Graph Edit Distance in the exact context

Mostafa Darwiche, Romain Raveaux, Donatello Conte, Vincent t'Kindt

Graph Edit Distance in the exact context

Mostafa Darwiche[1,2], Romain Raveaux[1], Donatello Conte[1], and Vincent T'Kindt[2]

[1] Université de Tours, LIFAT EA6300, 64 avenue Jean Portalis, 37200 Tours, France
[2] Université de Tours, LIFAT EA6300, ROOT ERL CNRS 7002, 64 avenue Jean Portalis, 37200 Tours, France
{mostafa.darwiche,romain.raveaux,donatello.conte,tkindt}@univ-tours.fr

Abstract. This paper presents a new Mixed Integer Linear Program (MILP) formulation for the Graph Edit Distance (GED) problem. The contribution is an exact method that solves the GED problem for attributed graphs. It has an advantage over the best existing one when dealing with the case of dense of graphs, because all its constraints are independent from the number of edges in the graphs. The experiments have shown the efficiency of the new formulation in the exact context.

Keywords: Graph Edit Distance · Graph Matching · Mixed Integer Linear Program.

1 Introduction

Graphs are very powerful in modeling structural relations of objects and patterns. A graph consists of two sets of vertices and edges. The vertices represent the main components, while the edges show the link between those components. In a graph, it is also possible to store information and features about the object, by assigning attributes to vertices and edges. Graphs have been used in many applications and fields, such as *Pattern Recognition* to model objects in images and videos [14]. Also, graphs form a natural representation of the atom-bond structure of molecules, therefore they have applications in *Cheminformatics* field [12]. A common task is then, the ability to compare graphs or find (dis)similarities between them. Such a task enables comparing objects and patterns that are represented by graphs, and this is known as *Graph Matching* (GM). GM has been split into different sub-problems, which mainly fall under two categories: *exact* and *error tolerant*. The first one is very strict, while the second is more flexible and tolerant to differences in topologies and attributes, which makes it more suitable for real-life scenarios.

Graph Edit Distance (GED) problem is an error-tolerant graph matching problem. It provides a dissimilarity measure between two graphs, by computing the cost of editing one graph to transform it into another. The set of edit operations are substitution, insertion and deletion, and can be applied on both vertices and edges. There is a cost associated to each edit operation. Solving the GED problem consists in finding the sequence of edit operations that minimizes the

total cost. GED, by concept, is known to be flexible because it has been shown that changing the edit cost properties can result in solving other matching problems such as, maximum common subgraph, graph and subgraph isomorphism [4]. GED is a minimization problem that was proven to be NP-hard. The problem is complex and hence it was mostly treated by heuristic methods in order to compute sub-optimal solutions in reasonable time. A famous heuristic is called *Bipartite Graph Matching* (BP), which is known to be fast [13]. BP breaks down the GED problem into a linear sum assignment problem that can be solved in polynomial time, using the *Hungarian* algorithm [11]. BP was integrated later in other heuristics such as *Fast BP*, *Square BP* and *Beam-search BP* [15, 6]. Two new heuristics: *Integer Projected Fixed Point* (IPFP) and *Graduate Non Convexity and Concavity Procedure* (GNCCP), were proposed by Bougleux et al. [3]. Both are adapted to operate over a *Quadratic Assignment Problem* (QAP) that models the GED. These heuristics aim at approximating the quadratic objective function to compute a solution and then improve it by applying projection methods. In a recent work by Darwiche et al. [5], a heuristic called *Local Branching GED* was proposed, that is based on local searches in the solution space of a *Mixed Integer Linear Program* (MILP). On the other hand, and in the exact context (e.g. methods that compute optimal solutions), there are three MILP formulations in the literature. Only two of them are designed to solve the general GED problem [8]. The third formulation was designed by Justice and Hero [7], and it is the most efficient formulation. However, it only deals with a special case of the GED problem, where attributes on edges are ignored and a constant cost is assigned to edges edit operations. As well, in the exact context, there is a branch and bound algorithm [2], which was shown later to be less efficient than MILP formulations.

The present work is with the interest of designing a new MILP formulation to solve the GED problem, and so contributes to the exact methods for GED. A new efficient formulation is proposed that has good performance w.r.t. existing formulations in the literature. The new formulation is inspired by $F2$, which is proposed by Lerouge et al. [8]. It is an improvement to $F2$ by modifying the variables and the constraints. It has the advantage over $F2$, that the constraints are independent from the number of edges in the graphs. The remainder is organized as follows: Section 2 presents the definition of the GED problem, followed with a review of $F2$ formulation. Then, Section 3 details the improved formulation. Section 4 shows the results of the computational experiments. Finally, Section 5 highlights some concluding remarks.

2 *GED* definition and *F2* formulation

2.1 *GED* problem definition

An attributed graph is a 4-tuple $G = (V, E, \mu, \xi)$ where, V is the set of vertices, E is the set of edges, such that $E \subseteq V \times V$, $\mu : V \to L_V$ (resp. $\xi : E \to L_E$) is the function that assigns attributes to a vertex (resp. an edge), and L_V (resp. L_E) is the label space for vertices (resp. edges).

Next, given two graphs $G = (V, E, \mu, \xi)$ and $G' = (V', E', \mu', \xi')$, GED is the task of transforming one graph source into another graph target. To accomplish this, GED introduces the vertices and edges edit operations: $(i \rightarrow k)$ is the substitution of two vertices, $(i \rightarrow \epsilon)$ is the deletion of a vertex, and $(\epsilon \rightarrow k)$ is the insertion of a vertex, with $i \in V, k \in V'$ and ϵ refers to the empty node. The same logic goes for edges. The set of operations that reflects a valid transformation of G into G' is called a complete edit path, defined as $\lambda(G, G') = \{o_1, ..., o_k\}$, where o_i is an elementary vertex (or edge) edit operation and k is the number of operations. GED is then

$$d_{min}(G, G') = \min_{\lambda \in \Gamma(G,G')} \sum_{o_i \in \lambda} \ell(o_i) \tag{1}$$

where $\Gamma(G, G')$ is the set of all complete edit paths, d_{min} represents the minimal cost obtained by a complete edit path $\lambda(G, G')$, and $\ell(.)$ is the cost function that assigns costs to elementary edit operations.

2.2 Mixed Integer Linear Program

The general MILP formulation is of the form:

$$\min_x \ c^T x \tag{2}$$

$$Ax \geq b \tag{3}$$

$$x_i \in \{0, 1\}, \forall i \in B \tag{4}$$

$$x_j \in \mathbb{N}, \forall j \in I \tag{5}$$

$$x_k \in \mathbb{R}, \forall k \in C \tag{6}$$

where $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$ are vectors of coefficients, $A \in \mathbb{R}^{m \times n}$ is a matrix of coefficients. x is a vector of variables to be computed. The variable index set is split into three sets (B, I, C), respectively stands for binary, integer and continuous. This formulation minimizes an objective function (Eq. 2) w.r.t. a set of linear inequality constraints (Eq. 3) and the bounds imposed on variables x e.g. integer or binary. A feasible solution to this formulation is a vector x with the proper values based on their defined types, that satisfies all the constraints. The optimal solution is a feasible solution that has the minimum objective function value. This approach of modeling decision problems (i.e. problems with binary and integer variables) is very efficient, especially for hard optimization problems.

2.3 *F*2 formulation

$F2$ is the best MILP formulation for the GED problem in the literature, it was proposed by Lerouge et al. [8]. It is based on a previous and straightforward MILP formulation, referred to as $F1$, by the same authors. $F2$ formulation is a more compact and improved version of $F1$ by reducing the number of variables and constraints. The compactness of $F2$ comes from the design of the objective function to be optimized. At first, it considers all vertices and edges of G as

deleted and vertices and edges of G' as inserted. Then, it solves the problem of finding the cheapest assignments/matching between the two sets of vertices and the two sets of edges. The matching in this context is the substitution edit operations for vertices and edges. Once, the cheapest matching is computed, the deletion and insertion operations can be concluded. All the remaining vertices in V (resp. in V') that are not matched with any vertex in V' (resp. in V), are considered as deleted (resp. inserted). The edges are treated in the same manner. Such design is helpful in reducing the number of variables and constraints in the formulation. In the following, $F2$ is detailed by defining the data of the problem, variables, objective function to minimize and constraints to respect.

Data. Given two graphs $G = (V, E, \mu, \xi)$ and $G' = (V', E', \mu', \xi')$, the cost functions, in order to compute the cost of each vertex/edge edit operations, are known and defined. Therefore, vertices cost matrix $[c_v]$ is computed as in equation 7 for every couple $(i, k) \in V \times V'$. The ϵ column is added to store the cost of deletion i vertices, while the ϵ row stores the costs of insertion k vertices. Following the same process, the matrix $[c_e]$ is computed for every $((i, j), (k, l)) \in E \times E'$, plus the row/column ϵ for deletion and insertion of edges.

$$c_v = \begin{array}{c} \begin{matrix} v_1 & v_2 & \dots & v_{|V'|} & \epsilon \end{matrix} \\ \begin{bmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,|V'|} & c_{1,\epsilon} \\ c_{2,1} & c_{2,2} & \dots & c_{2,|V'|} & c_{2,\epsilon} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ c_{|V|,1} & c_{|V|,2} & \dots & c_{|V|,|V'|} & c_{|V|,\epsilon} \\ c_{\epsilon,1} & c_{\epsilon,2} & \dots & c_{\epsilon,|V|} & 0 \end{bmatrix} \end{array} \begin{matrix} u_1 \\ u_2 \\ \vdots \\ u_{|V|} \\ \epsilon \end{matrix} \tag{7}$$

Variables. As mentioned earlier, $F2$ formulation focuses on finding the correspondences between the two sets of vertices and the two sets of edges. That is why two sets of decision variables are needed.

- $x_{i,k} \in \{0, 1\}\ \forall i \in V, \forall k \in V'$; $x_{i,k} = 1$ when vertices i and k are matched, and 0 otherwise.
- $y_{ij,kl} \in \{0, 1\}\ \forall (i, j) \in E, \forall (k, l) \in E'$; $y_{ij,kl} = 1$ when edge (i, j) is matched with (k, l), and 0 otherwise.

Objective function. The objective function to minimize is the following.

$$\min_{x,y} \sum_{i \in V} \sum_{k \in V'} \left(c_v(i,k) - c_v(i,\epsilon) - c_v(\epsilon,k)\right).x_{i,k} + \tag{8}$$
$$\sum_{(i,j) \in E} \sum_{(k,l) \in E'} \left(c_e(ij,kl) - c_e(ij,\epsilon) - c_e(\epsilon,kl)\right).y_{ij,kl} + \gamma$$

The objective function minimizes the cost of assigning vertices and edges with the cost of substitution subtracting the cost of insertion and deletion. The γ,

which is a constant and given in equation 9, compensates the subtracted costs of the assigned vertices and edges. This constant does not impact the optimization algorithm and it could be removed. It is there to obtain the GED value.

$$\gamma = \sum_{i \in V} c_v(i, \epsilon) + \sum_{k \in V'} c_v(\epsilon, k) + \sum_{(i,j) \in E} c_e(ij, \epsilon) + \sum_{(k,l) \in E'} c_e(\epsilon, kl) \tag{9}$$

Constraints. $F2$ has 3 sets of constraints.

$$\sum_{k \in V'} x_{i,k} \leq 1 \ \forall i \in V \tag{10}$$

$$\sum_{i \in V} x_{i,k} \leq 1 \ \forall k \in V' \tag{11}$$

$$\sum_{(k,l) \in E'} y_{ij,kl} \leq x_{i,k} + x_{j,k} \ \forall k \in V', \forall (i,j) \in E \tag{12}$$

Constraints 10 and 11 are to make sure that a vertex can be only matched with maximum one vertex. It is possible that a vertex is not assigned to any other, in this case it is considered as deleted or inserted. Here is the key point of this formulation: $F2$ is flexible by allowing some vertices/edges not to be matched. The objective function gets to decide whether a substitution is cheaper than a deletion/insertion or not. γ takes care of the unmatched vertices/edges and includes their deletion or insertion costs to the objective function. Finally, constraints 12 guarantee preserving edges matching between two couple of vertices. In other words, to match two edges $(i,j) \rightarrow (k,l)$, their vertices must be matched first, i.e. $i \rightarrow k$ and $j \rightarrow l$ OR $i \rightarrow l$ and $j \rightarrow k$.

The presented version of $F2$ formulation, and for the sake of simplicity, is applied to undirected graphs. For the directed case, it simply splits the constraints 12 into two sets of constraints. For more details, please refer to the paper [9].

3 Improved MILP formulation ($F3$)

3.1 $F3$ formulation

$F3$ is a new and an improved MILP formulation, inspired by $F2$, to solve the GED problem. It shares some parts of $F2$ and it is defined as follows.

Data. Same as in $F2$ formulation, $F3$ uses the cost matrices $[c_v]$ and $[c_e]$.

Variables. $F3$ introduces two sets of decision variables $x_{i,k}$ and $y_{ij,kl}$ as in $F2$. However, it includes more y variables, by creating two variables: $y_{ij,kl}$ and $y_{ij,lk}$ for every $((i,j),(k,l)) \in E \times E'$. Let $\overline{E}' = \{(l,k) : \forall (k,l) \in E'\}$. The variables of the formulation are as follows.

- $x_{i,k} \in \{0,1\} \ \forall i \in V, \forall k \in V'$; $x_{i,k} = 1$ when vertices i and k are matched, and 0 otherwise.
- $y_{ij,kl} \in \{0,1\} \ \forall (i,j) \in E, \forall (k,l) \in E' \cup \overline{E}'$; $y_{ij,kl} = 1$ when edge (i,j) is matched with (k,l), and 0 otherwise.

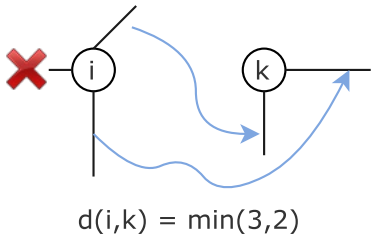

Fig. 1. Example of edges assignment when assigning two vertices

Objective function. It is basically the same function as in $F2$ formulation, except for the cost sum over the y variables to include all of them.

$$\min_{x,y} \sum_{i \in V} \sum_{k \in V'} \left(c_v(i,k) - c_v(i,\epsilon) - c_v(\epsilon,k)\right) .x_{i,k} + \tag{8-a}$$
$$\sum_{(i,j) \in E} \sum_{(k,l) \in E' \cup \overline{E}'} \left(c_e(ij,kl) - c_e(ij,\epsilon) - c_e(\epsilon,kl)\right) .y_{ij,kl} + \gamma$$

Constraints. $F3$ formulation shares the same sets of constraints 10 and 11, that assure a vertex is only matched with one vertex at most. However, it re-writes the constraints 12 in a different fashion.

$$\sum_{(i,j) \in E} \sum_{(k,l) \in E' \cup \overline{E}'} y_{ij,kl} \leq d_{i,k} \times x_{i,k} \ \forall i \in V, \forall k \in V' \tag{12-a}$$

With $d_{i,k} = min(degree(i), degree(k))$. The degree of a vertex is the number of edges incident to the vertex. The constraints stands for: whenever two vertices are matched, e.g. $(i \rightarrow k)$, the maximum number of edges substitution that can be done is equal to the minimum degree of the two vertices. Figure 1 shows an example of the case. Two edges at most can be substituted and the third of i has to be deleted. Of course, the deletion of all edges is possible, if it costs less than the substitutions. These constraints force matching the edges and respecting the topological constraint defined in the GED problem.

The given formulation handles the case of undirected graphs. Though, it can be adapted to deal with the directed case, by setting $\overline{E}' = \{\phi\}$ (because edges (i,j) are different from (j,i) and they are already included in E), and replacing the objective function Eq. 8-a by the objective function of $F2$ Eq. 8.

3.2 *F2* vs. *F3*

The most important improvement in the proposed formulation is that $F3$ has sets of constraints independent of the number of edges in the graphs. Constraints 10 and 11 are shared by both formulations and they do not include edges. However, constraints 12 rely on the edges of G, which is not the case of the constraints

12-a in $F3$. Table 1 shows the number of variables and constraints in both formulations. Clearly, $F3$ has (2 times) more y variables than $F2$. The reason behind creating two y variables for each couple of edges, is to accommodate to the symmetry case that appears when dealing with undirected graphs, i.e. $(i, j) = (j, i)$. By doing so, the constraints 12 can be re-written differently by relying only on the vertices of the graphs (constraints 12-a). Note that, this comparison is done for undirected graphs. In the other case, the symmetry is discarded, and both formulations have the same number of variables.

Table 1. Nb. of variables and constraints in $F2$ and $F3$

	Nb. of variables	Nb. of Constraints
$F2$	$\|V\| \times \|V'\| + \|E\| \times \|E'\|$	$\|V\| + \|V'\| + \|V\| \times \|E\|$
$F3$	$\|V\| \times \|V'\| + \|E\| \times \|E'\| \times 2$	$\|V\| + \|V'\| + \|V\| \times \|V'\|$

In the GED problem, edge operations are driven by vertex-vertex matching. On this basis, the difficulty in $F2$ and $F3$ comes from the x decision variables, rather than the y variables. Moreover, $F2$ formulation is more sensitive to the density of the graphs (% connectivity, $D = \frac{2|E|}{|V|(|V|-1)}$), because its constraints depend on the edges, which is not the case in $F3$. This reasoning led to make the following two assumptions, by distinguishing between two cases:

1. Non-dense graphs: even if $F3$ has more y variables than in $F2$, its performance will not be degraded compared to $F2$.
2. Dense graphs: $F3$ will have less constraints than $F2$, since $F3$ has a number of constraints independent from the number of edges. Consequently, $F3$ tends to perform better than $F2$.

To validate those assumptions, both formulations are tested over two graph databases. The results are discussed in the next section.

4 Computational Experiment

4.1 Databases

Two databases are selected from the literature in order to evaluate $F3$.

MUTA. This database consists of graph that model chemical molecules [1]. It is commonly used when testing GED methods, mainly because it contains different subsets of small and large graphs. It allows exploiting GED methods and shows their behaviors when the instances get more difficult. There are 7 subsets, each of which has 10 graphs of same size (10 to 70 vertices) and a subset of also 10 graphs with mixed graph sizes. Each pair of graphs is considered as an instance. Therefore, a total of 800 instances (100 per subset) are considered in this experiment. The density of the graphs is very low ($D = 7\%$), hence they are considered as non-dense graphs. The choice of the edit operations costs is based on the values defined in [1].

CMUHOUSE. This database contains 111 graphs corresponding to 3-D images of houses [10], each graph consists of 30 vertices with attributes described using Shape Context feature vector. The graphs are extracted from 3-D house images, where the houses are rotated with different angles. This is interesting because it enables testing and comparing graphs that represent the same house but positioned differently inside the images. For this database, there are 660 instances in total. The density of these graphs is higher than MUTA graphs, $D = 18\%$. Two versions of this database are considered: CMUHOUSE-NA is the version where attributes are not considered when calculating the costs; CMUHOUSE-A a second version with costs computed based on the functions given in [16].

4.2 Experiment settings

Both formulations are implemented in C language, and solved by CPLEX 12.7.1 with time limit 900 seconds. The tests were executed on a machine with the following configuration: Windows 7 (64-bit), Intel Xeon E5 4 cores and 8 GB RAM. For each formulation, the following values are computed for each subset of graphs: t_{avg} is the average CPU time in seconds for all instances, d_{avg} is the deviation percentage between the solutions obtained by one formulation, and the best computed by both formulations. For example, given an instance I, the deviation percentage for $F3$ is equal to $\frac{sol_I^{F3} - best_I}{best_I} \times 100$, with $best_I = min(sol_I^{F2}, sol_I^{F3})$. Lastly, η_I and η'_I represent, respectively, the number of optimal solutions obtained by a formulation, and the number of solutions for which, a given formulation has provided the minimum (smaller objective function value, without necessarily a proof of optimality).

4.3 Results and analysis

MUTA results. Table 2 shows the results obtained for both formulations for each subset of graphs. Looking at d_{avg} for $F2$, it scores the smallest values for all the subsets, except for subset 70. However, the gap between both formulations is small, especially with small instances (0% for subsets 10 and 20). In terms optimal solutions (η), $F3$ has higher numbers for subsets $30, 40, 50$ and $Mixed$, with greater differences: for subsets 30 at 76 optimal solutions against 48, and subset 50 at 31 optimal solutions against 19. Regarding η', $F2$ has higher numbers for most of the subsets ($30, 50, 60$ and $Mixed$). However, η' of $F3$ are not far the ones of $F2$. At last, $F2$ is faster than $F3$ for small and medium subsets ($10, 20, 30$ and $Mixed$). But, for the rest of the subsets, both formulations suffer from high computation time and reach the time limit set ($900s$). The conclusion of this experiment: both formulations seems to be very close in terms of performance and efficiency in computing optimal solutions. It is hard to tell which formulation is better. This result corroborates the first assumption, that is $F3$ is as good as $F2$ in the case of non-dense graphs.

Table 2. Results of MUTA instances

		10	20	30	40	50	60	70	Mixed
F3	$t_{avg}(s)$	0.10	3.07	365.44	575.65	770.61	810.51	811.10	410.08
	d_{avg}	**0.00**	**0.00**	0.74	0.54	1.78	3.60	**2.55**	0.80
	η	**100**	**100**	**81**	**76**	**31**	10	10	**62**
	η'	**100**	**100**	91	**90**	68	53	**61**	78
F2	$t_{avg}(s)$	**0.05**	**0.99**	**320.35**	**571.65**	**766.63**	**802.94**	**802.69**	**370.36**
	d_{avg}	**0.00**	**0.00**	**0.21**	**0.51**	**1.52**	**1.46**	2.76	**0.15**
	η	**100**	**100**	79	48	19	**11**	**11**	61
	η'	**100**	**100**	**93**	84	**69**	**69**	60	**91**

Table 3. Results of CMUHOUSE instances

		CMUHOUSE-NA	CMUHOUSE-A
F3	$t_{avg}(s)$	**497.07**	416.75
	d_{avg}	**0.70**	**0.22**
	η	**365**	**633**
	η'	**644**	**652**
F2	$t_{avg}(s)$	880.74	**278.78**
	d_{avg}	604.11	4.68
	η	25	505
	η'	54	548

CMUHOUSE results. Table 3 presents the results of both formulations for both versions of CMUHOUSE. In the case of CMUHOUSE-NA (no attributes), the instances seem to be harder than the version with attributes. When ignoring the attributes, the similarities between vertices and edges are high and it does not allow to easily differentiate between them. The average deviation for $F3$ is 0.70% against 604.11% for $F2$, the difference is remarkably high. This is also seen when looking at η and η', respectively, $365, 644$ for $F3$ against $25, 54$ for $F2$. $F3$ was able to compute optimal solutions for more than 50% of the instances. It looks like $F2$ had hard time with these instances in converging towards good solutions. The version with attributes (CMUHOUSE-A) is easier, but still $F3$ has scored $d_{avg} = 0.22\%$ against 4.68% for $F2$. $F3$ has solved more instances to optimality (652) than $F2$ (505). Based on these results, the second assumption also holds true. CMUHOUSE graphs are more dense than MUTA, which means that $F3$ has less constraints, since all its constraints are independent from the number of edges in the graphs. As a result, $F3$ has performed better than $F2$.

5 Conclusion

In this work, a new MILP formulation is proposed for the GED problem. The new formulation is an improvement to the best existing one. The results of the experiments have shown the efficiency of this formulation, especially in the case of dense graphs. This is due to the fact that, the constraints are independent from the edges in the graphs. The next step will be to evaluate the new formulation

against more graph databases with different settings, i.e. graphs with high and very high densities.

References

1. Abu-Aisheh, Z., Raveaux, R., Ramel, J.: A graph database repository and performance evaluation metrics for graph edit distance. In: Graph-Based Representations in Pattern Recognition - 10th IAPR-TC-15.Proceedings. pp. 138–147 (2015)
2. Abu-Aisheh, Z., Raveaux, R., Ramel, J.Y., Martineau, P.: An exact graph edit distance algorithm for solving pattern recognition problems. In: 4th International Conference on Pattern Recognition Applications and Methods 2015 (2015)
3. Bougleux, S., Brun, L., Carletti, V., Foggia, P., Gaüzère, B., Vento, M.: Graph edit distance as a quadratic assignment problem. Pattern Recognition Letters **87**, 38–46 (2017)
4. Bunke, H.: On a relation between graph edit distance and maximum common subgraph. Pattern Recognition Letters **18**(8), 689–694 (1997)
5. Darwiche, M., Conte, D., Raveaux, R., TKindt, V.: A local branching heuristic for solving a graph edit distance problem. Computers & Operations Research (2018)
6. Ferrer, M., Serratosa, F., Riesen, K.: Improving bipartite graph matching by assessing the assignment confidence. Pattern Recognition Letters **65**, 29–36 (2015)
7. Justice, D., Hero, A.: A binary linear programming formulation of the graph edit distance. IEEE Transactions on Pattern Analysis and Machine Intelligence **28**(8), 1200–1214 (2006)
8. Lerouge, J., Abu-Aisheh, Z., Raveaux, R., Héroux, P., Adam, S.: New binary linear programming formulation to compute the graph edit distance. Pattern Recognition **72**, 254–265 (2017)
9. Lerouge, J., Abu-Aisheh, Z., Raveaux, R., Héroux, P., Adam, S.: New binary linear programming formulation to compute the graph edit distance. Pattern Recognition **72**, 254–265 (2017). https://doi.org/10.1016/j.patcog.2017.07.029, https://doi.org/10.1016/j.patcog.2017.07.029
10. Moreno-García, C.F., Cortés, X., Serratosa, F.: A graph repository for learning error-tolerant graph matching. In: Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR). pp. 519–529. Springer (2016)
11. Munkres, J.: Algorithms for the assignment and transportation problems. Journal of the society for industrial and applied mathematics **5**(1), 32–38 (1957)
12. Raymond, J.W., Willett, P.: Maximum common subgraph isomorphism algorithms for the matching of chemical structures. Journal of computer-aided molecular design **16**(7), 521–533 (2002)
13. Riesen, K., Neuhaus, M., Bunke, H.: Bipartite graph matching for computing the edit distance of graphs. In: International Workshop on Graph-Based Representations in Pattern Recognition. pp. 1–12. Springer (2007)
14. Sanfeliu, A., Fu, K.S.: A distance measure between attributed relational graphs for pattern recognition. IEEE Transactions on Systems, Man, and Cybernetics **SMC-13**(3), 353–362 (May 1983). https://doi.org/10.1109/TSMC.1983.6313167
15. Serratosa, F.: Computation of graph edit distance: reasoning about optimality and speed-up. Image and Vision Computing **40**, 38–48 (2015)
16. Zhang, Z., Shi, Q., McAuley, J.J., Wei, W., Zhang, Y., Van Den Hengel, A.: Pairwise matching through max-weight bipartite belief propagation. In: CVPR. vol. 5, p. 7 (2016)