasets composed of very small graphs (Alkane, LETTER and GREC10). When the graphs are larger than 10 vertices, all of the instance are not optimally solved, and the error grows (reaching for example 30% on GREC-15). Beyond 15 vertices, A* cannot converge to optimality because of the memory saturation phenomenon. In fact, the size of the list OPEN that contains pending solutions grows exponentially according to the graph size and the bipartite heuristic fails to prune the search tree efficiently.

When analyzing the F1 and F2 results, it can be seen that for datasets composed of smaller graphs (Alkane, LETTER, GREC, MUTA10-20), F1 and F2 both converge to optimality. In these cases, F2 is faster than F1, with a factor that is between 2 and 6. For larger graphs (PAH and MUTA), the optimal solution is not always reached, because of the time restriction. As an example, on PAH, which contains graphs of up to 24 vertices, the F1 deviation is not zero and the percentage of optimal solutions is only 51%. On this dataset, F2 is faster and converges to the optimal solution for all of the instances. This configuration also occurs for GREC20. For datasets where both F1 and F2 do not solve all of the instances to optimality, the deviation is lower for F2 than for F1. The deviation gap between the two methods can reach 20% on PAH. All of these results corroborate the theoretical analysis provided in Section 4.4. One can highlight that for the largest graphs, F1 and F2 take similar amount of time. This finding can be explained by the fact that the time limit is reached for the majority of instances by both methods.

JH performs very well in terms of speed and accuracy. For datasets that

gather rather small graphs, such as LETTER, Alkane and MUTA10, all of the instances are completely solved to optimality by JH, with a computing time that is slightly worse than using F2. When the graphs are medium-size (PAH and MUTA20), JH becomes faster than F2. For MUTA-30 to MUTA-60, the number of optimal solutions drastically decreases for every method however JH could still find 50% of the optimal solutions on MUTA-50 against 11% and 10% for F2 and F1, respectively. Beyond its inability to consider edge labels, the only weakness of JH is a memory exhaustion phenomenon when the graphs hold 70 vertices. On MUTA-70, JH could not find any solution for half of the instances while F1 and F2 find some solutions. This result shows that our proposal can better scale up to large graphs than the Justice and Hero model from a memory point of view. This behavior corroborates the theoretical statement made in section 3.1, in the JH's model, the number of constraints and variables grows quadratically as a function of the number of vertices in the graphs $(|V_1|+|V_2|)^2$, and thus, it does impact considerably the memory consumption of the program.

5.4.2. Comparing approximate methods

Columns 5 to 10 in Table 3 present the accuracy and time performance that is observed for approximate methods under the experimental protocol presented above. The mean deviation and mean running time are reported for each method on each subset. For the objective of obtaining a quantitative comparison, the accuracy gaps $(\overline{deviation^{p_i}} - \overline{deviation^{p_j}})$ and time ratio $(\overline{time^{p_i}} / \overline{time^{p_j}})$ between the methods are tabulated in Table 4. When examining the mean deviation of the methods in $\mathcal{P}_a$ = {BP, BS, FBP, H, F1LP, F2LP}, the main observation is that F1LP and F2LP are by far the most accurate approximate methods for the computation of the GED (see Table 3 and Table 4). More precisely, F2LP is 4.3% more accurate than F1LP. Because F2LP contains fewer variables and fewer constraints than F1LP, this observation confirms that the time complexity for solving a linear program increases with the number of variables and the number of constraints. F2LP is also 12.1% more accurate than BS which is the most accurate of the state-of-the-art approximate methods.

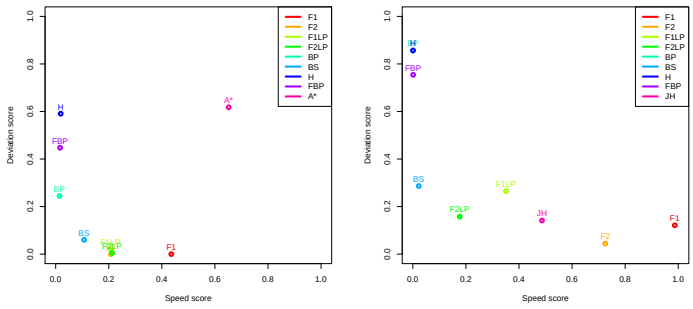
A finer examination according to the type of dataset leads us to conclude that the smaller the graphs are, the better the approximation. For small attributed graphs (LETTER, GREC) F2LP leads to a near-optimal approximation and achieves a very low or null deviation. On GREC, the error committed is less than 1%. The deviation of F2LP increases with the size of the graph. It can reach 25% on PAH or MUTA but remains lower than any other approximate method.

As it was the case for exact methods, we can also note that the datasets that are composed of graphs without any label, such as PAH or Alkane, are very difficult to process. However, all of the instances were solved by approximate methods within the time limit of 300 s. We can then conclude that each method returned its best attainable approximation.

As mentioned earlier, lower bounds are also used by branch-and-cut techniques to prune the search space when solving optimization problems. The good performance of F2LP contributes to explaining the convergence speed of F2 over F1 as it prunes the search space more efficiently.

Table 3 also reports the average time to compute a graph comparison for each method and each subset. All of the approximate methods are faster to compute than any of the exact methods. Among the approximate methods, H, BP and FBP are by far the fastest methods. They provide comparable speed results. Any instance of our datasets is solved by these methods in less than one second. The advantage of FBP over BP does not appear clearly. It can be explained by our experiments protocol. The graphs to be compared have comparable size on 14 subsets. When $|V_1| \simeq |V_2|$, this case corresponds to the FBP worst case because a cost matrix of size $|V_1| \times |V_2|$ must be processed.

The speed gap with other approximate methods (BS, F1LP and F2LP) increases with the size of the graphs. It reaches a factor of 5000 on MUTA-70 between H and F2LP. As conjectured in 4.4, F2LP is faster to compute than F1LP, mainly because the former contains fewer variables and constraints.



(a) GREC (b) MUTA

Figure 1: A synthesis on deviation and response time. Lowest values are the best.

5.4.3. Synthesis

As expected, it can be observed that the exact methods require more time than the approximate methods and that the fast methods often lead to a rough approximation. To illustrate this compromise, Figure 1 presents the speed-deviation performance of each method for the GREC and MUTA. JH could not be applied on GREC because it does not handle labels on edges. On the other hand, A* does not appear on MUTA because every instance has led to an exhaustion of available memory.

We can notice that F1 and A* are dominated methods since they do not outperform any other method on either deviation or speed criterion. Other reviewed methods correspond to a compromise between speed and accuracy. The choice of a method is application-dependent, if accuracy is a matter then F2 is the most appropriate choice. At the opposite, FBP and BP are the fastest in our benchmark. In between, methods provide speed/accuracy trade-offs. The results reported in Table 4 corroborate the aforementioned observations.

Generally, our models appear to be quite accurate and outperform in this way other methods from the literature. F2 outperforms the other methods on all of the datasets in terms of the accuracy. The most spectacular improvement is an average gain of 45% against BP, FBP and H. The average deviation gap

between F2 and F2LP is only 6% whereas it is 18% between F2 and BS, but for BS, it is 5 times faster. F2LP is 12% more accurate than BS.

Finally, we can also state that F1LP and F2LP are not only approximately 7 times slower than BP, FBP and H but also approximately 40% more accurate. Numerical results are detailed in Table 4.

6. Conclusions

In this paper, two exact BLP formulations of the GED problem have been presented. The first formulation F1 is a didactic expression of the GED problem, while F2 is a more refined program where variables and constraints have been condensed to reduce the search space. Two lower bound approximations of the GED (F1LP and F2LP) have been derived from the exact formulations, using the continuous relaxation technique. Theoretically and practically speaking, we showed that F2LP is faster to compute than F1LP. Formulations were evaluated on publicly available databases. In all cases, F1 and F1LP were slower and less accurate than F2 and F2LP, respectively. This result experimentally validated F2 and the choice of reducing the number of variables and constraints of F1. The experimental comparison with the state-of-the-art methods showed that, among the exact methods that operate under a time constraint, F2 and JH were the most accurate. When applicable, JH was generally faster, however, it faces excessive memory needs when the sizes of the graphs grow. The experiments also showed that the continuous relaxation of the domain constraints leads to very accurate approximate methods. F2LP is the most accurate approximate method of our benchmark while it holds a comparative runtime with BS. It is however slower than BP, FBP and H. Finally, the last point is the generality of our proposal. Our formulations are the most general in the sense that they can handle different types of attributed relational graphs: directed or undirected graphs, simple graphs or multigraphs, with a combination of symbolic, numeric and/or string attributes on the vertices and edges. F2 and F2LP were part of the Graph Distance contest which was organized in the context of the ICPR

Table 3: Deviation and running time of all of the methods. **Opt**:: Percentage of instances solved to optimality. Mean deviation in % and mean running time in milliseconds. The lower the better. **OM** stands for Out of Memory. **NA** means not applicable. Blue: The best deviations. Red: The best running times. In **bold**, the best deviations among the approximate methods.

	F1			F2			A*			JH			BP		BS		H		FBP		F1LP		F2LP	
	Opt	Dev	Time	Opt	Dev	Time	Opt	Dev	Time	Opt	Dev	Time	Dev	Time	Dev	Time	Dev	Time	Dev	Time	Dev	Time	Dev	Time
PAH	51	19.4	194428	100	0	36444	OM	OM	OM	100	0	1529	398	1.4	99	139	79	1	370	0	31.2	4858	**27.4**	1403
Alkane	100	0	176	100	0	39	100	0	8720	100	0	165	133	0.2	**14**	4	78	0.4	111	0	42.4	138	18.7	38
LETTER HIGH	100	0	43	100	0	9	100	0	3	100	0	13	8.9	0	4.5	1	21.5	0	24	0	0.1	26	**0**	8
LETTER MED	100	0	8	100	0	5	100	0	1	100	0	10	13.2	0	2.3	1	17.2	0	27	0	**0**	8	**0**	5
LETTER LOW	100	0	8	100	0	5	100	0	1	100	0	10	9.6	0	2.5	1	21.8	0	28	0	**0**	8	**0**	5
GREC 5	100	0	4	100	0	4	100	0	3	NA	NA	NA	1	0	0.1	2	5.8	0	5.9	0	**0**	4	**0**	4
GREC 10	100	0	81	100	0	34	90.1	4	36831	NA	NA	NA	4	1	1	34	6.8	2	4.7	1	0.2	81	**0.1**	34
GREC 15	100	0	3192	100	0	547	42.5	31.1	172886	NA	NA	NA	8.8	2	2.1	245	12.1	4	8.8	2	1.4	607	**0.3**	372
GREC 20	99.5	0	12347	100	0	981	OM	OM	OM	NA	NA	NA	5	4	3.1	908	26.9	8	5.2	4	1.9	1171	**0.3**	642
MUTA 10	100	0	128	100	0	50	OM	OM	OM	100	0	82	24.2	1	2.1	6	50.8	1	23	1	2	128	**0.3**	50
MUTA 20	100	0	25696	100	0	4156	OM	OM	OM	100	0	946	47.1	1	13.3	97	58.3	2	39.2	1	14.9	1472	**6.2**	1024
MUTA 30	16	4.8	265237	52	1.4	185830	OM	OM	OM	86	0	72534	79.2	2	23.6	519	68	4	66.5	1	22.6	8010	**14.9**	4891
MUTA 40	11	10.2	270076	27	3.4	236786	OM	OM	OM	68	0	133974	82.2	5	29.3	1620	68.1	7	71.8	5	23.2	21917	**14.1**	11760
MUTA 50	10	23.2	270461	11	8.2	268795	OM	OM	OM	50	0	193869	98.2	8	44.1	3963	77.8	12	87.4	8	31.6	43945	**18.8**	22630
MUTA 60	10	23.1	271625	10	9.2	270169	OM	OM	OM	19	0	249685	92.9	12	35.8	8198	74.2	17	82.2	11	31.9	95714	**18.8**	43822
MUTA 70	10	18.8	273124	10	7.1	270359	OM	OM	OM	OM	OM	OM	78.9	18	27.2	15773	79.9	23	73	17	35.3	212545	**25.2**	97304
PROT 20	83.1	0	80070	94.6	0	35571	OM	OM	OM	NA	NA	NA	3.2	13	1.8	3203	12.7	25	12.6	13	1.2	4756	**0.8**	2179
PROT 30	37.9	0.1	217330	74	0	130134	OM	OM	OM	NA	NA	NA	5.7	27	1.8	17743	19.1	54	68.6	28	1	11695	**0.7**	5672
PROT 40	10.9	0.2	273159	21.5	0	255382	OM	OM	OM	NA	NA	NA	3.5	49	2	50737	13.3	97	18.1	52	1.9	58087	**1.2**	21805
ILPISO 10	100	0	23	100	0	22	OM	OM	OM	NA	NA	NA	16.6	2	12.3	66	39.8	2	17	2	**0**	22	**0**	21
ILPISO 25	91.6	0	31001	91	0.2	35410	OM	OM	OM	NA	NA	NA	33.4	18	26.1	11053	54.6	33	35.3	19	7	14367	**4.5**	4259
ILPISO 50	56.9	6.6	138887	55.5	2.7	141750	OM	OM	OM	NA	NA	NA	24.1	295	80.1	260665	59.4	557	25.8	310	8.4	125938	**9.4**	114598

Table 4: Relative comparison of all of the methods according to their running time and accuracy. **Dev gap**: Mean deviation gap between the methods in terms of the percentage over all of the databases. The higher the value is, the better the outcome. **Time ratio**: Mean time ratio between the methods over all of the databases. The second line is , for example, the time of method x on F2. The lower the value is, the better the outcome. Blue: Comparison with the best deviations. Red: Comparison with the best running time.

	F1		F2		BP		BS		H		FBP		F1LP		F2LP	
	Dev gap	Time ratio	Dev gap	Time ratio	Dev gap	Time ratio	Dev gap	Time ratio	Dev gap	Time ratio	Dev gap	Time ratio	Dev gap	Time ratio	Dev gap	Time ratio
F1	0	1	-3.3	1.2	48.4	5063.3	14.7	6.2	38.2	2739.7	50	4897.1	6.9	3.8	2.6	7
F2	3.3	0.8	0	1	51.7	4074.2	18	5	41.5	2204.5	53.3	3940.4	10.2	3.1	5.9	5.6
BP	-48.4	0	-51.7	0	0	1	-33.7	0	-10.2	0.5	1.6	1	-41.5	0	-45.8	0
BS	-14.7	0.2	-18	0.2	33.7	815.9	0	1	23.5	441.5	35.3	789.1	-7.8	0.6	-12.1	1.1
H	-38.2	0	-41.5	0	10.2	1.8	-23.5	0	0	1	11.8	1.8	-31.3	0	-35.6	0
FBP	-50	0	-53.3	0	-1.6	1	-35.3	0	-11.8	0.6	0	1	-43.1	0	-47.4	0
F1LP	-6.9	0.3	-10.2	0.3	41.5	1317.6	7.8	1.6	31.3	712.9	43.1	1274.3	0	1	-4.3	1.8
F2LP	-2.6	0.1	-5.9	0.2	45.8	723.7	12.1	0.9	35.6	391.6	47.4	699.9	4.3	0.5	0	1

2016 conference [4]. In perspective, quadratic programming solvers are obtaining more and more efficiency and we want to investigate the definition of binary quadratic programming formulations of the GED problem. Finally, another interesting task for future research will be to use lower and upper bounds to build an optimized nearest neighbor search.

References

[1] K. Riesen, M. Neuhaus, H. Bunke, Graph Embedding in Vector Spaces by Means of Prototype Selection, in: Proc. of the 6th IAPR Int. Workshop on Graph-based Repr. in Pattern Recogn., 383–393, 2007.

[2] H. Bunke, K. Riesen, Towards the unification of structural and statistical pattern recognition, Pattern Recogn. Lett. 33 (7) (2012) 811–825.

[3] S. Kramer, L. D. Raedt, Feature Construction with Version Spaces for Biochemical Applications, in: Proc. of the 18th ICML, 258–265, 2001.

[4] N. Sidère, P. Héroux, J.-Y. Ramel, Vector Representation of Graphs: Application to the Classification of Symbols and Letters, in: Proc. of the Int. Conf. on Doc. Anal. and Recogn., 681–685, 2009.

[5] P. Ren, R. Wilson, E. Hancock, Graph Characterization via Ihara Coefficients, IEEE Trans. on Neural Networks 22 (2) (2011) 233–245.

[6] J. Shi, J. Malik, Normalized Cuts and Image Segmentation, IEEE Trans. on PAMI 22 (8) (2000) 888–905.

[7] K. Riesen, H. Bunke, Graph Classification and Clustering Based on Vector Space Embedding, World Scientific Publishing Co., Inc., 2010.

[8] T. Gärtner, J. Lloyd, P. Flach, Kernels for Structured Data, in: Inductive Logic Programming, vol. 2583 of *LNCS*, Springer, 66–83, 2003.

[4] https://gdc2016.greyc.fr/

[9] P. Foggia, M. Vento, Graph Embedding for Pattern Recognition, in: Recognizing Patterns in Signals, Speech, Images and Videos, vol. 6388 of *LNCS*, Springer, 75–82, 2010.

[10] D. Raviv, R. Kimmel, A. M. Bruckstein, Graph Isomorphisms and Automorphisms via Spectral Signatures, IEEE Trans. on PAMI 35 (8) (2013) 1985–1993.

[11] S. Umeyama, An Eigendecomposition Approach to Weighted Graph Matching Problems, IEEE Trans. on PAMI 10 (5) (1988) 695–703.

[12] H. Bunke, G. Allermann, Inexact Graph Matching for Structural Pattern Recognition, Pattern Recogn. Lett. 1 (4) (1983) 245–253.

[13] H. Bunke, P. Dickinson, M. Kraetzl, M. Neuhaus, M. Stettler, Matching of Hypergraphs — Algorithms, Applications, and Experiments, in: Applied Pattern Recognition, vol. 91 of *Studies in Computational Intelligence*, Springer, 131–154, 2008.

[14] O. Kostakis, Classy: fast clustering streams of call-graphs, Data Mining and Knowledge Discovery 28 (5) (2014) 1554–1585.

[15] B. Gaüzère, L. Brun, D. Villemin, Two new graphs kernels in chemoinformatics, Pattern Recogn. Lett. 33 (15) (2012) 2038 – 2047.

[16] A. Fischer, H. Bunke, Character prototype selection for handwriting recognition in historical documents, in: Proc. of the 19th Europ. Signal Processing Conference, 1435–1439, 2011.

[17] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, L. Zhou, Comparing Stars: On Approximating Graph Edit Distance., PVLDB 2 (1) (2009) 25–36, URL `http://dblp.uni-trier.de/db/journals/pvldb/pvldb2.html#ZengTWFZ09`.

[18] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., NY, USA, 1979.

[19] K. Riesen, S. Fankhauser, H. Bunke, Speeding Up Graph Edit Distance Computation with a Bipartite Heuristic, in: Mining and Learning with Graphs, MLG 2007 Proceedings, 2007.

[20] A. Fischer, R. Plamondon, Y. Savaria, K. Riesen, H. Bunke, A Hausdorff Heuristic for Efficient Computation of Graph Edit Distance, in: Proc. of the Joint IAPR Int. Workshops, S+SSPR 2014, 83–92, 2014.

[21] A. Fischer, C. Y. Suen, V. Frinken, K. Riesen, H. Bunke, Approximation of graph edit distance based on Hausdorff matching, Pattern Recogn. 48 (2) (2015) 331 – 343.

[22] S. Fankhauser, K. Riesen, H. Bunke, P. J. Dickinson, Suboptimal Graph Isomorphism using bipartite Matching, IJPRAI 26 (6).

[23] K. Riesen, H. Bunke, Approximate graph edit distance computation by means of bipartite graph matching, Image Vision Comput. 27 (7) (2009) 950–959.

[24] M. Neuhaus, K. Riesen, H. Bunke, Fast Suboptimal Algorithms for the Computation of Graph Edit Distance, in: Proc. of the Joint IAPR International Workshops, SSPR & SPR 2006, 163–172, 2006.

[25] R. Myers, R. C. Wilson, E. R. Hancock, Bayesian Graph Edit Distance, IEEE Trans. on PAMI 22 (6) (2000) 628–635.

[26] R. Raveaux, J.-C. Burie, J.-M. Ogier, A graph matching method and a graph matching distance based on subgraph assignments, Pattern Recogn. Lett. 31 (5) (2010) 394–406.

[27] D. Justice, A. Hero, A Binary Linear Programming Formulation of the Graph Edit Distance, IEEE Trans. on PAMI 28 (8) (2006) 1200–1214.

[28] P. Héroux, P. Le Bodic, S. Adam, Datasets for the Evaluation of Substitution-Tolerant Subgraph Isomorphism, in: Graphics Recognition. Current Trends and Challenges, LNCS, Springer, 240–251, 2014.

[29] K. Riesen, H. Bunke, IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning, in: Proc. of the Joint IAPR Workshops S+SSPR, vol. 5342 of *LNCS*, Springer, 287–297, 2008.

[30] IAPR TC 15, GREYC Datasets, URL `https://iapr-tc15.greyc.fr/links.html`, 2013.

[31] H. Bunke, On a relation between graph edit distance and maximum common subgraph, Pattern Recogn. Lett. 18 (8) (1997) 689–694.

[32] H. Bunke, K. Shearer, A Graph Distance Metric Based on the Maximal Common Subgraph, Pattern Recogn. Lett. 19 (3-4) (1998) 255–259.

[33] M. Neuhaus, H. Bunke, Bridging the Gap between Graph Edit Distance and Kernel Machines, vol. 68 of *Series in Machine Perception and Artificial Intelligence*, WorldScientific, 2007.

[34] X. Cortés, F. Serratosa, Learning graph-matching edit-costs based on the optimality of the oracle's node correspondences, Pattern Recogn. Lett. 56 (2015) 22–29.

[35] T. S. Caetano, J. J. McAuley, L. Cheng, Q. V. Le, A. J. Smola, Learning Graph Matching, IEEE Trans. on PAMI 31 (6) (2009) 1048–1058.

[36] X. Gao, B. Xiao, D. Tao, X. Li, A Survey of Graph Edit Distance, Pattern Anal. Appl. 13 (1) (2010) 113–129.

[37] K. Riesen, Structural Pattern Recognition with Graph Edit Distance - Approximation Algorithms and Applications, Advances in Comp. Vis. and Pattern Recogn., Springer, 2015.

[38] P. Hart, N. Nilsson, B. Raphael, A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE Trans. on SMC 4 (2) (1968) 100–107.

[39] H. A. Almohamad, S. O. Duffuaa, A Linear Programming Approach for the Weighted Graph Matching Problem, IEEE Trans. on PAMI 15 (5) (1993) 522–525.

[40] J. Munkres, Algorithms for the Assignment and Transportation Problems, J. of the Society of Industrial and Applied Mathematics 5 (1) (1957) 32–38.

[41] K. Riesen, H. Bunke, Improving bipartite graph edit distance approximation using various search strategies, Pattern Recogn. 48 (4) (2015) 1349–1363.

[42] L. A. Wolsey, Integer programming, Wiley-Interscience, NY, USA, 1998.

[43] R. Saigal, Linear Programming: A Modern Integrated Analysis, Kluwer Academic Publishers, 1995.

[44] F. Serratosa, Fast computation of Bipartite graph matching, Pattern Recogn. Lett. 45 (2014) 244–250.

[45] F. Serratosa, Computation of graph edit distance: Reasoning about optimality and speed-up, Image Vision Comp. 40 (2015) 38–48.