Solving a special case of the Graph Edit Distance Problem with Local Branching

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

▶ **To cite this version:**

Mostafa Darwiche, Romain Raveaux, Donatello Conte, Vincent t'Kindt. Solving a special case of the Graph Edit Distance Problem with Local Branching. Matheuristics 2018, Jun 2018, Tours, France. hal-01717709

HAL Id: hal-01717709
https://hal.science/hal-01717709

Submitted on 24 Sep 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Solving a special case of the Graph Edit Distance Problem with Local Branching

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

[1] Laboratoire d'Informatique (LI), Université François Rabelais Tours, France
[2] Laboratoire d'Informatique (LI), ERL-CNRS 6305, Université François Rabelais Tours, France
{mostafa.darwiche,romain.raveaux,donatello.conte,tkindt}@univ-tours.fr

Abstract. The Graph Edit Distance (GED) problem is a well-known graph matching problem. Solving the GED problem implies minimizing a dissimilarity measure between graphs that normally represent objects and patterns. It is known to be very flexible and can work on any type of graphs. GED^{EnA} (Edges no Attributes) is a sub-problem of GED that deals with a special type of graphs where edges do not carry attributes. Both are modeled as minimization problems and proven to be NP-Hard. Many heuristics are defined in the literature to give approximated solutions in a reasonable CPU time. Some other work have used mathematical programming to come up with Mixed Integer Linear Program (MILP) models. The present work proposes the use of Local Branching heuristic over a powerful MILP model to solve the GED^{EnA} problem. Mainly, a MILP model is iteratively modified by adding additional constraints to define neighborhoods in the solution space which are explored using a black-box solver. A problem-dependent exploration is performed to find efficient solutions. Lastly, the proposed heuristic is evaluated w.r.t literature heuristics.

Keywords: Graph Edit Distance, Graph Matching, Local Branching Heuristic.

1 Introduction

Graph-based representations can efficiently model objects and patterns. The graphs provide a structural representation of an object, by defining the main components that form the object using vertices, and drawing the relations between them using edges. The structural information provided by the topology of the graph can be enriched by attributes (labels). Attributes are assigned to vertices and edges to carry information about subparts of the object. Attributed graphs have been actively used in many fields, such as Pattern Recognition to perform object recognition, image registration, tracking and many other applications [12]. Also, attributed graphs form a natural representation of the atom-bond structure of molecules, therefore they have also applications in Cheminformatics field [10].

The *Graph Edit Distance* (GED) problem is an error-tolerant graph matching problem. It provides a dissimilarity measure between two graphs, by minimizing the edit operation costs to transform one graph into another. The set of edit operations are substitution, insertion and deletion, and can be applied on both vertices and edges. Solving the GED problem consists in finding the set of edit operations that minimizes the total cost of matching one graph into another. The GED problem, 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 like maximum common subgraph, graph and subgraph isomorphism [3]. GED is a NP-hard problem [14], so its optimal solution cannot be obtained in polynomial time, unless $P = NP$. Nevertheless, many heuristics have been proposed to compute good solutions in reasonable amount of time. The works in [11, 13] presented fast algorithms that mainly solve the linear sum assignment problem between two sets of vertices, and then deduce the edges assignment. The vertices cost matrix includes information about the edges, through an estimation of the edges assignment cost implied by assigning two vertices. However, one drawback in this approach is that it takes into account only local structures in the graph. Other algorithms based on beam search are presented in [4, 9]. The first one explores a truncated search tree built from all vertices assignments. The second algorithm solves the vertices assignment problem and then, by using a beam search, it tries to improve the initial assignment by switching vertices. The GED problem has been also addressed by the means of mathematical programming. Two types of mathematical formulations can be found in the literature: linear models as in [7] and quadratic as in [2]. A sub-problem of GED is GED^{EnA}, where input graphs do not have attributes on their edges, implying a null cost for edge substitution operations. The same aforementioned heuristics and exact solution methods can be applied to the GED^{EnA} problem. In addition, a very efficient $MILP^{JH}$ model is proposed in [6], works only for the GED^{EnA} problem. This problem is also NP-hard [14] and has many applications, notably in Cheminformatics field.

In the present work, a Local Branching (LocBra) heuristic is proposed to solve the GED^{EnA} problem, with the intent to strongly outperforms the available literature heuristics. LocBra was originally introduced by Fischetti and Lodi [5], as a general metaheuristic based on MILP formulations. Typically, a local branching heuristic is a local search algorithm, which improves an initial solution by exploring a series of defined neighborhoods via the solution of restricted MILP formulations. As well, it involves techniques such as intensification and diversification during the exploration process. LocBra depends on a MILP model: $MILP^{JH}$ is chosen in the implementation of LocBra, because it is one of the most efficient models for the GED^{EnA} problem [7]. The remainder is organized as follows: Section 2 presents the definition of the GED^{EnA} problem, followed with a review of $MILP^{JH}$ model. Then, Section 3 details the proposed heuristic. And Section 4 shows the results of the computational experiments. Finally, Section 5 highlights some concluding remarks.

2 GED^{EnA} definition and $MILP^{JH}$ model

Before introducing the general Graph Edit Distance (GED) problem, the definition of attributed and directed graph is given first.

Definition 1. *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).*

Given two graphs $G = (V, E, \mu, \xi)$ and $G' = (V', E', \mu', \xi')$, solving the GED problem consists in transforming the source graph G into the graph target G'. To accomplish this, vertices and edges edit operations are introduced: $(u \to v)$ is the substitution of two nodes, $(u \to \epsilon)$ is the deletion of a node in G, and $(\epsilon \to v)$ is the insertion of a node in G, with $u \in V, v \in V'$ and ϵ refers to the empty node. The same logic goes for the edges. The set of operations that reflects a valid transformation of G into G' is called a complete edit path, defined as $\lambda(G, G') = \{o_1, ..., o_m\}$ where o_i is an elementary vertex (or edge) edit operation and m is the number of such operations.

Definition 2. *The Graph Edit Distance between two graphs G and G' is defined by:*

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

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 ℓ is the cost function that assigns the costs to elementary edit operations.

For the GED^{EnA} problem, the graphs are the same as in Definition 1, except that the attribute set for edges is empty (i.e. $L_E = \{\phi\}$). Consequently, the costs of edge edit operations are 0 for substitution and a constant for insertion and deletion (i.e. $\ell(e \to f) = 0$, $\ell(e \to \epsilon) = \ell(\epsilon \to f) = \tau$, $\forall e \in E; f \in E'$ and $\tau \in \mathbb{R}^+$).

$MILP^{JH}$ is a model proposed in [6] that solves the GED^{EnA} problem. The main idea consists in determining the permutation matrix minimizing the L_1 norm of the difference between the adjacency matrix of the input graph and the permuted adjacency matrix of the target one. The details about the construction of the model can be found in [6]. The model is as follows:

$$\min_{x,s,t \in \{0,1\}^{N \times N}} \left(f(x, s, t) = \sum_{i=1}^{N} \sum_{j=1}^{N} \ell\left(\mu(u_i), \mu'(v_j)\right) x_{ij} + \left(\frac{1}{2} \cdot \tau \cdot (s_{ij} + t_{ij}) \right) \right) \tag{2}$$

such that

$$\sum_{k=1}^{N} A_{ik} x_{kj} - \sum_{c=1}^{N} x_{ic} A'_{cj} + s_{ij} - t_{ij} = 0 \ \forall i, j \in \{1, 2, ..., N\} \tag{3}$$

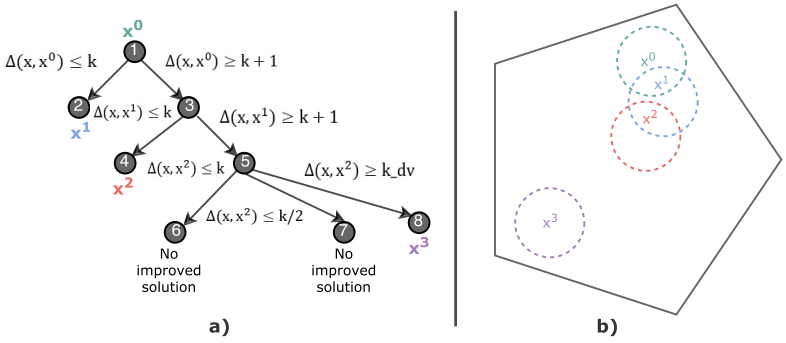

Fig. 1. Local branching flow. a) depicts the left and right branching. b) shows the neighborhoods in the solution space

$$\sum_{i=1}^{N} x_{ik} = \sum_{j=1}^{N} x_{kj} = 1 \ \forall k \in \{1, 2, ..., N\} \tag{4}$$

where A and A' are the adjacency matrices of graphs G and G' respectively, $\ell : (\mu(u_i), \mu'(v_j)) \rightarrow \mathbb{R}^+$ is the cost function that measures the distance between two vertices attributes. Variables x, s and t, are permutation matrices of size $N \times N$, and of boolean type, with $N = |V| + |V'|$. Matrix x represents the vertices matching, i.e. $x_{ij} = 1$ means a vertex $i \in V \cup \{\epsilon\}$ is matched with vertex $j \in V' \cup \{\epsilon\}$. While s and t are for edges matching. Hence, the objective function (Eq. 2) minimizes both, the cost of vertices and edges matching. As for constraint 3, it is to make sure that when matching two couples of vertices, the edges between each couple have to be mapped. Constraint 4 guarantees the integrity of x i.e. every row and column contains precisely a single 1. Therefore, one vertex in G can be only matched with one vertex in G'.

3 A Local Branching Heuristic for the GED^{EnA} problem

In this section, the main features of the local branching heuristic are explained as presented originally in [5], with improvements suggested to the GED^{EnA} problem outlined when appropriate.

Local branching is a metaheuristic that involves the solution of a MILP model to perform a local search and explore the neighborhoods of promising solutions by means of a branching scheme. In addition, it involves mechanisms such as intensification and diversification. Starting from an initial solution x^0, it defines the *k-opt neighborhood* $N(x^0, k)$, with k a given integer: the neighborhood set $N(x^0, k)$ contains the solutions that are within a distance no more than k from x^0 (in the sense of the *Hamming distance*). This implies adding the following *local branching constraint* to the base $MILP^{JH}$ model:

$$\Delta(x, x^0) = \sum_{(i,j) \in S^0} (1 - x_{ij}) + \sum_{(i,j) \notin S^0} x_{ij} \leq k \tag{5}$$

with, $S^0 = \{(i,j) : x^0_{ij} = 1\}$. This new model is then solved leading to the search of the best solution in $N(x^0, k)$. This phase corresponds to intensifying the search in a neighborhood, e.g. node 2 in Fig. 1-a. If a new solution x^1 is found, the constraint (5) is replaced by $\Delta(x, x^0) \geq k+1$ (node 3 in Fig. 1-a). This constraint makes sure that visited solutions (e.g. x^0) will not be selected twice. Next, a new constraint 5 is added but now with x^1 to explore its neighborhood. The process is repeated until a stopping criterion is met, e.g. a *total time limit* is reached. Moreover, Solving the restricted MILP (with constraint 5) might not return a better solution because the search neighborhood is still big and the node time limit imposed, is reached. For instance, assuming that at node 6 (Fig. 1-a) the solution of model $MILP^{JH}$ plus equation $\Delta(x, x^2) \leq k$ does not lead to a feasible and improved solution in the given time limit. Then, it might be interesting to apply a complementary intensification phase, by adding constraint $\Delta(x, x^2) \leq k/2$ and by solving the new model. If again, no feasible solution is found (e.g. node 7 of Fig. 1-a), then a diversification phase is applied to skip the current region and find another in the solution space (e.g. node 8). The diversification step applied here is different from the original one proposed in [5] and is detailed in the next paragraph. Fig. 1-b shows the evolution of the solution search and the neighborhoods. It is worth mentioning that in the original local branching version, the local branching constraint includes all the binary variables. However in this version, only x_{ij} variables in $MILP^{JH}$, which represent only the vertices matching, are considered. This is due to the fact that matching vertices is the core problem in the GED^{EnA}, and edge matching can easily be deduced after the vertices are matched.

Another improvement to the heuristic is the diversification mechanism, in which only a subset of *important* x_{ij}'s is included in the local branching constraint:

$$\Delta'(x, x^p) = \sum_{(i,j) \in S^p_{imp}} (1 - x_{ij}) + \sum_{(i,j) \notin S^p_{imp}} x_{ij} \geq k\_dv \tag{6}$$

with B_{imp} the index set of binary important variables and $S^p_{imp} = \{(i,j) \in B_{imp} : x^p_{ij} = 1\}$. The selection of these variables is based on the assumption that one variable is considered important if changing its value from $1 \rightarrow 0$ (or the opposite) highly impacts the objective function value. Forcing the solver to find a new solution that changes the values of these variables, enables changing the explored search space. Consequently, B_{imp} is defined as follows: **i)** it computes a special cost matrix $[M_{ij}]$ for each possible assignment of a vertex $u_i \in V \cup \{\epsilon\}$, to a vertex $v_j \in V' \cup \{\epsilon\}$. Each value $M_{ij} = c_{ij} + \theta_{ij}$, where c_{ij} is the vertex edit operation cost induced by assigning vertex u_i to vertex v_j, and θ_{ij} is the cost of assigning the set of edges $E_i = \{(u_i, w) \in E\}$ to $E_j = \{(v_j, w') \in E'\}$. This assignment problem, of size $max(|E_i|, |E_j|) \times max(|E_i|, |E_j|)$, is solved by the Hungarian algorithm [8] which requires $O(max(|E_i|, |E_j|)^3)$ time. **ii)** the standard deviation is computed at each row of $[M_{ij}]$, resulting in a vector $\sigma = [\sigma_1, ..., \sigma_{|V|}]$. Basically, the σ_i's with high values indicate that the variables that represent assignments with vertex u_i have impact on the objective function value more

than the other variables. **iii)** To select u_i vertices, the values of the vector σ are split into two clusters C_{min} and C_{max}, using a simple clustering algorithm. **iv)** Finally, for every σ_i belonging to C_{max} cluster, the indices of all the binary variables x_{ij} that correspond to the assignments of vertex u_i are added to B_{imp}. Consequently, the local structure of a vertex is considered to assess its influence on the objective function value. This version of the diversification is more efficient that the original one proposed by [5], because mainly it includes information about the instance at hand.

	S	10	20	30	40	50	60	70	Mixed
LocBra	t_{avg}	0.17	1.12	212.36	364.86	580.04	753.48	751.44	332.32
	d_{avg}	**0.00**	**0.00**	**0.00**	0.06	**0.02**	**0.17**	**0.59**	**0.04**
	η_I	**100**	**100**	**100**	98	**99**	**93**	**79**	**95**
CPLEX-900	t_{avg}	0.13	1.02	141.07	247.80	451.40	723.68	745.91	305.72
	d_{avg}	**0.00**	**0.00**	**0.00**	**0.00**	0.30	0.55	1.05	0.12
	η_I	**100**	**100**	**100**	**100**	90	81	68	**95**
BeamSearch-5	t_{avg}	**0.00**	**0.00**	**0.01**	**0.03**	**0.07**	**0.11**	**0.18**	**0.09**
	d_{avg}	15.17	36.60	47.21	58.69	72.13	62.96	68.71	21.20
	η_I	35	10	10	10	10	10	10	12
SBPBeam-5	t_{avg}	0.01	0.10	0.45	1.37	3.19	5.56	10.72	3.38
	d_{avg}	20.43	44.90	76.45	82.54	98.90	95.02	94.62	27.16
	η_I	15	10	10	10	10	10	10	10
IPFP-10	t_{avg}	0.01	0.06	0.20	0.30	0.39	0.66	1.05	0.46
	d_{avg}	3.44	10.84	18.31	21.34	22.59	25.9	27.63	7.45
	η_I	69	28	14	11	10	10	10	19
GNCCP-0.1	t_{avg}	0.16	1.30	4.77	11.78	22.08	72.29	111.30	28.53
	d_{avg}	13.29	22.00	26.93	21.69	31.46	21.99	27.61	10.66
	η_I	57	35	6	6	5	9	4	17

Table 1. LocBra vs. literature heuristics on MUTA instances

4 Computational Experiments

Instances and experiment settings. The conducted experiments have been done on reference databases from the Pattern Recognition community, where researches have introduced the graph matching problems. One database of chemical molecules is selected: MUTA [1]. It contains different subsets of small and large graphs and is known to be difficult to solve. It has 7 subsets, each of which has 10 graphs of same size (10 to 70 vertices), plus one more subset of 10 graphs of mixed sizes. Each pair of graphs is considered as an instance. Therefore, MUTA has a total of 800 instances (100 per subset).

To solve the $MILP^{JH}$, solver CPLEX 12.6.0 is used, and LocBra algorithm is implemented in C language. The tests were executed on a machine with the following configuration: Windows 7 (64-bit), Intel Xeon E5 4 cores and 8 GB RAM.

Comparative heuristics. LocBra is evaluated against the following heuristics from the literature: *BeamSearch-*α [9], *SBPBeam-*α [4], *IPFP-it* and *GNCCP-d* [2]. In addition and since LocBra uses CPLEX with a time limit, it is interesting to study the performance of the solver itself with the same time limit in order to see whether the proposed heuristic actually improves the solution of the problem:

	S	10	20	30	40	50	60	70	Mixed
LocBra	t_{avg}	**0.17**	**1.12**	212.36	364.86	580.04	753.48	751.44	332.32
	d_{avg}	**0.00**	**0.00**	**0.00**	0.02	**0.02**	**0.13**	**0.54**	**0.04**
	η_I	**100**	**100**	**100**	99	**99**	**94**	**80**	**97**
BeamSearch-15000	t_{avg}	8.57	80.65	167.48	279.11	439.68	640.29	938.66	828.52
	d_{avg}	1.35	26.66	47.45	52.29	63.98	62.51	63.71	-
	η_I	88	12	10	10	10	10	10	-
SBPBeam-400	t_{avg}	0.84	10.02	47.65	139.75	322.43	590.86	1155	326.64
	d_{avg}	20.43	44.90	76.45	82.45	98.90	94.94	94.54	26.95
	η_I	15	10	10	10	10	10	10	10
IPFP-20000	t_{avg}	1.20	9.62	48.90	115.14	240.54	528.82	903	303.21
	d_{avg}	3.44	10.18	16.45	17.17	19.00	18.99	20.70	6.03
	η_I	69	29	14	11	10	10	10	21
GNCCP-0.03	t_{avg}	0.55	6.41	**29.80**	**81.24**	**195.89**	**396.37**	946.25	**185.55**
	d_{avg}	3.23	6.67	17.20	15.74	18.38	16.12	18.17	5.13
	η_I	81	34	4	6	5	9	5	17

Table 2. LocBra vs. literature heuristics with extended running time on MUTA instances

this heuristic is referred to as *CPLEX-t*. LocBra is applied on all instances with the following parameter values: $k = 20$, $k\_dv = 30$, $total\_time\_limit = 900s$, $node\_time\_limit = 180s$. Similarly, t is set to $900s$ for *CPLEX-t*. For the literature heuristics, which are known to converge fast without having a time limit parameter, two versions are considered. The default versions are the heuristics with the default parameter values as they were originally defined, while the extended versions are with different parameter values that extend their running time to reach about $900s$ on the average. The reason behind this, is to evaluate the quality of the solutions computed by the heuristics when giving the same running time as LocBra. Therefore, the heuristics with default parameters are: *BeamSearch-5*, *SBPBeam-5*, *IPFP-10* and *GNCCP-0.1*. The extended versions: *BeamSearch-15000*, *SBPBeam-400*, *IPFP-20000* and *GNCCP-0.03*. All parameter values are set based on preliminary experiments that are not shown here.

For each heuristic, 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 percentages between the solutions obtained by one heuristic, and the best solutions found. Given an instance I and a heuristic H, the deviation percentage is equal to $\frac{solution_I^H - bestSolution_I}{bestSolution_I} \times 100$, with $bestSolution_I$ the smallest solution value found by all heuristics for I. Lastly, η_I is the number of instances for which a given heuristic has found the best solutions.

Results and analysis. Table 1 shows the results of the heuristics with their default parameter values. Considering the d_{avg} metric, *LocBra* and *CPLEX-900* seem to have scored the lowest values among the heuristics. *LocBra* performed better on hard instances (subsets 50, 60 and 70), than *CPLEX-900*. The highest value obtained by *LocBra* is 0.59%, which means it computed the best/smallest solutions. The same conclusion is seen when looking at the η_I values. The literature heuristics are strongly outperformed by *LocBra* with very high d_{avg} values. However, in terms of CPU time, *BeamSearch-5* seems to be the fastest with smallest t_{avg} values. On the other hand, Table 2 reports the results obtained by the extended versions of the heuristics. Again, *LocBra* appears to be the best heuristic in terms of of deviations (always less than 0.6%).

5 Conclusions

This work presents a local branching heuristic for the GED^{EnA} problem, which significantly improves the literature heuristics and provides near optimal solutions. An important factor is the diversification procedure that is problem dependent and really helps escaping local optima. Next, more techniques will be investigated in order to boost the solution of the method and also to allow dealing with graphs that have attributes on their edges.

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. Bougleux, S., Brun, L., Carletti, V., Foggia, P., Gaüzère, B., Vento, M.: Graph edit distance as a quadratic assignment problem. Pattern Recognition Letters (2016), //www.sciencedirect.com/science/article/pii/S0167865516302665
3. Bunke, H.: On a relation between graph edit distance and maximum common subgraph. Pattern Recognition Letters 18(8), 689–694 (1997)
4. Ferrer, M., Serratosa, F., Riesen, K.: Improving bipartite graph matching by assessing the assignment confidence. Pattern Recognition Letters 65, 29–36 (2015)
5. Fischetti, M., Lodi, A.: Local branching. Mathematical programming 98(1-3), 23–47 (2003)
6. 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)
7. Lerouge, J., Abu-Aisheh, Z., Raveaux, R., Héroux, P., Adam, S.: Exact graph edit distance computation using a binary linear program. In: Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR). pp. 485–495. Springer (2016)
8. Munkres, J.: Algorithms for the assignment and transportation problems. Journal of the society for industrial and applied mathematics 5(1), 32–38 (1957)
9. Neuhaus, M., Riesen, K., Bunke, H.: Fast suboptimal algorithms for the computation of graph edit distance. In: Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR). pp. 163–172. Springer (2006)
10. 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)
11. 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)
12. Sanfeliu, A., Alquézar, R., Andrade, J., Climent, J., Serratosa, F., Vergés, J.: Graph-based representations and techniques for image processing and image analysis. Pattern recognition 35(3), 639–650 (2002)
13. Serratosa, F.: Fast computation of bipartite graph matching. Pattern Recognition Letters 45, 244–250 (2014)
14. Zeng, Z., Tung, A.K., Wang, J., Feng, J., Zhou, L.: Comparing stars: on approximating graph edit distance. Procee. of the VLDB Endowment 2(1), 25–36 (2009)