Sub-optimal Graph Matching by Node-to-node Assignment Classification

Xavier Cortés, Donatello Conte, Francesc Serratosa

▶ **To cite this version:**

Xavier Cortés, Donatello Conte, Francesc Serratosa. Sub-optimal Graph Matching by Node-to-node Assignment Classification. Donatello Conte; Jean-Yves Ramel; Pasquale Foggia. Graph-Based Representations in Pattern Recognition, 11510, Springer, pp.35-44, 2019, Lecture Notes in Computer Science, 978-3-030-20080-0. 10.1007/978-3-030-20081-7_4 . hal-02285797

HAL Id: hal-02285797
https://hal.science/hal-02285797

Submitted on 13 Sep 2019

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.

Sub-optimal Graph Matching by Node-to-node Assignment Classification

Xavier Cortés[1], Donatello Conte[1], and Francesc Serratosa[2]

[1] Laboratoire d'Informatique Fondamentale et Appliquée de Tours
(LIFAT - EA6300), Tours, France
xavier.cortes@univ-tours.fr, donatello.conte@univ-tours.fr
[2] Universitat Rovira i Virgili
Tarragona, Catalonia, Spain
francesc.serratosa@urv.cat

Abstract. In the recent years, Graph Edit Distance has awaken interest in the scientific community and some new graph-matching algorithms that compute it have been presented. Nevertheless, these algorithms usually cannot be used in real applications due to runtime restrictions. For this reason, other graph-matching algorithms have also been used that compute an approximation of the graph correspondence with lower runtime. Clearly, in a real application, there is a tradeoff between runtime and accuracy. One of the most costly part in these algorithms is the deduction of the node-to-node mapping. We present a new graph-matching algorithm that returns a graph correspondence without the explicit computation of the assignment problem. This is done thanks to a classification of the node-to-node assignment learned in a previous training stage.

Keywords: Graph edit distance, Node assignment classification, Graph embedding, Graph matching

1 Introduction

Attributed graphs have found widespread applications in several research fields of structural pattern recognition [1–3]. This is due to their ability to represent structured objects through unary and binary local entities. To compare them, several distance measures between attributed graphs have been presented [2, 3]. Among them, one of the most used distances is the Graph Edit Distance [4, 5].

Typically, the problem is mathematically formulated as a quadratic assignment problem, which consists of finding the node-to-node assignment that minimizes an objective function encoding local dissimilarities (a linear term) and structural dissimilarities (a quadratic term). To do so, it is needed to define the cost functions between the linear terms and also the quadratic terms, given the application at hand. Note that a proper definition of these cost functions is crucial to achieve good classification or recognition results. For this reason, several methods have been presented to learn these edit costs [6, 7]. Moreover, the Graph Edit Distance has been demonstrated to be an NP-hard problem. For

this reason, several algorithms that return a graph correspondence in polynomial time with respect to the number of nodes at the expense of not having the certainty of being the correspondence that minimizes the graph edit distance, have been presented [8–12].

In this paper, we present another graph matching algorithm that deduces the graph correspondences in a sub-optimal way, as the ones commented above. The novelty of our algorithm is that we classify the node-to-node assignments at a first step and then there is no need of using a combinatorial optimization algorithm (such as the Hungarian method [13]) to deduce the graph correspondence, thus avoiding their computational cost.

2 Definitions

2.1 Attributed Graphs and Graph Edit Distance

We define an attributed graph G as a quadruplet $G = \{\sum_v, \sum_e, \gamma_v, \gamma_e\}$ where $\sum_v = \{v_i | i = 1...n\}$ is the set of n nodes and $\sum_e = \{e_{i,j} | i, j \in 1...n\}$ is the set of edges connecting pairs of nodes. $\gamma_v = \{v_i \rightarrow \psi_i | i = 1..n\}$ and $\gamma_e = \{v_i \rightarrow E(v_i) | i = 1..n\}$ are functions that map the nodes and edges to their attribute values, respectively. $\psi_i \in \mathbb{R}^m$ maps each node to its m local attributes and $E(\cdot)$ refers to the degree of a certain node [14, 15]. For simplicity, in this paper we only consider undirected and unattributed edges. However, all the concepts presented in this paper could be extended to directed and attributed edges.

The Graph Edit Distance (GED) [4] is a distance between two attributed graphs G and G'. It consists of the best combination of edit operations that transform G into G'. Three operations are considered on the local attributes of the nodes and also on its structures: Substitution, deletion and insertion. To quantify the degree of distortion that each edit operation introduces, a cost is assigned to them depending on the attributes on the involved nodes or edges. A sequence of edit operations that completely transform G into G' is referred to as edit path $\lambda_{G,G'}$ between G and G'. The cost of an edit path is the sum of the costs of the edit operations included on it. Thus, the GED is defined as the edit path from one graph into another that obtains the minimum cost under all possible edit paths $\mathcal{T}_{G,G'}$ between G and G'. Formally:

$$GED(G, G') = \min_{\lambda_{G,G'} \in \mathcal{T}_{G,G'}} Cost(G, G', \lambda_{G,G'}) \tag{1}$$

The edit path $\lambda_{G,G'}$ can be defined through a node-to-node matching f between nodes of both graphs where $f(v_i) = v'_a$. Graphs can be enlarged by null nodes to assure having the same order.

2.2 Approximating the Graph Edit Distance

The graph-matching algorithms that return the GED are based on exploring all the combinations of correspondences f between G and G' and selecting the one

with the minimum cost. However, several approaches have appeared to reduce the computational complexity of the GED computation [16, 9, 17, 18] at the expense of returning a sub-optimal correspondence. Usually, these algorithms are based on two main steps:

In the first step, a cost matrix is filled with the edit cost between all combinations of the local structures of both graphs. The computational cost of this step is approximated by $O(s \times n^2)$, where s is the computational cost of mapping the local structures.

In the second step, a node-to-node matching f is found. The problem at hand is seen as a minimization of the sum of the linear assignation given the cost matrix. Thus, the computational cost of this second step is $O(n^3)$ or $O(n^2)$ depending on whether the matching is deduced by the Bipartite graph matching [9, 18], or the Greedy edit matching [17, 14, 8, 16], respectively. In the case of the Bipartite graph matching, it is usually solved through the Munkres or Jonker-Volgenant algorithms [13, 19]. In another case, the matching between nodes f is obtained through an algorithm that iteratively selects the minimum value per row and discards the selected columns for the remaining rows.

3 Learning Graph Matching

We propose a sub-optimal graph matching algorithm which avoids the second step in the classical sub-optimal graph matching algorithms (see 2.2). This is because the second part turns out to be more expensive than the first part, from the computational time point of view. In the next section, we list the known methods that learn the edit costs, from which our method is inspired.

3.1 Learning the edit costs and Graph embedding

Several methods have been presented to learn the edit costs based on supervised machine learning techniques, which can be divided in two main groups. The ones that return a constant on the edit operations [20–24], and the other ones that define the edit costs as functions. For instance, in [25], they use a probabilistic model of the distribution of graph edit operations. Another paper is based on a self-organizing map model [26] in which the edit costs are the output of a Neural Network. In both papers, the learning set is composed of classified graphs and the edit costs are optimized with regard to Dunns index [27]. Recently, two new papers assume the cost matrix could filled as the output of a supervised machine learning model. In [7], the authors use a Neural Network to learn only the substitution costs (no insertion nor deletion operations are allowed). And in [6], a general framework is presented to learn and define this costs.

3.2 From Edit Costs Estimation to Node Assignment Classification

Inspired by methods such as the ones in [7, 6], we propose a supervised machine learning model that splits the node-to-node assignments in two classes, depending whether the learning database considers that they have to be mapped in f

or not. Note that in [7, 6], the learning algorithms deduce edit costs instead of discerning between two classes. The key idea of our model is to decide if a node in G is mapped to a node in G' using a classifier. Then, the classes *mapped* or *non-mapped* are assigned considering the output of a previously trained classifier. Note that a node could remain unmapped in f if it is classified as *non-mapped* in all cases. We treat this particular case as a deletion or an insertion of the corresponding node.

Our method is independent of the classification method (Support Vector Machine, K-Nearest Neighbours, Neural Network, etc.), however, in any case we need to transform each node-to-node mapping into a vector that becomes the input of the classifier. Thus, we propose to embed each matching into a vector, similar as proposed in [7]. In this case, the embedded representation of a mapping between two nodes v_i and v'_a of G and G', is $x_{v_i \to v'_a} = [\gamma_v(v_i), \gamma_e(v_i), \gamma'_v(v'_a), \gamma'_e(v'_a)] \in \mathbb{R}^{(m+1)\cdot 2}$.

Our matching algorithm is shown in Algorithm 1. It is a greedy algorithm that goes across all nodes $v_i \in G$ and, for each of them, deduces the first node $v'_a \in G'$ that can be mapped to v_i. Note that this strategy avoids: a) The explicit computation of the graph correspondences in the second step of the classical sub-optimal graph matching algorithms. b)The whole computation of the cost matrix in the first step of the classical sub-optimal graph matching algorithms. Note that in our case, instead of having a cost matrix, we have a classification matrix.

Nevertheless, it is important to remark that the model does not return a distance but only a graph correspondence. Moreover, the performance of the model depends on the quality of the classifier.

Algorithm 1: Graph matching based on node assignment classification.

```
   Data: graph G and graph G'
   Result: matching f
 1 f = empty node-to-node correspondence
 2 forall v_i in G do
 3 |   forall v'_a in G' and not in f do
 4 |   |   x = embed(v_i → v'_a)
 5 |   |   y = class_predictor(x)
 6 |   |   if y = mapped then
 7 |   |   |   f(v_i) = v'_a
 8 |   |   |   break_loop
 9 |   |   end
10 |   end
11 end
```

Figure 1 (on the left) shows a pair of graphs (blue and red) and the optimal graph correspondence (green arrows), edges do not have attributes and the node

edit cost is the distance between attributes. In the center of Figure 1 the cost matrix computed in the first step of a sub-optimal graph-matching algorithm (Section 2.2) is shown. Finally, Figure 1 (on the right) shows the outputs of the classifier (T: *mapped* or F: *non-mapped*) by our proposed algorithm. Grey cells are the node pairs that our algorithm needs to analyze. In our example, only 7 values have been computed. Being 4 the minimum and 10 the maximum number in the worst case (n and $n(n+1)/2$, respectively, where n is the number of nodes).

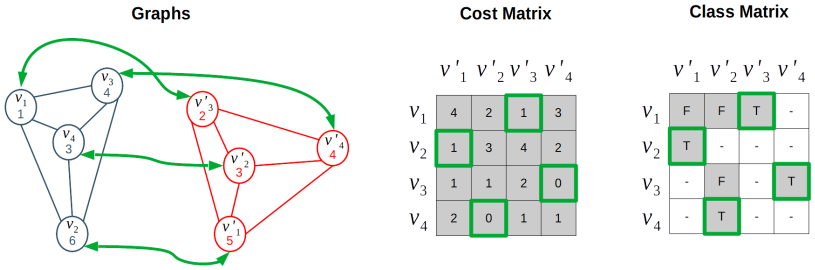

Fig. 1. Left: A pair of graphs (blue and red) and the optimal correspondence (green arrows). Centre: The cost matrix. Right: The computed classes (T: *mapped* or F: *non-mapped*). Highlighted in green: Node-to-node mappings. Grey: Computed values (nodes are processed consecutively from 1 to 4).

3.3 Training the Classifier

In supervised machine learning, databases entries are composed of an element and its expected outputs. In our case, an entry p in the database is composed of a pair of graphs $(G^p, G^{p\prime})$ and their ground-truth correspondence $\hat{f}^p$. These correspondences $\hat{f}^p$ have been deduced by an external system (typically a human expert) and they are considered to be the best mappings for our learning purposes. The aim of the learning method is to define a model that, given a pair of nodes, returns the class *mapped* when the two nodes are mapped in the ground-truth, and *non-mapped* otherwise.

To do so, we define two sets of node-to-node mappings: The one with node pairs that have to be mapped according the ground-truth correspondences and the set of node pairs assumed they do not have to be mapped. Assuming that the ground-truth correspondences are bijective functions, each node of G is mapped to a single node of G', while it has not to be mapped to the rest of nodes of G'. This means that for each node-to-node mapping classified as *mapped*, there are $n-1$ node-to-node mappings classified as *non-mapped*, where n is the order of the graphs. In order to prevent imbalance problem, the node-to-node mappings in *mapped* are repeated $n-1$ times when we feed the training algorithm.

4 Experimental Evaluation

In this section, we show the performance of our graph-matching algorithm in terms of accuracy and runtime. To do so, we compare our results to different existing approaches in the literature. In Section 4.1, we describe the databases used in our experiments while in Section 4.2, we present the computed accuracy and runtime. Finally, in Section 4.3, we evaluate the performance of our method using synthetically generated graphs to analyze how the graph order affects our model. Experiments were conducted using MATLAB 2018 on a Windows 10, with an Intel i5 processor at 1.6 GHz and 8GB of RAM.

4.1 Database Description

We used the House-Hotel [28] and a database synthetically generated by the method in [29]. The House-Hotel database [28] consists of two sequences of frames showing two computer modeled objects, 111 frames of a House and 101 frames of a Hotel moving and rotating on its own axis throughout the scene. Each frame has 30 salient points manually labelled. Each salient point represents a graph node and it is attributed by 60 Context Shape features. The salient points has been triangulated by Delaunay triangulation according to its coordinates to build the edges. Since the salient points are manually labelled, we know the ground-truth correspondence between the nodes of the graphs. As more separated are the frames that represents each graph, the differences between graphs increase and consequently, more difficult is the graph matching process. We performed different experiments changing the number of frames of separation between the frames in the video sequence. For each experiment, we built three sub-sets of graph pairs (train, validation and test).

The synthetic database was composed of several sets of graphs that had the same order. We generated different pairs of graph prototypes inspired by the method detailed in [29]. Nodes has four Integer attributes randomly generated in a range from 0 to 999. Edges are unattributed and it was imposed a probability of 20% for each pair of nodes to be connected by an edge. Next, a collection of pairs of graphs has been generated by distorting original prototypes adding some Gaussian noise (Standard deviation: 500) to the last attribute of each node.

4.2 Graph Matching Performance

We analyzed the graph matching accuracy and the time spent to perform it. The matching accuracy is defined as the number of node-to-node mappings in the deducted correspondences that are equal to the node-to-node mappings in the ground-truth correspondences, normalized by the graph order.

Our method was implemented with two different classifiers: a) A Neural Network with one hidden layer that has 60 neurons and an output layer with two neurons (a neuron per class: *mapping* and *non-mapping*). b) A Quadratic Support Vector Machine, which classifies the node-to-node mappings among the two classes: *mapping* and *non-mapping*.

Moreover, we compared our approach to four graph-matching methods (Table 1). In the first two methods, the cost function is defined as an edit cost based on the Euclidean distance between the node attributes plus the difference of the degree of these attributes, as it was done in [14, 15]. In the other two, the cost function is defined as the output of a Neural Network previously trained as in [7].

Table 1. Graph matching methods used in this paper for comparative purposes.

Cost Function	Solver	Complexity	Year	Reference
Edit Cost	Hungarian	$O(n^3)$	2009	[9]
Edit Cost	Greedy	$O(n^2)$	2017	[16]
Neural Network	Hungarian	$O(n^3)$	2018	[7]
Neural Network	Greedy	$O(n^2)$	-	-

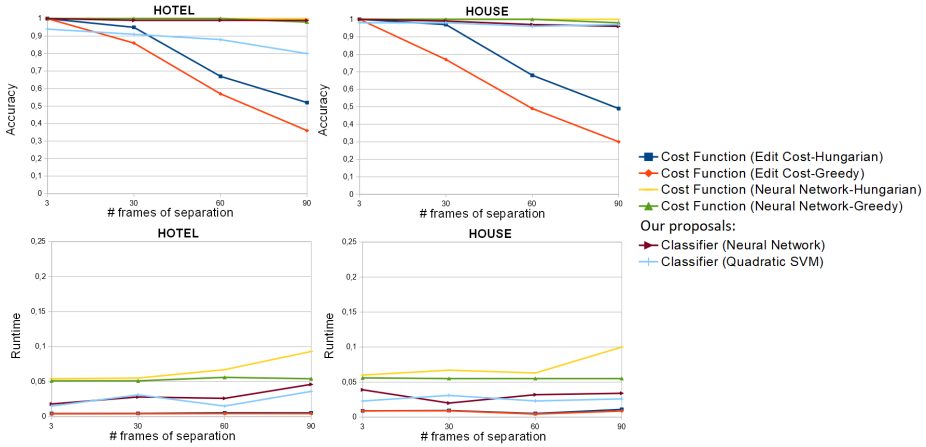

Fig. 2. Matching accuracy (top) and Runtime in seconds (bottom) versus number of frames of separation. Database: Hotel-House.

Figure 2 (top) shows the matching accuracy versus the number of frames of separation using the Hotel-House database. When the number of frames of separation increases, graphs tend to be more different and this fact is reflected on the the accuracy decrease. On the bottom of the Figure 2 the mean runtime spent to perform a matching between two graphs (in seconds) is shown. There is a slight tendency of increasing the runtime.

In these experiments, our method shows a good balance on runtime and matching accuracy. The accuracy is similar to the Neural Network methods [7] but the runtime is lower. Moreover, the GED methods [9, 16] are faster but return a worse matching accuracy when the number of frames of separation increases.

4.3 Runtime analysis

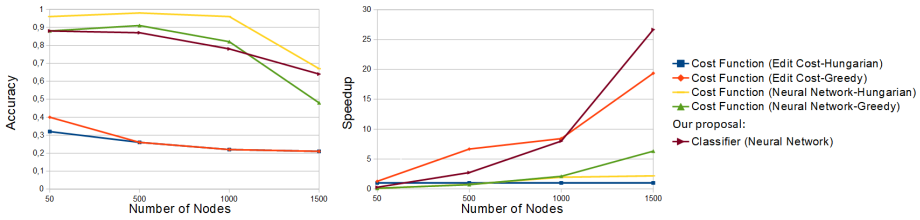

Fig. 3. Matching accuracy and speedup with respect to the GED-Hungarian versus number of nodes per graph. Database: Synthetic.

We analyzed the relevance of the graph order with regard to the performance of the model in the synthetic database. Our model was the one implemented with a Neural Network that has 6 neurons in the hidden layer. The Quadratic Support Vector Machine was not used in this experiment due to the high computational time necessary in the learning step.

Figure 3 (left) shows the accuracy versus the number of nodes of the graphs. We observe that there is an important decrease in the accuracy when the order of the graphs achieves 1500 nodes. Moreover, the methods that use edit costs return low accuracies given any graph order.

Figure 3 (right) shows the speedup of each graph matching alg: $Speedup_{alg} = Runtime_{[9]}/Runtime_{alg}$, where $Runtime_x$ is the runtime of algorithm x. We observe that the speedup of our model grows faster than any other method when we increase the graphs order. We achieve the best results in terms of speedup and the second best accuracy when the graphs order is 1500 nodes.

We observed that there is an extra computational cost when using a classifier like a Neural Network instead of a cost function such as the Edit Costs due to the number of internal operations that have to be carried out. For this reason, when the graph order is small, there is no improvement in terms of runtime with respect to the methods that use Edit Costs and the Hungarian solver. However, for larger graphs, the increase of runtime due to the classifier is compensated by the fact that we do not need to evaluate all node-to-node correspondences and it is not necessary to solve the assignment problem either.

5 Conclusions

In this paper, we present a fast approach to deduce the graph correspondence. Previous methods are based on two steps. In the first one, a cost matrix is filled and in the second one, a linear solver is applied on it to deduce the graph correspondence. In our proposal, we do not need this second step since the first one, the node-to-node mapping, is directly deduced. To do so, we have used a

classifier that separates the node-to-node assignment in two classes: *mapped* and *non-mapped*. In the experimental section, we show that our method achieves a good accuracy with a low matching runtime, comparing it to four existing methods. The experiments show a larger decrease of runtime, compared to the other methods, when the graph order increases. This allows to compute matchings between graphs with the proposed method for very large graphs in an acceptable computational time.

Acknowledgements

This research is supported by projects TIN2016-77836-C2-1-R and DPI2016-78957-R and AEROARMS (H2020-ICT-2014-1-644271 (Spain), and by DANIEAL2 project of Centre-Val-de-Loire Region (France).

References

1. Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, 18(03):265–298, 2004.
2. Lorenzo Livi and Antonello Rizzi. The graph matching problem. *Pattern Analysis and Applications*, 16(3):253–283, 2013.
3. Pasquale Foggia, Gennaro Percannella, and Mario Vento. Graph matching and learning in pattern recognition in the last 10 years. *International Journal of Pattern Recognition and Artificial Intelligence*, 28(01):1450001, 2014.
4. Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129, 2010.
5. Francesc Serratosa. Graph edit distance: restrictions to be a metric. *Pattern Recognition*, 90:250–256, 2019.
6. Pep Santacruz and Francesc Serratosa. Learning the sub-optimal graph edit distance edit costs based on an embedded model. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 282–292. Springer, 2018.
7. Xavier Cortés, Donatello Conte, Hubert Cardot, and Francesc Serratosa. A deep neural network architecture to estimate node assignment costs for the graph edit distance. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 326–336. Springer, 2018.
8. Andreas Fischer, Kaspar Riesen, and Horst Bunke. Improved quadratic time approximation of graph edit distance by combining hausdorff matching and greedy assignment. *Pattern Recognition Letters*, 87:55–62, 2017.
9. Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision computing*, 27(7):950–959, 2009.
10. Francesc Serratosa and Xavier Cortés. Interactive graph-matching using active query strategies. *Pattern Recognition*, 48(4):1364–1373, 2015.
11. Miquel Ferrer, Francesc Serratosa, and Kaspar Riesen. Improving bipartite graph matching by assessing the assignment confidence. *Pattern Recognition Letters*, 65:29–36, 2015.

12. Francesc Serratosa. Speeding up fast bipartite graph matching through a new cost matrix. *International Journal of Pattern Recognition and Artificial Intelligence*, 29(02):1550010, 2015.
13. J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society of Industrial and Applied Mathematics*, 5(1):32–38, March 1957.
14. Xavier Cortés, Francesc Serratosa, and Kaspar Riesen. On the relevance of local neighbourhoods for greedy graph edit distance. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 121–131. Springer, 2016.
15. Francesc Serratosa and Xavier Cortés. Graph edit distance: Moving from global to local structure to solve the graph-matching problem. *Pattern Recognition Letters*, 65:204–210, 2015.
16. Andreas Fischer, Kaspar Riesen, and Bunke Horst. Improved quadratic time approximation of graph edit distance by combining hausdorff matching and greedy assignment. *Pattern Recognition Letters*, 87:55–62, 2017.
17. Kaspar Riesen, Miquel Ferrer, Rolf Dornberger, and Horst Bunke. Greedy graph edit distance. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 3–16. Springer, 2015.
18. Francesc Serratosa. Fast computation of bipartite graph matching. *Pattern Recognition Letters*, 45:244–250, 2014.
19. Roy Jonker and Ton Volgenant. Improving the hungarian assignment algorithm. *Oper. Res. Lett.*, 5(4):171–175, October 1986.
20. Tibério S Caetano, Julian J McAuley, Li Cheng, Quoc V Le, and Alex J Smola. Learning graph matching. *IEEE Trans. on PAMI*, 31(6):1048–1058, 2009.
21. Marius Leordeanu, Rahul Sukthankar, and Martial Hebert. Unsupervised learning for graph matching. *International journal of computer vision*, 96(1):28–45, 2012.
22. Xavier Cortés and Francesc Serratosa. Learning graph-matching edit-costs based on the optimality of the oracle's node correspondences. *Pattern Recognition Letters*, 56:22–29, 2015.
23. Xavier Cortés and Francesc Serratosa. Learning graph matching substitution weights based on the ground truth node correspondence. *IJPRAI*, 30(02):1650005, 2016.
24. Shaima Algabli and Francesc Serratosa. Embedding the node-to-node mappings to learn the graph edit distance parameters. *Pattern Recognition Letters*, 112:353–360, 2018.
25. Michel Neuhaus and Horst Bunke. Automatic learning of cost functions for graph edit distance. *Information Sciences*, 177(1):239–247, 2007.
26. Michel Neuhaus and Horst Bunke. Self-organizing maps for learning the edit costs in graph matching. *IEEE Trans. on SMC, Part B*, 35(3):503–514, 2005.
27. J. C. Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4(1):95–104, 1974.
28. Carlos Francisco Moreno-García, Xavier Cortés, and Francesc Serratosa. A graph repository for learning error-tolerant graph matching. *Proceedings of SSPR*, 2016.
29. Francesc Serratosa. A methodology to generate attributed graphs with a bounded graph edit distance for graph-matching testing. *IJPRAI*, 32(11):1850038, 2018.