Learning Graph Matching with a Graph-Based Perceptron in a Classification Context

Romain Raveaux[1], Maxime Martineau[1], Donatello Conte[1], and Gilles Venturini[1]

Université de Tours, Laboratoire d'Informatique (EA 6300), 64 avenue Jean Portalis, Tours, France
maxime.martineau@univ-tours.fr

Abstract. Many tasks in computer vision and pattern recognition are formulated as graph matching problems. Despite the NP-hard nature of the problem, fast and accurate approximations have led to significant progress in a wide range of applications. Learning graph matching functions from observed data, however, still remains a challenging issue. This paper presents an effective scheme to parametrize a graph model for object matching in a classification context. For this, we propose a representation based on a parametrized model graph, and optimize it to increase a classification rate. Experimental evaluations on real datasets demonstrate the effectiveness (in terms of accuracy and speed) of our approach against graph classification with hand-crafted cost functions.

Keywords: Learning graph matching, graph classification, graph edit distance

1 Introduction

Graphs are frequently used in various fields of computer science since they constitute a universal modeling tool which allows the description of structured data. The handled objects and their relations are described in a single and human-readable formalism. Hence, tools for graphs supervised classification and graph mining are required in many applications such as pattern recognition [1], chemical components analysis [2], structured data retrieval [3]. Different approaches have been proposed during the last decade to tackle the problem of graph classification. A first one consists in transforming the initial problem in a common statistical pattern recognition one by describing the graphs with vectors in a Euclidean space [2]. Another family of approaches also consists in using kernel-based machine learning algorithms. Contrary to the approaches mentioned above, the graphs are not explicitly but implicitly projected in a Euclidean space, through the use of a graph kernel which computes inner products in the graph space. Many kernels have been proposed in the literature [4]. Another possible approach also consists in projecting the graphs in a Euclidean space of a given dimension but using a distance matrix between each of the graphs. In such cases, a dissimilarity measure between graphs has to be designed. Kernels can be derived

from the distance matrix. It is the case for multidimensional scaling methods proposed in [5].

All the aforementioned approaches aim at projecting the graphs into a vector or kernel space however this process may impact the interpretability of the results. This paper deals with paradigms that operate directly on the graph space and can thus capture more structural distortions.

Graph space ($d : G \times G \rightarrow \mathbb{R}$). To classify unknown objects using the K-Nearest Neighbor paradigm, one needs to define a metric that measures the distance between the unknown object and the elements in the learning database. The similarity or dissimilarity between two graphs requires the computation and the evaluation of the "best" matching between them. Since exact isomorphism rarely occurs in pattern analysis applications, the matching process must be error-tolerant, i.e., it must tolerate differences on the topology and/or its labeling. For instance, in the Graph Edit Distance (GED) [1], the graph matching process and the dissimilarity computation are linked through the introduction of a set of graph edit operations. Each edit operation is characterized by a cost, and GED is the total cost of the least expensive set of operations that transform one graph into another one. Since graph matching is NP-hard most research has long focused on developing accurate and efficient approximate algorithms.

Recent studies have revealed that simple graphs with hand-crafted structures and dissimilarity functions, typically used in graph matching, are insufficient to capture the inherent structure underlying the problem at hand. As a consequence, a better optimization of the graph matching objective does not guarantee better correspondence accuracy [6, 7] and neither better classification rate. To tackle this issue a set of parameters in the graph matching problem has to be learned. Such a learned matching function would better model the inherent structure of the classification problem without losing the iterpretability of the results.

2 Problem statement

In this section, we formally define the problem of learning discriminative graph matching.

Attributed graph is considered as a triple (V, E, L) such that: V is a set of vertices. E is a set of edges such as $E \subseteq V \times V$. L is a set of attributes of the nodes and edges. For the sake of clarity, we abuse the set notation such that L_i is a label associated to vertex v_i and L_{ij} is a label associated to an edge (v_i, v_j).

Graph Matching problem Let $G^1 = (N^1, E^1, L^1)$ and $G^2 = (N^2, E^2, L^2)$ be two graphs, with $N^1 = \{1, \cdots, n\}$ and $N^2 = \{1, \cdots, m\}$. In order to apply deletion or insertion operation on nodes, node sets are augmented by dummy elements. The deletion of each node $v_i \in N^1$ is modeled as a mapping $v_i \rightarrow \epsilon_i^2$ where ϵ_i^2 is the dummy element associated with v_i. As a consequence, the set N^2 is increased by $\max(0, n-m)$ dummy elements ϵ^2 to form a new set $V^2 = N^2 \cup \epsilon^2$. The node set N^1 is increased similarly by $\max(0, m-n)$ dummy elements ϵ^2

to form $V^1 = N^1 \cup \epsilon^1$. Note that V^1 and V^2 have the same cardinality ($n1 = n2 = \max(n, m)$). A solution of graph matching is defined as a subset of possible correspondences $y \subset V^1 \times V^2$, represented by a binary assignment matrix $Y \in \{0,1\}^{n_1 \times n_2}$, where n_1 and n_2 denote the size of V^1 and V^2, respectively. If $v_i^1 \in V^1$ matches $v_a^2 \in V^2$, then $Y_{i,a} = 1$, and $Y_{i,a} = 0$ otherwise. We denote by $y \in \{0,1\}^{n_1 n_2}$, a column-wise vectorized replica of Y. With this notation, graph matching problems can be expressed as finding the assignment vector y^* that minimizes a score function $d(G^1, G^2, y)$ as follows:

Definition 1. *Graph Matching formulation*

$$y^* = \underset{y}{\operatorname{argmin}} \quad d(G^1, G^2, y), \tag{1a}$$

$$\textit{subject to} \quad y \in \{0,1\}^{n_1 n_2} \tag{1b}$$

$$\sum_{i=1}^{n_1} y_{i,a} = 1 \quad \forall a \in [1, \cdots, n_2] \tag{1c}$$

$$\sum_{a=1}^{n_2} y_{i,a} = 1 \quad \forall i \in [1, \cdots, n_1] \tag{1d}$$

The function $d(G^1, G^2, y)$ measures the dissimilarity of graph attributes, and is typically decomposed into a first order dissimilarity function $d_V(L_i^1, L_a^2)$ for a node pair $v_i^1 \in V^1$ and $v_a^2 \in V^2$, and a second-order similarity function $d_E(L_{ij}^1, L_{ab}^2)$ for an edge pair $e_{ij}^1 \in E^1$ and $e_{ab}^2 \in E^2$. Dissimilarity functions are usually represented by a symmetric dissimilarity matrix D, where a non-diagonal element $D_{ia;jb} = d_E(L_{ij}^1, L_{ab}^2)$ contains the edge dissimilarity of two correspondences (v_i^1, v_a^2) and (v_j^1, v_b^2) and a diagonal term $D_{ia;ia} = d_V(L_i^1, L_a^2)$ represents the node dissimilarity of a correspondence (v_i^1, v_a^2).

Thus, the matching function of graph matching is defined as:

$$d(G^1, G^2, y) = \sum_{y_{ia}=1} d_V(L_i^1, L_a^2) + \sum_{y_{ia}=1} \sum_{y_{jb}=1} d_E(L_{ij}^1, L_{ab}^2) = y^T D y \tag{2}$$

In essence, the score accumulates all the dissimilarity values relevant to the assignment. The Definition 1 is referred to as an integer quadratic programming. More precisely, it is the quadratic assignment problem, which is known to be NP-hard. Many efficient approximate algorithms have been proposed for this problem [8–11].

Parametrized Graph Matching In the context of scoring functions defined in Eq. 2, an interesting question is what can be learned to improve graph matching. To address this, we parameterize Eq. 2 as follows. Let $\pi(a) = i$ denote an assignment of node v_a^2 in G^2 to node v_i^1 in G^1, i.e. $y_{ia} = 1$. A joint feature map

$\Phi(G^1, G^2, y)$ is defined by aligning the relevant dissimilarity values of Eq.2 into a vectorial form as: $\Phi(G^1, G^2, y) = [\cdots, d_V(L^1_{\pi(a)}, L^2_a), \cdots, d_E(L^1_{\pi(a)\pi(b)}, L^2_{ab}), \cdots]$

By introducing weights on all elements of this feature map, we obtain a discriminative score function:

$$d(G^1, G^2, y, \beta) = \beta\Phi(G^1, G^2, y) \tag{3a}$$

$$= [\cdots, d_V(L^1_{\pi(a)}, L^2_a)\beta_a, \cdots, d_E(L^1_{\pi(a)\pi(b)}, L^2_{ab})\beta_{ab}, \cdots] \tag{3b}$$

where β is a weight vector encoding the importance of node and edge dissimilarity values. In the case of uniform weights, i.e. $\beta = 1\ \forall\beta$, Eq. 3 it reduces to the conventional graph matching score function of Eq. 2: $d(G^1, G^2, y) = d(G^1, G^2, y; 1)$.

The discriminative weight formulation is general in the sense that it can assign different parameters for individual nodes and edges. However, it does not learn a graph model underlying the feature map, and requires a reference graph G^2 at query time, whose attributes cannot be modified in the learning phase.

Graph classification problem For sake of clarity, the rest of the paper is focused on a 2-class problem but the paradigm can be extended to a multi-class problem. A linear classifier is a function that maps its input $x \in \mathbb{R}^q$ (a real-valued vector) to an output value $f(x) \in \{0, 1\}$ (a single binary value).

$$f(x) = \begin{cases} 1 & \text{if } \beta \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

where β is a vector of real-valued weights, $w \cdot x$ is the dot product $\sum_{i=1}^{q} \beta_i x_i$, where q is the number of inputs to the classifier and b is the bias. The bias shifts the decision boundary away from the origin and does not depend on any input value. The value of $f(x)$ (0 or 1) is used to classify x as either a positive or a negative instance, in the case of a binary classification problem. If b is negative, then the weighted combination of inputs must produce a positive value greater than $|b|$ in order to push the classifier over the 0 threshold.

To extend this paradigm to graph, let $\mathcal{D}$ be the set of graphs. Given a graph training set $TrS = \{< G_i, c_i >\}_{i=1}^{M}$, where $G_i \in \mathcal{D}$ is a graph and $c_i \in \mathcal{C}$ is the class of the graph among the two classes. The learning of a graph classifier consists in inducing from TrS a mapping function $f(G) \rightarrow \mathcal{C}$ which assigns a class to an unknown graph.

$$f(G) = \begin{cases} 1 & \text{if } \beta \cdot \Phi(G, G^m, y) + b > 0 \\ 0 & \text{otherwise} \end{cases} \quad \text{With } G^m \text{ a model graph}$$

Let $\Delta(TrS, f)$ be a function computing the error rate obtained by a classifier f. We represent the error for the p^{th} training sample by $error_p = \delta(C_p, f(G_p))$, where C_p is the target value, $f(G_p)$ is the value produced by the classifier and

$\delta(a, b)$ is the Kronecker Delta function. The error rate (Δ) is the mean of errors $error_p$ over the set TrS between the ground-truth values and values produced by the classifier. Straightforwardly, we define $\eta = 1 - \Delta$ as the classification rate.

To address the problem of learning graphs matching, we start with the discriminative weight formulation of Eq.3. We learn the weights β from labelled examples from TrS minimizing the function Δ. The objective function is the error rate function with extra β weights.

3 State of the art

The literature on learning similarity/dissimilarity matching functions can be roughly categorized into two parts whether the objective is to minimize an error rate on the number of matched graph components (matching level) or an error rate on a classification task (classification level).

Matching level. In this category, the purpose is to minimize the average Hamming distance between a ground-truth 's correspondence and the automatically deducted correspondence. Caetano et al. [6] use a 60-dimensional node similarity function for appearance similarity and a simple binary edge similarity for edges. Leordeanu et al. [12] do not use d_V, and instead employ a multi-dimensional function d_E for dissimilarity of appearance, angle, and distance. The work of Torresani et al. [13] can be viewed as adopting 2-dimensional d_V and d_E functions for measuring appearance dissimilarity, geometric compatibility, and occlusion likelihood. In [14] a method to learn the real numbers for the insertion $d_V(\epsilon \rightarrow v)$ and deletion $d_V(v \rightarrow \epsilon)$ costs on nodes and edges is proposed. An extension to substitution costs is presented in [15]. While the optimization methods for learning these functions are different, all of them are essentially aimed at learning common weights for all the edge and node dissimilarity functions in a matching context. The discriminative weight formulation Eq. 3 is more general in the sense that it can assign different parameters for individual nodes and edges. In [7], the discriminative weight formulation is also employed. The learning problem is turned into a regression problem and a structured support vector machine (SSVM) is used to minimize it.

Classification level. Learning graph matching in a classification context is more challenging since the ground truth is given at the class level and not at the node/edge level. In [8], a grid search on a validation set is used to determine the values of the parameters $\beta^n_{del} = \beta^n_{ins}$, which corresponds to the cost of a node deletion or insertion, and $\beta^e_{del} = \beta^e_{ins}$, which corresponds to the costs of an edge deletion or insertion. Neuhauss et al. [16] address the issue of learning dissimilarity functions for numerically labeled graphs from a corpus of sample graphs. A system of self-organizing maps (SOMs) that represent the distance measuring spaces of node and edge labels was proposed. The learning process is based on the concept of self-organization. It adapts the edit costs in such a way that the similarity of graphs from the same class is increased, whereas the similarity of graphs from different classes decreases. Two limitations can be put forward (i) attributes must be numeric vectors and (ii) the method aimed at

learning common weights for all the edges and nodes ($\beta_{del}, \beta_{ins}, \beta_{sub}$). From the same authors, in [17], the graph matching process is formulated in a stochastic context and perform a maximum likelihood parameter estimation of the distribution of matching operations. The underlying distortion model is learned using an Expectation Maximization algorithm. The matching costs are adapted so as to decrease the distance between graphs from the same class, leading to compact graph clusters.

Adapting methods that operate at the matching level is not trivial since node correspondences must be inferred from the class label. The neural methods proposed in [16] works at the classification level but it is limited to vector attributes and common weights shared to all nodes and edges. The former limitation is leveraged in [17] thanks to a probabilistic framework but the Expectation Maximization algorithm is not robust as the neural-based minimizer. In this paper we propose to merge both ideas, a neural-based algorithm and the discriminative weight formulation to learn graph matching dissimilarity functions in a classification context.

4 Proposal: a Graph-based perceptron

The perceptron is an algorithm for learning a binary classifier $\mathcal{C} = \{0, 1\}$. In the context of neural networks, a perceptron is an artificial neuron using the Heaviside step function as the activation function. A global picture of the graph-based perceptron is depicted in Figure 1. The conventional perceptron is adapted to graphs thanks to three main features : a) The learning rule to update the weight vector β. b) The graph matching algorithm to find y^*. c) The graph model G^m

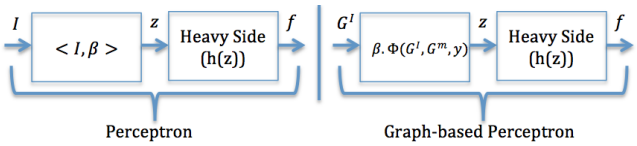

Fig. 1. Overview of the perceptron and our proposal a modified perceptron for graph

Learning rule. The learning rule aims at modifying β. The weights should be updated in cases of wrong classifications. The correction must take into account the amount and the sign of the committed error.

$$\text{Learning rule: } \beta(t+1) = \beta(t) + \alpha(c_i - c_k)\Phi(G^i, G^m, y) \tag{4}$$

To show the time-dependence of β, we use $\beta_i(t)$ as the weight at time t. The parameter α is the learning rate, where $0 < \alpha \leq 1$. $(c_i - c_k)$ is the error function. This error is positive if $(c_i > c_k)$ or negative if $(c_i < c_k)$. The learning rule is the

steepest gradient descent. It tries to reduce the error in the direction of the error descending along the gradient. If we consider the $\Phi(G^i, G^m, y)$ entries associated with weight β respectively.

Graph matching solver. Many efficient approximate algorithms have been proposed to solve the graph matching problem defined in Definition 1.In [8], Riesen et al. have reformulated the Quadratic Assignment Problem of Definition 1 to a Linear Sum Assignment Problem (LSAP). Nodes of both graphs are involved in the assignment problem. A cost matrix is computed to enumerate pair-wise node distances. The LSAP can be solved in polynomial time $O(n^3)$ which makes this approach very fast.

Graph model. The graph matching is computed between an input graph G^i and a model graph G^m. The choice of a model graph among a set of graphs is of first interest. The model graph should represent the diversity of attributes and topologies which can be found in the graph set TrS. The graph model selection rule is defined as follows : $G^m = arg \max_{G \in TrS} |G|$ .With $|G| = |V| + |E|$. Accordingly G^m is the largest graph of the set. In such a way that G^m may gather a large diversity of attributes along with different structures. Other definition could hold such as the median graph definition but this is beyond the scope of the paper.

Learning algorithm. We design the learning algorithm of the graph-based perceptron. Algorithm 1 is an $O(\#iter.|TrS|)$ deterministic algorithm. Solving the parametrized graph matching problem is indicated in Line 8. Line 9 is the classification step while lines 10 to 12 are the application of the learning rule defined Eq. 4 when the classification is wrong. Finally, it is worth mentioning that classifying an entire test set (TeS) is done by only $|TeS|$ call to the graph matching algorithm involved in the function Φ. This low complexity makes it a fast classifier.

Data: $TrS = \{< G^i, c_i >\}_{i=1}^{M}$
Data: $\#iter$ is the maximum number of iterations
Data: α learning rate
Result: Learned β. A weight vector
1 $\beta \leftarrow 0$ and $t \leftarrow 0$
2 **while** $error > 0$ *and* $iter < \#iter$ **do**
3 error $\leftarrow 0$ and iter $\leftarrow 0$
4 **for** $G_i \in TrS$ **do**
5 $y^* \leftarrow \underset{y}{\text{argmin}} \quad \beta(t) \cdot \Phi(G^i, G^m, y)$ `// Solve problem in Definition 1`
6 $c_i \leftarrow heavyside(\beta(t) \cdot \Phi(G^i, G^m, y^*))$
7 $c_k \leftarrow getLabel(G^i)$
8 **if** $c_i - c_k$ *!= 0* **then**
9 $\beta(t+1) \leftarrow \beta(t) + \alpha(c_i - c_k)\Phi(G^i, G^m, y^*)$
10 error$\leftarrow$ error +1
11 **end**
12 t $\leftarrow$ t +1
13 **end**
14 error $\leftarrow$ error$/|TrS|$
15 iter $\leftarrow$ iter +1
16 **end**

Algorithm 1: Learning graph-based perceptron scheme

5 Experiments

Two graph databases LETTER HIGH (LETTER for short) and GREC were chosen from from the IAM repository [18]. Each database consists on a set of different graph instances divided in different classes where each class is composed of a training set and a test set. Datasets are described in Table 1. Matching functions d_v and d_e were taken from [8].

Database	size (TrS,TeS)	♯classes	node labels	edge labels	$\overline{\lvert V\rvert}$	$\overline{\lvert E\rvert}$	max $\lvert V\rvert$	max $\lvert E\rvert$	balanced
LETTER (high)	(750,750)	15	x,y	none	4.7	4.5	9	9	Y
GREC	(286,528)	22	x,y	Line types	11.5	12.2	25	30	Y

Table 1. Summary of graph data set characteristics.

A commonly used approach in pattern classification is based on nearest-neighbor classification. That is, an unknown object is assigned the class or identity of its closest known element, or nearest neighbor (1-NN). Two versions were involved in the tests. A 1-NN with no weights ($\beta = 1$) called NW-1-NN and a tuned 1-NN (T-1-NN) where $\beta^e_{del} = \beta^e_{ins}$ and $\beta^n_{del} = \beta^n_{ins}$ values are taken from [8]. To assess the performance of our learning scheme and our new classifier, two experiments were performed. First, the impact of the learning rate α was studied on 2 classes of the GREC dataset (class 0 and 1) and second, pair-wise binary classifications were carried out among all classes of two datasets GREC and LETTER. To sum-up all theses experiments, the mean classification rate ($\overline{\eta}$) during the training and the test phases are reported along with the standard deviation ($std(\eta)$). The time in milliseconds to classify all instances is also considered. In Figure 2, the impact of the learning rate is depicted. A high learning rate leads to fast convergence with many oscillations around 80% of classification rate, while at the opposite a low learning rate implies a slow but stable convergence. A trade-off can be found with an intermediate value ($\alpha = 0.01$). This value was chosen to perform the rest of the experiments with a number of iterations set to 100. To continue on the learning capability of the Algorithm 1, in Table 2, the classification rate obtained during the learning phase are tabulated (column η_{TrS}). A first comment leads to say that with the highest classification rate GREC was easier to learn than LETTER. A second comment is the clear capability of learning of our method. In fact, a dummy classifier with "bad" weights $\beta = 1$ would produce a random classification and a classification rate of 0.5. Finally, binary classifications results on $(22-1)^2 = 441$ and 196 pairs of classes for GREC and LETTER, respectively, are synthesized in Table 2. First, on the speed side, our classifier is by far the fastest with a speed gain of about 350 (350 times faster). In fact, time complexity of our graph-based perceptron is linear in function of the test set size ($|TeS|$) whereas the complexity of the 1-NN grows quadratically in function of $|TrS|.|TeS|$. On the classification rate side, on GREC, our proposal clearly outperformed the NW-1-NN classifier with no-weights while obtaining similar results than the T-1-NN classifier. On LETTER,

the situation is different, the NW-1-NN classifier provides astonished results as good as the T-1-NN. We can conclude that dissimilarity functions d_V and d_E are well suited on their own for the problem and that performances come from the good graph prototypes of TrS. With a single model graph our approach does not succeed to capture the whole variability of the problem. However, the 15% loss of accuracy is counter balanced by a large speed-up.

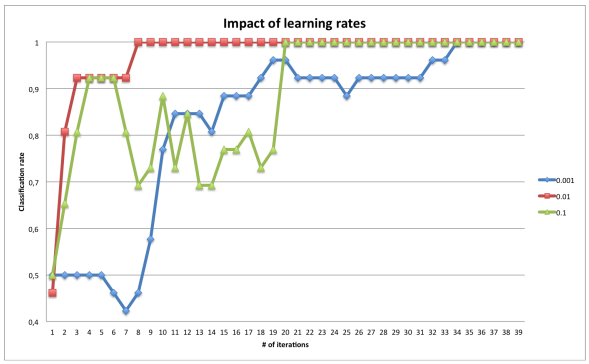

Fig. 2. Learning rate as a function of the number of iterations

	GREC					LETTER				
	$\overline{\eta_{TrS}}$	$\overline{\eta}$	$std(\eta)$	$\overline{time}$	$std(time)$	$\overline{\eta_{TrS}}$	$\overline{\eta}$	$std(\eta)$	$\overline{time}$	$std(time)$
Proposal	0.9733	0.9488	0.1054	**87.31**	24.49	0.8610	0.8262	0.1279	**31.09**	6.42
NW-1-NN ($\beta = 1$)	NA	0.5235	0.0561	1588.83	870.46	NA	**0.9735**	0.0294	1584.15	510.37
T-1-NN ([8])	NA	**0.9992**	0.0096	1789.52	990.08	NA	**0.9735**	0.0295	1573.96	490.51

Table 2. Classification results on GREC and LETTER. The best results are marked in bold style.

6 Conclusion

In this paper, a graph classifier operating in the graph space was presented. A graph-based perceptron was proposed to learn discriminative graph matching in a classification context. Graph matching was parametrized to build a weighted formulation. This weighted formulation is used to define a perceptron classifier. Weights are learned thanks to the gradient descent algorithm. Classification results on two publicly available datasets demonstrated a large speed-up in classification (350 times faster in average) with a loss of accuracy of 4% in average. As the conventional perceptron, the graph-based perceptron will be extended to multi-class problems. Another perceptive is to extend our work to multiple layers and consequently to learn mid-level graph-based representations.

References

1. Kaspar Riesen. *Structural Pattern Recognition with Graph Edit Distance - Approximation Algorithms and Applications*. Advances in Computer Vision and Pattern Recognition. Springer, 2015.
2. Benoit Gaüzère, Luc Brun, and Didier Villemin. Two new graphs kernels in chemoinformatics. *Pattern Recogn. Lett.*, 33(15):2038 – 2047, 2012.
3. Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. Structured representations in a content based image retrieval context. *J. Visual Communication and Image Representation*, 24(8):1252–1268, 2013.
4. Michel Neuhaus and Horst Bunke. *Bridging the Gap Between Graph Edit Distance and Kernel Machines*. World Scientific Publishing Co., Inc., 2007.
5. Volker Roth, Julian Laub, Motoaki Kawanabe, and Joachim M. Buhmann. Optimal cluster preserving embedding of nonmetric proximity data. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(12):1540–1551, 2003.
6. T. S. Caetano, J. J. McAuley, L. Cheng, Q. V. Le, and A. J. Smola. Learning graph matching. *IEEE Trans. on Pattern Anal. Mach. Intell.*, 31(6):1048–1058, 2009.
7. Minsu Cho, Karteek Alahari, and Jean Ponce. Learning graphs to match. In *IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013*, pages 25–32. IEEE Computer Society, 2013.
8. Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vision Comput.*, 27(7):950–959, 2009.
9. Michel Neuhaus, Kaspar Riesen, and Horst Bunke. Fast suboptimal algorithms for the computation of graph edit distance. In *SSPR/SPR*, pages 163–172, 2006.
10. Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A graph matching method and a graph matching distance based on subgraph assignments. *Pattern Recogn. Lett.*, 31(5):394–406, 2010.
11. Francesc Serratosa. Fast computation of bipartite graph matching. *Pattern Recogn. Lett.*, 45:244–250, 2014.
12. Marius Leordeanu, Rahul Sukthankar, and Martial Hebert. Unsupervised learning for graph matching. *International Journal of Computer Vision*, 96(1):28–45, 2012.
13. Lorenzo Torresani, Vladimir Kolmogorov, and Carsten Rother. Feature correspondence via graph matching: Models and global optimization. In *Computer Vision - ECCV*, pages 596–609, 2008.
14. Xavier Cortés and Francesc Serratosa. Learning graph-matching edit-costs based on the optimality of the oracle's node correspondences. *Pattern Recogn. Lett.*, 56:22–29, 2015.
15. Xavier Cortés and Francesc Serratosa. Learning graph matching substitution weights based on the ground truth node correspondence. *IJPRAI*, 30(2), 2016.
16. Michel Neuhaus and Horst Bunke. Self-organizing maps for learning the edit costs in graph matching. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 35(3):503–514, 2005.
17. Michel Neuhaus and Horst Bunke. Automatic learning of cost functions for graph edit distance. *Inf. Sci.*, 177(1):239–247, 2007.
18. Kaspar Riesen and Horst Bunke. Iam graph database repository for graph based pattern recognition and machine learning. In *SSPR*, volume 5342 of *Lecture Notes in Computer Science*, pages 287–297. 2008.