Exact Graph Edit Distance Computation Using a Binary Linear Program

Julien Lerouge[1], Zeina Abu-Aisheh[2], Romain Raveaux[2], Pierre Héroux[1], and Sébastien Adam[1](✉)

[1] Normandie Univ, UNIROUEN, UNIHAVRE, INSA Rouen, LITIS, 76000 Rouen, France
Sebastien.Adam@univ-rouen.fr
[2] LI Tours, Avenue Jean Portalis, Tours, France

Abstract. This paper presents a binary linear program which computes the exact graph edit distance between two richly attributed graphs (i.e. with attributes on both vertices and edges). Without solving graph edit distance for large graphs, the proposed program enables to process richer and larger graphs than existing approaches based on mathematical programming and the A^* algorithm. Experiments are led on 7 standard graph datasets and the proposed approach is compared with two state-of-the-art algorithms.

Keywords: Graph edit distance · Binary linear program

1 Introduction

Computing the dissimilarity between two graphs is a crucial issue for graph-based pattern recognition problems, e.g. malware detection [10], chemoinformatics [5], or document analysis [2]. A large number of algorithms are proposed in the literature to compute graph dissimilarity. Among existing approaches, Graph Edit Distance (GED) has retained a lot of attention during the two last decades. Using GED, graph dissimilarity computation is directly linked to a matching process through the introduction of a set of graph edit operations (e.g. vertex insertion, vertex deletion). Each edit operation being characterized by a cost, the GED is the total cost of the least expensive sequence of edit operations that transforms one graph into the other one. A major theoretical advantage of GED is that it is a dissimilarity measure for arbitrarily structured and arbitrarily attributed graphs. Moreover, together with the dissimilarity, it provides the corresponding sequence of edit operations, which can be itself useful for application purpose. A limitation for the use of GED is its computational complexity. Indeed, as stated in [22], the GED problem is NP-hard. This explains why many recent contributions focus on the computation of GED approximations which can be applied on large graphs. Among existing approximations, some are based on the proposition of new heuristics to improve the performance of exact approaches [3,21] whereas others propose faster but suboptimal methods which approximate the

A. Robles-Kelly et al. (Eds.): S+SSPR 2016, LNCS 10029, pp. 485–495, 2016.
DOI: 10.1007/978-3-319-49055-7_43

exact GED (e.g. [1,4,14–16,18,20]). These latter are faster, but do not guarantee to find the optimal solution.

In this paper, we are interested in the exact computation of the GED between two graphs, for a given set of edit costs. In this context, the main family of existing approaches is based on the widely known A* algorithm [3,21]. This algorithm relies on the exploration of the tree of solutions. In this tree, a node corresponds to a partial edition of the input graph towards the target one. A leaf of the tree corresponds to a complete edit path which transforms one graph into the other. The exploration of the tree is led by developing the most promising ways on the basis of an estimation of GED. For each node, this estimation is the sum of the cost associated to 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 [7]. 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 approximation quality and their computation time [3,21].

A second family of algorithms consists in using Binary Linear Programing (BLP) for computing the GED. Justice and Hero [9] proposed a BLP formulation of the GED problem. The proposed program searches for the permutation matrix which minimizes the cost of transforming G_1 into G_2, with G_1 and G_2 two unweighted and undirected attributed graphs. The criterion to be minimized takes into account costs for matching vertices, but the formulation does not integrate the ability to process graphs with labels on their edges.

In some previous works, we also have investigated the use of Integer Linear Programing for graph matching problems. In [11,13], a formulation and a toolbox are proposed for substitution-tolerant subgraph isomorphism and its use for symbol spotting in technical drawings. In [12], a BLP-based minimum cost subgraph matching is described. It has been used for document analysis purpose in [6]. In this paper, we propose an extension of [12] for GED computation. The proposed framework can be used to compute the distance between richly attributed graphs (i.e. with attributes on both vertices and edges) which can not be tackled using the BLP formulation proposed in [9]. Experiments led on reference datasets show that the new BLP formulation can compute an exact GED on larger graphs than A* based algorithm.

This paper is organized as follows: Sect. 2 presents the important definitions necessary for introducing our formulations of the GED. Then, Sect. 3 describes the proposed binary linear programming formulation. Section 4 presents the experiments and analyses the obtained results. Section 5 provides some concluding remarks.

2 Definitions

In this paper, we are interested in computing GED between attributed graphs. This section is dedicated to the introduction of the notations and definitions used in the remaining of the paper.

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 $\forall e = ij \in E, i \in V$ and $j \in V$,*
- *$\mu : V \to L_V$ is a vertex labeling function which associates the label $\mu(v)$ to all vertices v of V, where L_V is the set of possible labels for the vertices,*
- *$\xi : E \to L_E$ is an edge labeling function which associates the label $\xi(e)$ to all edges e of E, where L_E is the set of possible labels for the edges.*

The vertices (resp. edges) label space L_V (resp. L_E) may be composed of any combination of numeric, symbolic or string attributes.

Definition 2. *The graph edit distance $d(.,.)$ is a function*

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

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 $o = (o_1, \ldots, o_k)$ allowing to transform G_1 into G_2. An elementary edit operation o_i is one of vertex substitution ($v_1 \to v_2$), edge substitution ($e_1 \to e_2$), vertex deletion ($v_1 \to \epsilon$), edge deletion: ($e_1 \to \epsilon$), vertex insertion ($\epsilon \to v_2$) and edge insertion ($\epsilon \to e_2$) with $v_1 \in V_1$, $v_2 \in V_2$, $e_1 \in E_1$ and $e_2 \in E_2$. ϵ is a dummy vertex or edge which is used to model insertion or deletion. $c(.)$ is a function which associates a cost to each elementary edit operation o_i.

In the more general case where GED is computed between attributed graphs, edit costs are generally defined as functions of 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 linked to the attributes of the inserted/deleted vertex (resp. edge):

$$\begin{aligned} &c(v_1 \to v_2) = c(v_2 \to v_1) = f_v(\mu_1(v_1), \mu_2(v_2)) \\ &c(e_1 \to e_2) = c(e_2 \to e_1) = f_e(\xi_1(e_1), \xi_2(e_2)) \\ &c(v \to \epsilon) = c(\epsilon \to v) = g_v(\mu(v)) \\ &c(e \to \epsilon) = c(\epsilon \to e) = g_e(\xi(e)) \end{aligned}$$

3 BLP Formulation of GED

Binary Linear Programing is a restriction of integer linear programming (ILP) with binary variables. Its general form is :

$$\min_x c^T x \tag{1a}$$

$$\text{subject to } Ax \leq b \tag{1b}$$

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

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 (1c) which respects linear inequality constraints (1b). The constraint (1c) which enumerates the admissible values of variables is called a domain constraint. If the program has at least a feasible solution, then the optimal solutions are the ones that minimize the objective function (1a) which is a linear combination of variables of x weighted by the components of the vector c.

In this section, we successively define the variables, the objective function and the linear constraint of the BLP used for formulating GED as a BLP.

3.1 BLP-GED Variables

Our goal is to compute 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 the rest of this section, for the sake of simplicity of notations, we consider that the graphs G_1 and G_2 are simple directed graphs. However, the formulations given in this section can be applied directly without modification to multigraphs. Besides, the extension to the undirected case only needs some slight modifications.

In the GED definition provided in Sect. 2, the edit operations that are allowed to match 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) of G_2 in G_1. For each type of edit operation, we define a set of corresponding binary variables:

- $\forall (i,k) \in V_1 \times V_2, x_{i,k} = \begin{cases} 1 \text{ if } i \text{ is substituted with } k, \\ 0 \text{ otherwise.} \end{cases}$
- $\forall (ij,kl) \in E_1 \times E_2, y_{ij,kl} = \begin{cases} 1 \text{ if } ij \text{ is substituted with } kl, \\ 0 \text{ otherwise.} \end{cases}$
- $\forall i \in V_1, u_i = \begin{cases} 1 \text{ if } i \text{ is deleted from } G_1 \\ 0 \text{ otherwise.} \end{cases}$
- $\forall ij \in E_1, e_{ij} = \begin{cases} 1 \text{ if } ij \text{ is deleted from } G_1 \\ 0 \text{ otherwise.} \end{cases}$
- $\forall k \in V_2, v_k = \begin{cases} 1 \text{ if } k \text{ is inserted in } G_1 \\ 0 \text{ otherwise.} \end{cases}$
- $\forall kl \in E_2, f_{kl} = \begin{cases} 1 \text{ if } kl \text{ is inserted in } G_1 \\ 0 \text{ otherwise.} \end{cases}$

Using these notations, we define an edit path between G_1 and G_2 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 $\mathbf{f} = (f_{kl})_{kl\in E_2}$.

In order to evaluate the global cost of an edit path, elementary costs for each edit operation must be defined. We adopt the following notations for these costs:

- $\forall (i,k) \in V_1 \times V_2, c(i \to k)$ is the cost of substituting the vertex i with k,
- $\forall (ij,kl) \in E_1 \times E_2, c(ij \to kl)$ is the cost of substituting the edge ij with kl,

– $\forall i \in V_1, c(i \to \epsilon)$ is the cost of deleting the vertex i from G_1,
– $\forall ij \in E_1, c(ij \to \epsilon)$ is the cost of deleting the edge ij from G_1,
– $\forall k \in V_2, c(\epsilon \to k)$ is the cost of inserting the vertex k in G_1,
– $\forall kl \in E_2, c(\epsilon \to kl)$ is the cost of inserting the edge kl in G_1.

3.2 Objective Function

The objective function is the overall cost induced by applying an edit path $(\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v}, \mathbf{e}, \mathbf{f})$ that transforms a graph G_1 into G_2, using the elementary costs defined above. GED between G_1 and G_2 is the minimum value of the objective function (2) over $(\mathbf{x}, \mathbf{y}, \mathbf{u}, \mathbf{v}, \mathbf{e}, \mathbf{f})$ subject to constraints detailed in Sect. 3.3.

$$\begin{aligned} d(G_1, G_2) = \min_{\mathbf{x},\mathbf{y},\mathbf{u},\mathbf{v},\mathbf{e},\mathbf{f}} \Bigg(& \sum_{i \in V_1} \sum_{k \in V_2} c(i \to k) \cdot x_{i,k} + \sum_{ij \in E_1} \sum_{kl \in E_2} c(ij \to kl) \cdot y_{ij,kl} \\ & + \sum_{i \in V_1} c(i \to \epsilon) \cdot u_i + \sum_{k \in V_2} c(\epsilon \to k) \cdot v_k \\ & + \sum_{ij \in E_1} c(ij \to \epsilon) \cdot e_{ij} + \sum_{kl \in E_2} c(\epsilon \to kl) \cdot f_{kl} \Bigg) \end{aligned} \tag{2}$$

3.3 Constraints

The constraints presented in this part are designed to guarantee that the admissible solutions of the BLP are edit paths that transform G_1 in a graph which is isomorphic to G_2. An edit path is considered as admissible if and only if the following conditions are respected:

1. 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. The remaining vertices are either deleted or inserted,
2. 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. The remaining edges are either deleted or inserted,
3. The vertices matchings and the edges matchings are consistent, i.e. the graph topology is respected.

The following paragraphs describe the linear constraints used to integrate these conditions into the BLP.

1 - Vertices Matching Constraints. The constraint (3) ensures that each vertex of G_1 is either matched to exactly one vertex of G_2 or deleted from G_1, while the constraint (4) ensures that each vertex of G_2 is either matched to exactly one vertex of G_1 or inserted in G_1:

$$u_i + \sum_{k \in V_2} x_{i,k} = 1 \quad \forall i \in V_1 \tag{3}$$

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

2 - Edges Matching Constraints. Similar to the vertex matching constraints, the constraints (5) and (6) guarantee a valid mapping between the edges:

$$e_{ij} + \sum_{kl \in E_2} y_{ij,kl} = 1 \quad \forall ij \in E_1 \tag{5}$$

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

3 - Topological Constraints. In order to respect the graph topology in the matching, an edge $ij \in E_1$ can be matched to an edge $kl \in E_2$ only if the head vertices $i \in V_1$ and $k \in V_2$, on the one hand, and if the tail vertices $j \in V_1$ and $l \in V_2$, on the other hand, are respectively matched. This quadratic constraint can be expressed linearly with the following constraints (7) and (8):

– ij and kl can be matched if and only if their head vertices are matched:

$$y_{ij,kl} \le x_{i,k} \quad \forall (ij, kl) \in E_1 \times E_2 \tag{7}$$

– ij and kl can be matched if and only if their tail vertices are matched:

$$y_{ij,kl} \le x_{j,l} \quad \forall (ij, kl) \in E_1 \times E_2 \tag{8}$$

Equations 2 to 8, coupled with domain constraints which ensure that the solution is binary leads to our BLP formulation :

(BLP-GED)

$$\min_{\mathbf{x},\mathbf{y},\mathbf{u},\mathbf{v},\mathbf{e},\mathbf{f}} \Bigg(\sum_{i \in V_1} \sum_{k \in V_2} c(i \to k) \cdot x_{i,k} + \sum_{ij \in E_1} \sum_{kl \in E_2} c(ij \to kl) \cdot y_{ij,kl} + \sum_{i \in V_1} c(i \to \epsilon) \cdot u_i + \sum_{k \in V_2} c(\epsilon \to k) \cdot v_k + \sum_{ij \in E_1} c(ij \to \epsilon) \cdot e_{ij} + \sum_{kl \in E_2} c(\epsilon \to kl) \cdot f_{kl} \Bigg) \tag{9a}$$

$$\text{subject to} \quad u_i + \sum_{k \in V_2} x_{i,k} = 1 \quad \forall i \in V_1 \tag{9b}$$

$$v_k + \sum_{i \in V_1} x_{i,k} = 1 \quad \forall k \in V_2 \tag{9c}$$

$$e_{ij} + \sum_{kl \in E_2} y_{ij,kl} = 1 \quad \forall ij \in E_1 \tag{9d}$$

$$f_{kl} + \sum_{ij \in E_1} y_{ij,kl} = 1 \quad \forall kl \in E_2 \tag{9e}$$

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

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

$$\text{with} \quad x_{i,k} \in \{0,1\} \quad \forall (i,k) \in V_1 \times V_2 \tag{9h}$$

$$y_{ij,kl} \in \{0,1\} \quad \forall (ij, kl) \in E_1 \times E_2 \tag{9i}$$

$$u_i \in \{0,1\} \quad \forall i \in V_1 \tag{9j}$$

$$v_k \in \{0,1\} \quad \forall k \in V_2 \tag{9k}$$

$$e_{ij} \in \{0,1\} \quad \forall ij \in E_1 \tag{9l}$$

$$f_{kl} \in \{0,1\} \quad \forall kl \in E_2 \tag{9m}$$

4 Experiments

In this section, we present some experimental results obtained using the proposed formulation, compared with those of two reference methods. The first one is based on the A* algorithm with a bipartite heuristic [21]. This method is the most well-known exact GED method and is often used to evaluate the accuracy of approximate methods. The second exact method is the BLP proposed by Justice and Hero in [9]. This method, called JH in the paper, is directly linked to our proposal. Since this method cannot deal with edge attributes, we could not perform JH on all our datasets.

In this practical work, our method has been implemented in *C#* using CPLEX Concert Technology. All 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.

The 3 methods are evaluated on 7 reference datasets:

– **GREC** [17] is composed of undirected graphs of rather small size (i.e. up to 20 vertices in our experiments). In addition, continuous attributes on vertices and edges play an important role in the matching procedure. Such graphs are representative of pattern recognition problems where graphs are involved in a classification stage.
– **PROTEIN** [17] is a molecule dataset. In similar datasets, vertices are labeled with atomic chemical elements, which imposes a stringent constraint of label equality in the matching process. In this particular dataset, vertices are labeled with chemical element sequences. The stringent constraint can be relaxed

thanks to the string edit distance. So the matching process can be tolerant and accommodate with differences on labels.

– **ILPISO** [8] stands apart from the others in the sense that this dataset hold directed graphs. The aim is to illustrate the flexibility of our proposal that can handle different types of graphs.
– **LETTER** [17] is broken down into three parts (LOW, MED, HIGH) which corresponds to distortion levels. The LETTER dataset is useful because it holds graphs of rather small size (maximum of 9 vertices), what makes feasible the computations of all GED methods.
– **PAH**[1] is a purely structural database with no labels at all. PAH is a hard dataset gathering graphs of rather large size ($\geq 20 \quad vertices$). Graphs are complex and hard to match in cases where neighborhoods and attributes do not allow easily to differentiate between vertices.

All these datasets are publicly available on IAPR TC15 website[2]. In order to evaluate the algorithms behaviours when the size of the problem grows, we have built subsets where all graphs have the same number of vertices for GREC, PROTEIN, ILPISO datasets. Concerning edit costs, we borrow the setting from [19] for GREC, PROT, and LETTER databases.

In order to evaluate an exact GED computation method, the natural criterion is the computation time. However, from a practical point of view, exact GED solving can be infeasible in a reasonable time or can overstep memory capacity. In our framework, if a time limit of **300** s is reached, the instance is considered as unsolved. The same situation arises when the memory limit of 1 Gb is reached.

Obtained results when comparing the 3 methods are presented in Table 1. Two values are given to qualify each method on each dataset: the percentage of instances solved to optimality (without time or memory problems), called "Opt" and the average running time in milliseconds called "Time". In the objective of a fair comparison, the average running time is calculated only from instances solved to optimality by all the methods. If the memory limit is reached, "OM" appears in the table. The "NA" values mean that the algorithm can not be applied to the dataset. It occurs for the JH method on GREC, PROT and ILPISO dataset since JH formulation does not take into account edge labels.

Some observations can be made from these results. First, as expected, A* is the worst method. It reaches the optimal solution only for datasets composed of very small graphs (LETTER and GREC10). A* is the fastest method for small graphs. When graphs are larger than 10 vertices, all the instances are not optimally solved. When graphs hold 10 vertices, the execution time increase considerably by a factor 2000 compared to graphs of 5 vertices. The combinatorial explosion is not prevented by the pruning strategy. Beyond 10 vertices, A* cannot converge to the optimality because of memory saturation phenomenon. The size of the list OPEN containing pending solutions grows exponentially according to the graph size and the bipartite heuristic fails to prune the search tree efficiently.

[1] https://brunl01.users.greyc.fr/CHEMISTRY/.
[2] https://iapr-tc15.greyc.fr/links.html.

Table 1. Results

	BLP-GED		A*		JH	
	Opt	Time	Opt	Time	Opt	Time
LETTER HIGH	100	43	100	3	100	13
LETTER MED	100	8	100	1	100	10
LETTER LOW	100	8	100	1	100	10
GREC 5	100	4	100	3	NA	NA
GREC 10	100	60	90.1	7980	NA	NA
GREC 20	99.5	11016	OM	OM	NA	NA
PROT 20	83.1	35376	OM	OM	NA	NA
PROT 30	37.9	81689	OM	OM	NA	NA
ILPISO 10	100	23	OM	OM	NA	NA
ILPISO 25	91.6	6545	OM	OM	NA	NA
ILPISO 50	56.9	17045	OM	OM	NA	NA
PAH	51	92991	OM	OM	100	948

Second, when analysing BLP-GED results, one can see that for datasets composed of smaller graphs (LETTER, ILPISO10, GREC5-10), BLP-GED converges to the optimality. For larger graphs (GREC20, PROT and ILPISO25-50), the optimal solution is not always reached, because of the time restriction but BLP-GED is the only method to solve instances and to output solutions. An interesting comment comes from the GREC dataset where nearly all instances are solved to optimality however increasing the graph size of only 5 vertices can lead to an important increase of time. The relation between graph size and solving time is not linear at all and it recalls us humbly how hard is the GED problem.

Finally, concerning JH, the method performs well for the PAH dataset solving 100 % of instances optimally against 51 % for BLP-GED. JH achieved better results because it is not generic but dedicated to compare unweighted graphs. However, on a smaller graphs like in LETTER, JH is the slowest method due to a higher number of variables and constraints than BLP-GED formulation. Finally, JH is not flexible and the method is unable to consider edge labels and cannot be applied to GREC, PROT and ILPISO.

5 Conclusion

In this paper, an exact binary linear programming formulation of the GED problem has been presented. One of the major advantage of our proposal against another binary program (JH) is that it can deal with a wide range of attributed relational graphs: directed or undirected graphs, simple graphs or multigraphs, with a combination of symbolic, numeric and/or string attributes on vertices and edges. Moreover, obtained results show that our BLP is about 100 times faster than $A*$ for medium-size graphs while being more memory efficient. Our future works concern the optimization and the approximation of our formulation of GED computation.

References

1. Fankhauser, S., Riesen, K., Bunke, H., Dickinson, P.J.: Suboptimal graph isomorphism using bipartite matching. IJPRAI **26**(6), 1250013 (2012)
2. Fischer, A., Bunke, H.: Character prototype selection for handwriting recognition in historical documents. In: 2011 19th European Signal Processing Conference, pp. 1435–1439, August 2011
3. Fischer, A., Plamondon, R., Savaria, Y., Riesen, K., Bunke, H.: A hausdorff heuristic for efficient computation of graph edit distance. In: Fränti, P., Brown, G., Loog, M., Escolano, F., Pelillo, M. (eds.) S+SSPR 2014. LNCS, vol. 8621, pp. 83–92. Springer, Heidelberg (2014). doi:10.1007/978-3-662-44415-3_9
4. Fischer, A., Suen, C.Y., Frinken, V., Riesen, K., Bunke, H.: Approximation of graph edit distance based on Hausdorff matching. Pattern Recogn. **48**(2), 331–343 (2015)
5. Gaüzère, B., Brun, L., Villemin, D.: Two new graphs kernels in chemoinformatics. Pattern Recogn. Lett. **33**(15), 2038–2047 (2012). http://www.sciencedirect.com/science/article/pii/S016786551200102X, Graph-Based Representations in Pattern Recognition
6. Hammami, M., Héroux, P., Adam, S., d'Andecy, V.P.: One-shot field spotting on colored forms using subgraph isomorphism. In: 2015 13th International Conference on Document Analysis and Recognition (ICDAR), pp. 586–590, August 2015
7. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968)
8. Héroux, P., Bodic, P., Adam, S.: Datasets for the evaluation of substitution-tolerant subgraph isomorphism. In: Lamiroy, B., Ogier, J.-M. (eds.) GREC 2013. LNCS, vol. 8746, pp. 240–251. Springer, Heidelberg (2014). doi:10.1007/978-3-662-44854-0_19
9. Justice, D., Hero, A.: A binary linear programming formulation of the graph edit distance. IEEE Trans. Pattern Anal. Mach. Intell. **28**(8), 1200–1214 (2006)
10. Kostakis, O.: Classy: fast clustering streams of call-graphs. Data Min. Knowl. Discov. **28**(5), 1554–1585 (2014). doi:10.1007/s10618-014-0367-9
11. Le Bodic, P., Héroux, P., Adam, S., Lecourtier, Y.: An integer linear program for substitution-tolerant subgraph isomorphism and its use for symbol spotting in technical drawings. Pattern Recogn. **45**(12), 4214–4224 (2012)
12. Lerouge, J., Hammami, M., Héroux, P., Adam, S.: Minimum cost subgraph matching using a binary linear program. Pattern Recogn. Lett. **71**, 45–51 (2016)
13. Lerouge, J., Bodic, P., Héroux, P., Adam, S.: GEM++: a tool for solving substitution-tolerant subgraph isomorphism. In: Liu, C.-L., Luo, B., Kropatsch, W.G., Cheng, J. (eds.) GbRPR 2015. LNCS, vol. 9069, pp. 128–137. Springer, Heidelberg (2015). doi:10.1007/978-3-319-18224-7_13
14. Myers, R., Wilson, R.C., Hancock, E.R.: Bayesian graph edit distance. IEEE Trans. Pattern Anal. Mach. Intell. **22**(6), 628–635 (2000)
15. Neuhaus, M., Riesen, K., Bunke, H.: Fast suboptimal algorithms for the computation of graph edit distance. In: Yeung, D.-Y., Kwok, J.T., Fred, A., Roli, F., Ridder, D. (eds.) SSPR /SPR 2006. LNCS, vol. 4109, pp. 163–172. Springer, Heidelberg (2006). doi:10.1007/11815921_17

16. Raveaux, R., Burie, J.C., Ogier, J.M.: A graph matching method and a graph matching distance based on subgraph assignments. Pattern Recogn. Lett. **31**(5), 394–406 (2010)
17. Riesen, K., Bunke, H.: IAM graph database repository for graph based pattern recognition and machine learning. In: da Vitoria Lobo, N., Kasparis, T., Roli, F., Kwok, J.T., Georgiopoulos, M., Anagnostopoulos, G.C., Loog, M. (eds.) Structural, Syntactic, and Statistical Pattern Recognition. LNCS, vol. 5342, pp. 287–297. Springer, Heidelberg (2008)
18. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image Vis. Comput. **27**(7), 950–959 (2009)
19. Riesen, K., Bunke, H.: Graph Classification and Clustering Based on Vector Space Embedding. World Scientific Publishing Co. Inc., River Edge (2010)
20. Riesen, K., Bunke, H.: Improving bipartite graph edit distance approximation using various search strategies. Pattern Recogn. **48**(4), 1349–1363 (2015)
21. Riesen, K., Fankhauser, S., Bunke, H.: Speeding up graph edit distance computation with a bipartite heuristic. In: Frasconi, P., Kersting, K., Tsuda, K. (eds.) Mining and Learning with Graphs, MLG 2007, Firence, Italy, 1–3 August 2007, Proceedings (2007)
22. Zeng, Z., Tung, A.K.H., Wang, J., Feng, J., Zhou, L.: Comparing stars: on approximating graph edit distance. Proc. VLDB Endowment. **2**, 25–36 (2009)