Learning error-correcting graph matching with a multiclass neural network

Maxime Martineau, Romain Raveaux, Donatello Conte, Gilles Venturini

Learning error-correcting graph matching with a multiclass neural network

Maxime Martineau Romain Raveaux Donatello Conte
Gilles Venturini
Laboratoire d'Informatique Fondamentale et Appliquées de Tours (EA 6300),
64 avenue Jean Portalis, Tours, France

April 12, 2018

Abstract

Many tasks in computer vision and pattern recognition are formulated as graph matching problems. Despite the NP-hard nature of such problems, fast and accurate approximations have led to significant progress in a wide range of applications. However, learning graph matching from observed data, remains a challenging issue. In practice, the node correspondences ground truth is rarely available. This paper presents an effective scheme for optimizing the graph matching problem in a classification context. For this, we propose a representation that is based on a parametrized model graph, and optimize the associated parameters. The objective of the optimization problem is to increase the classification rate. Experimental results on seven public datasets demonstrate the effectiveness (in terms of accuracy and speed) of our approach compared to four reference graph classifiers.

1 Introduction

Graphs are frequently used in various fields of computer science, since they constitute a universal modeling tool that enables the description of structured data. The handled objects and their relations are described in a single and human-readable formalism. Hence, tools for graph classification and clustering are required in many applications such as pattern recognition (Riesen, 2015), chemical component analysis (Gaüzere et al., 2012) and structured data retrieval (Raveaux et al., 2013). Various approaches have been proposed during the last decade for tackling the problem of graph classification. Graph classification is the problem of identifying to which of a set of categories, a new graph belongs on the basis of a training set. A graph classifier is a function that maps its input $G \in \mathbb{G}$ to an output value $f(G) \in \mathcal{C}$. $\mathbb{G}$ is the graph space and $\mathcal{C}$ is a set of categories or classes. Graph classifiers can be categorized into two categories based on whether the classifier operates in a graph space or in a vector space. Methods that operate in a vector space can also be split into two categories based on whether they rely on explicit graph embedding ($\phi : \mathbb{G} \rightarrow \mathbb{R}^n$) or implicit ($k :< \mathbb{G}, \mathbb{G} >\rightarrow \mathbb{R}$) graph embedding by means of graph kernels (Riesen and Bunke, 2010; Lozano and Escolano, 2013). Even if such approaches have proven to achieve high performance, they have two main drawbacks: One is a lack of interpretability. It is very difficult to return to the graph space from the kernel space. This problem is known as the "pre-image" problem (Bakır et al., 2004) and is still open. The second drawback is that graph kernel techniques rely on the choice of *a priori* structures defining a graph similarity (i.e.: walks, paths, cliques, etc.).

Graph space ($d : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{R}$). To classify unknown objects using the k-Nearest Neighbor (k-NN) 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 evaluation of the "best" matching between them. Since graph isomorphism rarely occurs in pattern analysis applications, the matching process must be error-tolerant, i.e., it must tolerate differences in the topology and/or its labeling. For instance, in the Graph Edit Distance (GED) problem (Riesen, 2015), a set of edit operations that are applied on nodes and edges (substitution, deletion and insertion) are introduced. Each edit operation is characterized by a cost and the dissimilarity measure is the total cost of the least expensive set of operations that transform one graph into another. In (Riesen, 2015; Bougleux et al., 2017), the

GED problem is shown to be equivalent to a Quadratic Assignment Problem (QAP). Since error-tolerant graph matching problems are NP-hard, most research has long focused on developing accurate and efficient approximate algorithms. In (Bougleux et al., 2017), with this quadratic formulation, two well known graph matching methods ,namely, the Integer Projected Fixed Point method (Leordeanu et al., 2009) and the Graduated Non Convexity and Concavity Procedure (Liu and Qiao, 2014), are applied to GED. In (Leordeanu et al., 2009), the heuristic improves an initial solution by solving a linear assignment problem (LSAP) and a relaxed QAP in which binary constraints are relaxed to the continuous domain. The algorithm iterates through gradient descent by comparing the relaxed QAP and the LSAP solutions via a line search. In (Liu and Qiao, 2014), a path following algorithm aims at approximating the solution of a QAP by considering a convex-concave relaxation through a modified quadratic function.

Positioning the paper. Our paper can be located in the literature by two entries: the complexity and the *a priori* knowledge integrated in the model. First, on the complexity side, conventional kernel-based and k-NN methods cannot be applied directly on large data sets as they are of quadratic complexity. The number of calls to the dissimilarity measure grows quadratically as a function of the number of graphs in the learning and test datasets. Second, graph kernel techniques rely on the choice of *a priori* structures defining a graph similarity (i.e.: walks, paths, cliques, etc.). Also, recent studies have revealed that hand-crafted edit costs or matching functions, which are typically used in graph matching, are insufficient for capturing the inherent structure that underlies the problem at hand. As a consequence, a better optimization of the graph matching problem does not guarantee better correspondence accuracy with respect to a user-defined ground-truth (Caetano et al., 2009; Cho et al., 2013) or a better classification rate. To tackle this issue, we introduce a set of parameters into the graph matching problem. Then, we use a learning scheme to obtain the best values for these parameters, according to the problem at hand. Such a learned matching function better model the inherent structure of the classification problem. This paper deals with paradigms that operate directly on the graph space and can thus capture more structural distortions than embedding techniques, as well as giving more interpretable solutions (because expressed directly in the graph space).

In Section 2, the definitions are established and the parametrized graph matching problem is presented. In Section 3, the state of the art is detailed and weaknesses are identified. In Section 4, the methodology, complexity and algorithms of our proposed approach are presented. Section 5, highlights an experimental protocol along with its results. Finally, in Section 6, conclusions and perspectives are discussed.

2 Problem statement

In this section, the problem of learning discriminative graph matching is formally defined. An attributed graph is considered as a triple $(V,\ E,\ L)$ where V is a set of vertices. E is a set of edges such as $E \subseteq V \times V$ and L is a set of attributes of the nodes and edges. For clarity, we abuse the set notation such that L_i is a label that is associated with vertex v_i and L_{ij} is a label that is associated with an edge (v_i, v_j). Graphs are assumed to be simple (no loops or multiple edges).

2.1 Graph matching problem

We define the error-correcting graph matching problem to compute the GED of two graphs. 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 $N^2 = \{1, \cdots, m\}$. To apply removal or insertion operations on nodes, node sets are augmented by dummy elements. The removal 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 that is associated with v_i. As a consequence, the set N^2 is increased by n 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 m dummy elements ϵ^2 to form $V^1 = N^1 \cup \epsilon^1$. Note that V^1 and V^2 have the same cardinality : $n1 = n2 = n + m$. Each element of the two graphs can be edited only once. A graph matching solution is defined as a subset of possible correspondences $y \subset V^1 \times V^2$, which are represented by a binary assignment matrix $Y \in \{0,1\}^{n_1 \times n_2}$, where n_1 and n_2 denote the sizes of V^1 and V^2, respectively. If $v_i^1 \in V^1$ matches with $v_a^2 \in V^2$, then $Y_{i,a} = 1$, or $Y_{i,a} = 0$ otherwise. We denote by $y \in \{0,1\}^{n_1 n_2}$, a column-wise vectorized replica of Y. With this notation, the error-correcting graph matching problem can be expressed as the problem of finding the assignment vector y^* that minimizes a score function $d(G^1, G^2, y)$ as follows:

Problem 1 *Graph matching formulation*

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

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

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

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

where constraints 1c and 1d indicate that each node of a graph must be matched with only one node of the other graph.
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 $e_{ij}^1 \in V^1 \times V^1$ and $e_{ab}^2 \in V^2 \times V^2$. Dissimilarity functions are usually represented by a symmetric dissimilarity matrix D, in which 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 objective 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 that are relevant to the assignment. Problem 1 models the error-correcting graph matching to compute the GED. In the literature, this problem is also called as the GED problem. The two names are equivalent except that the first name explicitly specifies the matching. Problem 1 is referred to as an integer quadratic programming problem. More precisely, it is the quadratic assignment problem, which is known to be NP-hard.

2.2 Parametrized graph matching

In the context of the objective function that is defined in Eq. 2, an interesting question arises: What can be learned to improve graph matching. To address this issue, 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 vector form as: $\Phi(G^1, G^2, y) = [\cdots, d_V(L_{\pi(a)}^1, L_a^2), \cdots, d_E(L_{\pi(a)\pi(b)}^1, L_{ab}^2), \cdots]$. By introducing a real-valued vector β to weight all elements of this feature map, we obtain a discriminative objective function:

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

$$= \cdots + \beta_a \cdot d_V(L_{\pi(a)}^1, L_a^2) + \\ + \beta_{ab} \cdot d_E(L_{\pi(a)\pi(b)}^1, L_{ab}^2) + \cdots \tag{3b}$$

where β is a weight vector that encodes the importance of node and edge dissimilarity values. In the case of uniform weights, i.e. $\beta = \mathbf{1}$, all elements of vector β are 1, and Eq. 3a is reduced to the conventional graph matching score function of Eq. 2: $d(G^1, G^2, y) = d(G^1, G^2, y, \mathbf{1})$. An example of the parametrized objective function is given in Figure 1. The discriminative weight formulation is general in the sense that it can assign different parameters to individual nodes and edges. However, it does not learn a graph model that underlies the feature map, and requires a reference graph G^2 at query time, whose attributes cannot be modified in the learning phase. The new discrete optimization problem can be rewritten from Problem 1 as follows:

Problem 2 *Parametrized graph matching problem*

$$y^* = \underset{y}{\operatorname{argmin}} \quad d(G^1, G^2, y, \beta) \tag{4a}$$

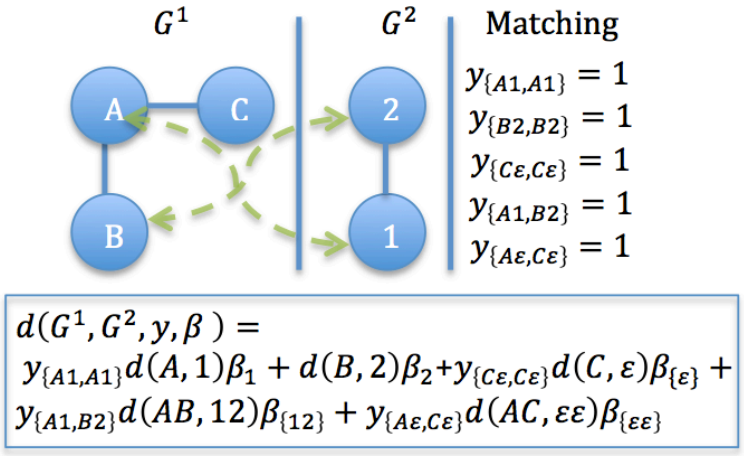


Figure 1: Illustration of the parametrized score function computation.

$$\textit{subject to} \quad y \in \{0,1\}^{n_1 n_2} \tag{4b}$$

$$\sum_{i=1}^{n_1} y_{i,a} = 1 \quad \forall a \in [1, \cdots, n_2] \tag{4c}$$

$$\sum_{a=1}^{n_2} y_{i,a} = 1 \quad \forall i \in [1, \cdots, n_1] \tag{4d}$$

2.3 Graph classification problem

For clarity, the rest of the definition is focused on a 2-class problem. However 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, and $\beta \cdot x$ is the dot product $\sum_{i=1}^{q} \beta_i x_i$, in which 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, the weighted combination of inputs must produce a positive value that is greater than $|b|$ in order to push the classifier over the 0 threshold. For clarity, b is often included in the weight vector β with $b = \beta_0 \times 1$. To extend this paradigm to graphs, 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, out of the two possible classes. The learning of a graph classifier consists of inducing from TrS a mapping function $f : \mathbb{G} \to \mathcal{C}$ that assigns a class to an unknown graph:

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

Let $\Delta(TrS, f)$ be a function that computes the error rate obtained by a classifier f. We represent the error for the p^{th} training sample by $error_p = 1 - \delta(c_p, f(G_p))$, where c_p is the target value, $f(G^p)$ is the value that is produced by the classifier and $\delta(\cdot,\cdot)$ 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 on the training set.

To address the problem of learning graph matching, we start with the discriminative weight formulation of Eq. 3a. We learn the weights β from labelled examples from TrS minimizing the function Δ. The objective function is the error rate function with extra weights β: $\min_{\beta} \Delta(TrS, f, \beta)$.

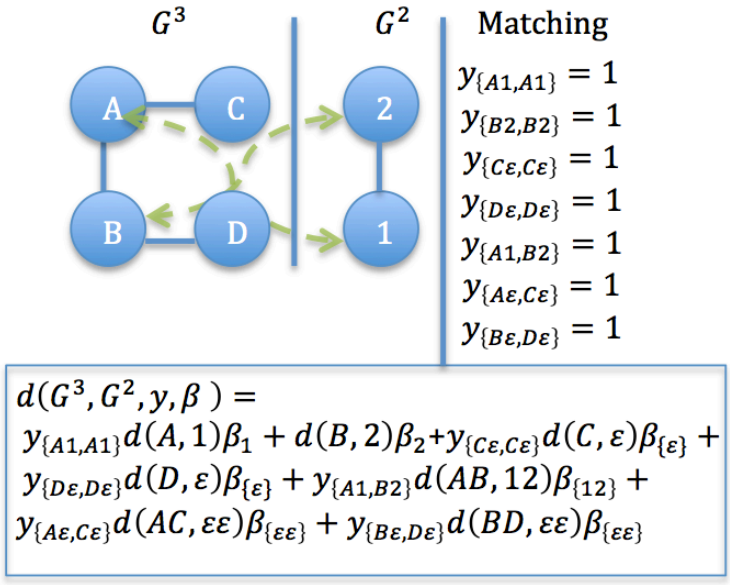


Figure 2: Parametrized matching function where G^1 has 7 graph components (4 nodes and 3 edges).

2.4 Fixed parametrized graph matching for machine learning

To perform conventional machine learning techniques such as Support Vector Machines (SVM) or Deep Neural Networks (DNN) on a data set, a fixed feature vector size is often mandatory. To obtain a fixed size, in Eq.3b, the sizes of the vectors β and $\Phi(G, G^m, y)$ must only depend on the size of G^m and not on the query graph $G = (N, E, L)$.

The graph elements of G and the components of graph $G^m(N^m, E^m)$ are aligned into a vectorial form by the function Φ. $\Phi(G, G^m, y)$ is a vector $\in \mathbb{R}^{|N^m|+|E^m|+2}$. Two extra components are added to accumulate node and edge deletion costs. For instance, it can occur that a node or an edge of G is mapped to a dummy element ϵ (node deletion or edge deletion). In such a case, the node deletions are mapped in the penultimate component of the vector $\Phi(\cdot)$. The edge deletions are mapped in the last component of the vector $\Phi(\cdot)$.

β is a vector $\in \mathbb{R}^{|N^m|+|E^m|+2}$. Individual weights are associated with each component of G^m. Two extra components are added to parametrize the node and edge deletion costs, respectively. A shared weight is associated with all node deletion costs. The same strategy is applied for edge deletions.

The value of the objective function remains the same. Substitutions and insertions have individual weights. Deletions share a mutual parameter. In Figure 1, G^1 holds 5 graph components and in Figure 2, G^1 holds 7 graph components. However, in both figures $\Phi(\cdot)$ and β are vectors of size 5 that only depend on model graph G^2.

3 State of the art

The literature on learning similarity/dissimilarity matching functions can be roughly categorized into two parts according to whether the objective is to minimize an error rate on the number of correctly 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 deduced correspondence. Caetano et al. (Caetano et al., 2009) use a 60-dimensional node similarity function for appearance similarity and a simple binary edge similarity for edges. Leordeanu et al. (Leordeanu et al., 2012) 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. (Torresani et al., 2008) can be viewed as adopting 2-dimensional d_V and d_E functions for measuring appearance dissimilarity, geometric compatibility, and occlusion likelihood. In (Cortés and Serratosa, 2015) a method for learning 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 (Cortés and Serratosa, 2016). While the optimization methods for learning these functions are different, all of them are essentially aimed at learning common weights for all the edges' and nodes' dissimilarity functions in a matching context. The discriminative weight formulation in Eq. 3a is more general in the sense that it can assign different parameters for individual nodes and edges. In (Cho et al., 2013), 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. In (Riesen and Ferrer, 2016), the main contribution is the prediction of whether two nodes match or not. The node assignment is represented by a vector of 24 features. These numerical features are extracted from the node-to-node cost matrix $\mathbf{C}$ which was used for the original matching process. Then, using the assignments derived from exact graph edit distance computation as the ground truth, each computed node assignment is labeled correct or incorrect. This set of labeled assignments is used to train an SVM endowed with a Gaussian kernel to classify the assignments computed by the approximation as correct or incorrect. A major drawback is the optimal solution requirements for performing the learning.

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 (Riesen and Bunke, 2009), a grid search on a validation set is used to determine the values of the parameters $K_n = \beta^n_{del} = \beta^n_{ins}$, which corresponds to the cost of a node deletion or insertion, and $K_e = \beta^e_{del} = \beta^e_{ins}$, which corresponds to the cost of an edge deletion or insertion. The main drawbacks of the grid search is the empirical definitions of intervals and bounds of the grid, which require expertise on the problem. The speed can also drastically decrease as the grid becomes larger. The method aimed at learning common weights for all the edges and nodes and only distinguishes weights by type of operations: deletion, insertion or substitution ($\beta_{del}, \beta_{ins}, \beta_{sub}$). Neuhaus et al. (Neuhaus and Bunke, 2005) 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 dissimilarity spaces of node and edge labels was proposed. The learning process adapts the edit costs in such a way that the similarity of graphs from the same class is increased. The matching are computed only once before learning the costs and each edit operation is considered independently from the matching it belongs. From the same authors, in (Neuhaus and Bunke, 2007), the graph matching process is formulated in a stochastic context. A maximum likelihood parameter estimation of the distribution of matching operations is performed. The underlying distortion model is a mixture of multivariate Gaussians. The model is learned using an Expectation Maximization algorithm. The matching costs are adapted to decrease the distances between graphs from the same class, thereby leading to compact graph clusters. The method requires the summation over all possible edit paths between two graphs which is not tractable in practice. Two limitations can be identified: (i) attributes must be numeric vectors of a fixed dimension. The methods focus on organizing the feature space rather than learning graph matching and (ii) the methods are not discriminative, they model graphs that belong to the same class. A clear picture of the state of the art is provided in Table 1. Adapting methods that operate at the matching level is not trivial since node correspondences must be inferred from the class labels. The neural methods that are proposed in (Neuhaus and Bunke, 2005) perform well at the classification level but are limited to vector attributes and each edit operation is considered independently from the matching it belongs. The former limitation is leveraged in (Neuhaus and Bunke, 2007) thanks to a probabilistic framework. However, the method is not discriminative and models graphs that belong to the same class. In this paper, the contribution is to match graphs and to learn edge/node dissimilarity functions at the same time as a single minimization problem. A neural-based algorithm and the discriminative weight formulation aim at learning graph matching dissimilarity functions in a discriminative way.

The first development of this idea was in paper (Raveaux et al., 2017). In this paper, we go further in three aspects: (i) Theoretically, we extend our method to multiclass problems. (ii) Then, we consider a new family of model graphs based on median graphs. (iii) Finally, four datasets are added to the experimental study. The experimental phase shows that the newly proposed approach achieves various improvements compared to the prior approach published in (Raveaux et al., 2017)

Ref	Ground-Truth Level	Shared parameters	Graph type	Optimization method	Learning space	Discriminative or generative	Matching and Learning
Neuhaus and Bunke (2005)	Class	No	Numeric	SOM	Vector	Generative	Independently
Neuhaus and Bunke (2007)	Class	No	Numeric	EM	Vector	Generative	Independently
Riesen and Bunke (2009)	Class	Yes	Any	Grid Search	Distance	Discriminative	Simultaneously
Riesen and Ferrer (2016)	Matching	No	Any	SVM	Distance	NA	Independently
Cortés and Serratosa (2015, 2016)	Matching	Yes	Any	Quadratic programming	Distance	NA	Independently
Caetano et al. (2009)	Matching	No	Numeric	Bundle	Vector	NA	Simultaneously
(Torresani et al., 2008)	Matching	No	Any	Bundle	Distance	NA	Simultaneously
Cho et al. (2013)	Matching	No	Any	SSVM	Distance	NA	Simultaneously
This paper	Class	No	Any	Neural network	Distance	Discriminative	Simultaneously

Table 1: Papers published about learning graph matching

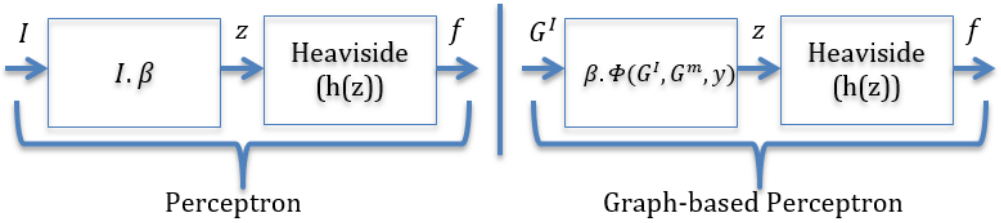


Figure 3: Overview of the perceptron and our proposal of a modified perceptron for graph

4 Proposal: a multiclass graph-based perceptron

In this section, the choice of the perceptron algorithm is explained. Then, on a 2-class problem, the key concepts of our graph-based perceptron are presented along with the changes from the original perceptron paradigm (Rosenblatt, 1957). Finally, an extension to multiclass problems is detailed.

4.1 The perceptron choice

The choice of the perceptron may seem dated compared to recent deep neural network architectures (He et al., 2015). However, four key points motivated this choice. The perceptron is deterministic, simple, capable of online learning and stackable. (i) Simple: The perceptron is a simple algorithm for binary classification in which the weights are adjusted in the direction of each misclassified example. (ii) Deterministic: Given a particular input, the algorithm will always produce the same output. In the graph domain, the parameter space is huge. Thus, having a reproducible process is appreciable. (iii) Online learning ability: The perceptron is capable of online learning (learning from samples one at a time). This is useful for larger datasets since there is no need to address the entire datasets in memory. (iv) Stackable: The simple perceptron neuron can be stacked to create a multilayer perceptron. The multilayer perceptron is a renown algorithm which is still in use in the latest deep learning architectures (He et al., 2015) and is especially involved in the dense layers.

4.2 2-class graph-based perceptron

The perceptron is an algorithm (Rosenblatt, 1957) 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 3. The conventional perceptron is adapted to graphs thanks to three main features : a) the learning rule for updating the weight vector β, b) the graph matching algorithm for finding y^*, and c) the graph model G^m.

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.

$$\text{Learning rule: } \beta(t+1) = \beta(t) + \alpha(c_k - c)\Phi(G, G^m, y^*) \tag{5}$$

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 $0 < \alpha \leq 1$. $(c_k - c)$ is the error function. This error is positive if $(c_k > c)$ and negative if $(c_k < c)$. The learning rule is the steepest gradient descent. It attempts to reduce the error in the direction of the error descending along the gradient when considering the $\Phi(G, G^m, y^*)$ entries associated with weight β.

Graph matching solver. A graph matching algorithm is utilized in the computation of the function f as follows:

1. $y^* \leftarrow$ Solve Problem 1 with $d(G, G_m, y, \beta)$.
2. $f \leftarrow heaviside(\beta(t) \cdot \Phi^T(G, G^m, y^*))$

Many efficient approximate algorithms have been proposed to solve the graph matching problem defined in Problem 1. In (Riesen and Bunke, 2009; Bougleux et al., 2017), Riesen et al. and Bougleux et al. have reformulated the Quadratic Assignment Problem of Problem 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 which makes this approach very fast. To reduce the graph matching problem to a LSAP, local rather than global relationships are considered.

Graph model. The graph matching is computed between an input graph G^i and a graph model G^m. The choice of a graph model among a set of graphs is of primary interest. The graph model should represent the diversity of attributes and topologies that can be found in the graph set TrS. The graph model selection rules can be split into two categories according to whether the choice depends on a graph dissimilarity measure or not. From this perspective, two ways of choosing the graph model are proposed. First, the graph model is defined as follows : $G^m = \arg\max_{G \in TrS} |G|$. With $|G| = |V| + |E|$. Accordingly, G^m_L is the largest graph of the set. G^m may gather a large diversity of attributes along with different structures. Second, the median graph is defined as $G^m = \arg\min_{G \in TrS} \sum_{G^i \in TrS} d(G^i, G, y^*_i, \beta)$. However this definition relies on predefined parameters β. Now, let us design the learning algorithm of the graph-based perceptron. Algorithm 1 is a deterministic algorithm. $\#iter$ is the maximum number of iterations or also called *epochs* in the literature. The parametrized graph matching problem is solved in Line 5. Line 6 is the classification step while lines 8 to 11 apply the learning rule defined in Eq. 5 when the classification is wrong.

Algorithm 1: The learning graph-based perceptron scheme

```
   Data: TrS = {(G, c)} and G^m
   Data: #iter is the maximum number of iterations
   Data: α learning rate
   Result: Learned β. A weight vector
 1 β ← 1 and t ← 0
 2 while error > 0 and iter< #iter do
 3     error ← 0 and iter ← 0
 4     for G ∈ TrS do
 5         y* ← argmin_y  β(t) · Φ^T(G, G^m, y) // Solve problem 2
 6         c ← heaviside(β(t) · Φ^T(G, G^m, y*))
 7         c_k ← getLabel(G)
 8         if c_k − c != 0 then
 9             β(t + 1) ← β(t) + α(c_k − c)Φ(G, G^m, y*)
10             error← error +1
11         end
12         t ← t +1
13     end
14     error ← error/|TrS|
15     iter ← iter +1
16 end
```

4.3 Multiclass graph-based perceptron

The perceptron provides a natural extension of the multiclass problem. Instead of having only one neuron in the output layer, with binary output, we could have N neurons leading to multiclass classification. A set of functions $f(G, c)$ map each possible input/output pair to a real value that represents the fitness of the pair (G, c). The resulting score is used to choose among many possible outputs: $c^* = \text{argmin}_c f(G, c)$. To extend graph-based perceptron to the multiclass case, two main

actions must be taken: a) the binary activation function must be replaced to gauge the fitness between the input and the class; b) the graph models must be computed for each class rather than the entire training dataset.

Activation function. The Heaviside step function can only produce a binary output. To enable generalization, a rectified linear unit (ReLU) is employed, $\min(0, d(G, G^m, y^*, \beta))$. Well admitted, ReLU leads to faster training algorithms than standard sigmoid activation functions (LeCun et al., 2012).

Graph model computation. The model graphs are specialized by class. They are computed on subsets that correspond to each class rather than the entire training set.

In Algorithm 2, the learning procedure of the multiclass graph-based perceptron is designed. In line 5, the selection of the most promising neuron is achieved by finding the minimum activation value (f_i) among all the neurons. Line 5 can be viewed as the classification step. Lines 8 and 9 are dedicated to weight updating. In the case of an incorrect classification, the misleading neuron (index i) must be pushed away from the expected neuron (index k). In line 8, the expected neuron weights (β_k) are reduced, which likely leads to the minimum activation value (f_k) in the next iteration. In the opposite way, in line 9, the misleading neuron weights (β_i) are increased to prevent the i^{th} neuron to be chosen during the next iteration.

Algorithm 2: Learning graph-based perceptron scheme in a multiclass context

```
   Data: TrS = {(G, c)} and G_m^i   ∀i ∈ {1, ··· , |C|}
   Data: #iter is the maximum number of iterations and α learning rate
   Result: Learned β_i   ∀i ∈ {1, ··· , |C|}. A weight vector for each class.
1  β_i ← 1   ∀i ∈ {1, ··· , |C|} and t ← 0
2  while error > 0 and iter< #iter do
3      error ← 0 and iter ← 0
4      for G ∈ TrS do
5          c_i ← arg min_{o∈{1,···,|C|}} f_o(G, o)
6          c_k ← getLabel(G)
7          if c_k − c_i != 0 then
8              β_i(t + 1) ← β_i(t) + αΦ(G, G_m^i, y*)
9              β_k(t + 1) ← β_k(t) − αΦ(G, G_m^k, y*)
10             error← error +1
11         end
12         t ← t +1
13     end
14     error ← error/|TrS| and iter ← iter +1
15
16 end
```

4.4 Complexity Analysis

In this section, we conduct a complexity analysis for the described algorithms. Algorithm 1 is a deterministic algorithm whose complexity in terms of calls to the matching solver is $O(\#iter \cdot |TrS|)$ where $\#iter$ is the number of iterations. In addition, the entire test set (TeS) is classified by only $|TeS|$ calls to the graph matching algorithm. This linear complexity makes the decision procedure a fast graph classifier. Classical graph classifiers require generally $|TrS| \times |TeS|$ calls to the graph matching solver. Algorithm 2 is also a deterministic algorithm. The algorithm complexity in terms of calls to the matching solver is $O(\#iter \cdot |TrS| \cdot |\mathcal{C}|)$. $|\mathcal{C}|$ is the number of classes. The classification of the entire test set is performed by $|TeS| \times |\mathcal{C}|$ calls to the graph matching algorithm. Finally, the complexity of the graph matching solver used in our approach is $O((n+m)^3)$ with n and m the number of nodes in G^1 and G^2, respectively

4.5 Discussion

In this section, we point out the classification capability of our proposal as well as giving advantages and inconvenient of the method. First, the single layer perceptron architecture is only capable of learning linearly separable patterns. However our proposal can go beyond linearly separable features thanks to the graph space. The perceptron algorithm is operating on the graph set TrS through a change of variable from G to $\Phi(G, G^m, y^*)$ vector of minimal edit costs for each element in the model graph G^m. Then, the weights β are optimized with respect to classification loss through the perceptron algorithm. But β is also implied in finding y^* as $y^* = \underset{y}{\text{argmin}} \quad d(G, G^m, y, \beta)$. It

means modifying β is not only acting on the linear projection of $\Phi(G, G^m, y^*)$ but also on the graph matching operator, which is a non-linear operation. Reinterpreting the minimization problem, it appears that both variables β and y are involved into the minimization of an empirical risk guided by the loss function l:

$$\min_{\beta, y} \sum_{(G^k, c_k) \in TrS} l(G^k, c_k, G^m, y_k, \beta) \tag{6}$$

$$l = \frac{1}{2}\left(c_k - heaviside(\beta \cdot \Phi(G^k, G^m, y_k^*))\right)^2 \tag{7}$$

We now describe the strengths and weaknesses of the proposed research method. The first strength of our approach is its adaptability. The graph matching solutions are adapted according to the considered dataset because learning and matching are performed in concert. Second, our approach is discriminative. The similarity of graphs from the same class is increased, whereas the similarity of graphs from different classes is decreased. Third, According to the complexity analysis in the previous section, our approach is very fast in the classification stage because it requires one graph comparison per class. One drawback of the method is the *a priori* choice of model graph (G^m). However, this meta-parameter could be learned in a specific training phase.

5 Experiments

In this section, experiments are presented. First seven datasets are described along with the six methods compared in this experimental study. Finally, the protocol and the results are detailed.

5.1 Experimental setting

Dataset. Seven graph databases were chosen from the public IAM repository Riesen and Bunke (2008) and from the Tarragona graph repository for learning error-tolerant graph matching (Moreno-García et al., 2016). Each database consists of a set of graph instances that are divided in different classes, where each class is composed of a training set TrS and a test set TeS. The datasets are described in Table 2. These databases are representative of a wide range of learning problems that occur in Computer Vision. Matching functions d_v and d_e were taken from Riesen and Bunke (2010) and (Moreno-García et al., 2016). For the sake of reproducibility, the codes of the cost functions are available at `https://sites.google.com/site/gperceptron` along with a video demonstration.

Table 2: On the left side, summary of graph data set characteristics. On the right side, best parameters according to learning strategies R-1NN and C-1NN taken from Paper I Riesen and Bunke (2010) and from Paper II Moreno-García et al. (2016)

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	R-1NN				C-1NN			
										α	K_n	K_e	From	α	K_n	K_e	
LETTER-HIGH	(750,750)	15	x,y	none	4.7	4.5	9	9	Y	0.9	1.7	0.75	Paper I	0.5	1	1	Pa
LETTER-MED	(750,750)	15	x,y	none	4.7	3.2	9	9	Y	0.7	1.9	0.75	Paper I	0.5	1	1	Pa
LETTER-LOW	(750,750)	15	x,y	none	4.7	3.1	9	9	Y	0.3	0.1	0.25	Paper I	0.5	1	1	Pa
CMU	(71,70)	2	Shape	none	30	154.4	30	158	Y	NA	NA	NA	NA	0.5	1000	1	Pa
GREC	(286,528)	22	x,y	Line types	11.5	12.2	25	30	Y	0.5	90	15	Paper I	NA	NA	*NA*	
Fingerprint	(378,1533)	4	none	angle	5.42	4.42	26	24	N	0.75	0.7	0.5	Paper I	NA	NA	*NA*	
COIL-DEL	(2400,1000)	100	x,y	none	21.5	54.2	77	222	Y	NA	NA	NA	NA	NA	NA	*NA*	

Table 3: Taxonomy of the compared methods

Classifier	Parameter learning	Model graph	Method's Name
1NN	R (Riesen and Bunke, 2009)	M	R-M-1NN
		TrS	R-1NN
	C (Cortés and Serratosa, 2015)	M	C-M-1NN
		TrS	C-1NN
Perceptron	Gradient descent	L	G-L-Perceptron
		M	G-M-Perceptron

Compared methods. A commonly used approach in pattern classification is based on nearest-neighbor classification. That is, an unknown object is assigned the class of its closest known element, or nearest neighbor (1NN). Two versions were utilized in the tests. R-1NN (Riesen and Bunke, 2009) and C-1NN (Cortés and Serratosa, 2015) where the values of $K_e = \beta^e_{del} = \beta^e_{ins}$ and $K_n = \beta^n_{del} = \beta^n_{ins}$ were borrowed from (Riesen and Bunke, 2010) and (Moreno-García et al., 2016), respectively. K_n corresponds to the cost of a node deletion or insertion, and K_e corresponds to the cost of an edge deletion or insertion. The aforementioned methods hold a meta parameter $\alpha \in [0, 1]$ which corresponds to the weighting parameter that controls whether the cost on the nodes or on the edges is more important. In Table 2, the best values of parameters (α, K_n, K_e) are summarized. The learning method in C-1NN is based on the minimization of the distance between the graph matching solver and the human ground-truth correspondences. This approach is a generative approach while R-1NN is a discriminative one. The R-1NN learning scheme aims at finding the best parameters by performing a grid search over the parameter space. The 1NN classifier is known to be slow as the size of the training set grows. For fair competition with our algorithms, two fast classifiers were derived from R-1NN and C-1NN: R-M-1NN and C-M-1NN are 1NN classifiers for which the training is reduced to one median graph per class. The median graphs were computed with a graph distance function and the best learned parameters (see Table 2). This training set reduction may lead to a loss of accuracy but produced a large speed-up.

Our proposal is called G-Perceptron which stands for graph-based perceptron. Two versions have been derived according to whether the graph model is the largest graph or the median graph, G-L-Percetron or G-M-Percetron, respectively. As mentioned in Section 4, predefined β parameters are required to compute the median graphs of the G-M-Perceptron method. A vanilla plain solution was adopted with $\beta = \mathbf{1}$. Let us recall that $\beta = \mathbf{1}$ is the initialization value of our algorithms. A summary of the six compared methods is reported in Table 3. The G-Percetron, C-1NN and R-1NN classifiers were implemented in JAVA 7 and run on a 2.6 GHz computer with 8 GB RAM. Graph matching solvers involved in C-1NN and R-1NN are based on the Linear Sum Assignment Problem (LSAP) which is solved in our implementation by the Hungarian method. This setting makes possible a fair comparison.

Protocol. 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 the LETTER-HIGH dataset and second, classifications were carried out on all datasets. To summarize the results of theses experiments, the classification rates (η) during the training and the test phases are reported along with the time, in **milliseconds**, for classifying all test instances. In our protocol, the retained β parameters for the test phase are those that maximize the classification rate on the training set. Finally, we define two metrics for accuracy and speed evaluation. Let $\mathcal{M}$ be the set of all methods.

$$AccDev^i = 100 * \frac{\max_{j \in \mathcal{M}} \eta^j_{TeS} - \eta^i_{TeS}}{\max_{j \in \mathcal{M}} \eta^j_{TeS}} \qquad TimeRatio^i = \frac{Time^i}{\min_{j \in \mathcal{M}} Time^j}$$

where η^i_{TeS} and $Time^i$ are the classification rate and the processing time for the method i. AccDev is the best when equal to zero. For TimeRatio, the closer to 100 it is, the better is the time performance.

5.2 Results

In Figure 4, the impact of the learning rate is depicted for the median and large graph models. A very high learning rate ($\alpha = 0.1$) leads to poor results. The search space exploration is too fast and saddle points are missed. A high learning rate ($\alpha = 0.01$) leads to unstable results with many oscillations while a low learning rate implies a slow but smooth convergence. A trade-off can be achieved with an intermediate value ($\alpha = 0.001$). This value was the best for the median and large graph models. For the rest of the experiments, $\alpha = 0.001$ was chosen and the number of iterations was set to 300. To continue the analysis of the learning ability of Algorithm 2, in Table 4, the classification rates obtained during the learning phase are tabulated (column η_{TrS}). The learning ability is demonstrated on all data sets. The classification rate is always higher on the training set than on the test set except on the CMU and Fingerprint databases where the number of classes is small. The loss between the training and test recognition rates is 3% on average. This result demonstrates the good generalization ability of our algorithms.

Now, we compare the median and large graph models. In Table 5, classification rate deviations and time ratios are reported.

Table 4: Classification results. The best classification rates are marked in blue while the best processing times are in red. "Best" means an improvement over other methods statistically significant according to a z-test with a confidence level of 70%

	G-L-Perceptron			G-M-Perceptron			R-1NN		C-1NN		R-M-1NN		C-M-1NN	
Database	η_{TrS}	η_{TeS}	Time	η_{TrS}	η_{TeS}	Time	η_{TeS}	Time	η_{TeS}	Time	η_{TeS}	Time	η_{TeS}	Time
LETTER-LOW	0.97	0.95	2060	1	0.98	1436	0.99	69700	0.96	71869	0.97	1341	0.98	1341
LETTER-MED	0.66	0.64	2278	0.90	0.87	1404	0.92	74506	0.93	72150	0.86	1388	0.81	1310
LETTER-HIGH	0.75	0.70	2433	0.86	0.81	1669	0.83	86377	0.84	84911	0.82	1731	0.71	1498
CMU	1	0.99	11420	1	0.99	11029	NA	NA	0.99	427955	NA	NA	0.99	10780
GREC	0.84	0.70	23339	0.84	0.75	14370	0.98	199087	NA	NA	0.96	14726	NA	NA
Fingerprint	0.64	0.68	7768	0.74	0.76	1576	0.58	260816	NA	NA	0.73	2138	NA	NA
Coil-DEL	0.60	0.57	1091777	0.52	0.52	471575	NA	NA	NA	NA	NA	NA	NA	NA

Median vs Large graph models. As expected, the classifier that is based on large graph models is slower than the one based on median graphs. On average the G-M-Perceptron is approximatively two times faster than G-L-Perceptron while providing an accuracy gain of 7%. This gain in accuracy reaches 19% for LETTER-MED/HIGH. This is because the median graph better models the intra-class distribution of a classification problem than the largest graph of a set. The median graph is a better prototype or representative of a class. Now, we compare the 1NN based approaches to the G-M-Perceptron.

G-perceptron vs 1NN. On average, he G-M-Perceptron is 1% less accurate than R-1NN. However, the deviations from this mean are large between datasets. On Fingerprint G-M-Percetron is 24% more accurate than R-1NN whereas in GREC, G-M-Percetron is 24% less accurate than R-1NN. A plausible explanation is the lack of data on GREC dataset, with only 17 graphs by class on average, the number of weights (464) is too large and the learning algorithm fails to converge completely. Compared to the learning scheme of C-1NN, our approach is competitive with a small loss of accuracy of 2% on average on the CMU and LETTER databases. Second, in term of speed, on average, G-M-Perceptron is 66 times faster than 1NN classifiers. The time complexity of our graph-based perceptron is quadratic as a function of the test set size and the number of classes ($|TeS| \cdot |\mathcal{C}|$) whereas the complexity of any 1NN classifier grows quadratically as a function of the test and training set sizes ($|TrS| \cdot |TeS|$) with $|\mathcal{C}| << |TrS|$.

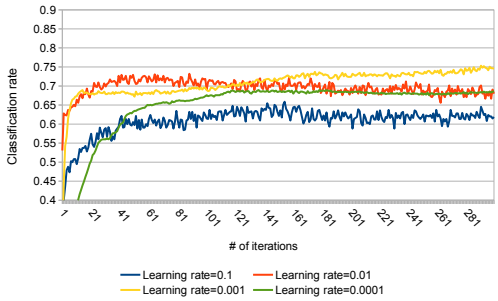
We compare the 1NN classifiers with reduced training sets to G-M-Perceptron. The training sets were reduced to one median graph per class (M-1NN).

G-Perceptron vs M-1NN. As expected, M-1NN classifiers are fast. C-M-1NN is the fastest approach. This is because the graph matching solver is based on a fast LSAP solver. The speed gain is 8% on average compared to G-M-Perceptron. However, G-M-Perceptron is 5% more accurate on average than C-M-1NN. This result is experimental proof of the merit of learning discriminative matchings. G-M-Perceptron and R-M-1NN are based on the same graph matching solver and achieved very similar speed performances on the LETTER datasets. However on the GREC and Fingerprint databases G-M-Perceptron is 20% faster than R-M-1NN. The median graphs are not the same for both approaches and on these difficult databases the median graph sizes do matter.

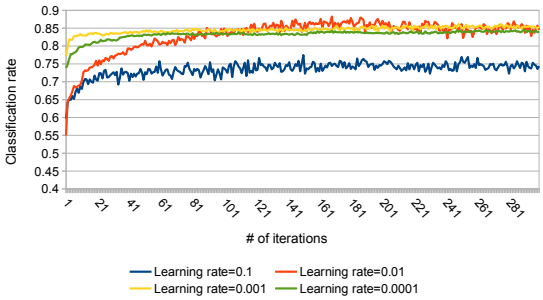
Generally, for the M-1NN classifiers, the median graphs were selected after the learning of the best parameter values (see Table 2) while in the G-M-Perceptron case, the median graphs were chosen from scratch with $\beta = \mathbf{1}$ and the parameter learning scheme had to compensate an eventual bad median graph choice. Our experimental setting is in favor of the M-1NN classifiers. Finally, no reference parameters are available for the Coil-DEL dataset. However, this dataset is interesting for assessing the capability of our method dealing with a large number of classes (100). For instance, on this dataset, a 1NN classifier with $\beta = \mathbf{1}$ obtained a classification rate $\eta_{TeS} = 0.552$ in 13496127 ms. G-M-Perceptron can reach nearly the same accuracy ($\eta_{TeS} = 0.518$) while being 27 times faster. In contrast, a fast M-1NN classifier based on median graphs with $\beta = \mathbf{1}$ achieved only a 1% recognition rate in 286416 ms.

6 Conclusions and Perspectives

In this paper, a graph-based perceptron was proposed for learning discriminative graph matching in a classification context. Our proposal is supported by a formal definition and two deterministic algorithms. Graph matching was parametrized to build a weighted formulation. This weighted formulation was used to define a perceptron classifier, in which each neuron is composed of a graph model and a vector of parameters. Each weight is associated with a graph component of the graph



(a) Large graph model



(b) Median graph model

Figure 4: Letter-HIGH : Impact of the learning rate on the convergence.

Table 5: Classification analysis. The top-2 of deviations are marked in blue while the 2 best time ratios are in red. Accuracy deviations (AccDev) are in % (The lower the better). Time ratio is the best when equal to one (The lower the better).

	G-L-Perceptron		G-M-Perceptron		R-1NN		C-1NN		R-M-1NN		C-M-1NN	
Database	AccDev	TimeRatio	AccDev	TimeRatio	AccDev	TimeRatio	AccDev	TimeRatio	AccDev	TimeRatio	AccDev	TimeRatio
LETTER-LOW	3.91	1.54	0.81	1.07	0.00	51.98	2.70	53.59	1.62	1.00	0.94	1.00
LETTER-MED	31.38	1.74	6.30	1.07	1.58	56.87	0.00	55.08	7.74	1.06	13.04	1.00
LETTER-HIGH	16.64	1.62	3.80	1.11	1.58	57.66	0.00	56.68	3.01	1.16	16.16	1.00
CMU	0.00	1.06	0.00	1.02	NA	NA	0.00	39.70	NA	NA	0.00	1.00
GREC	29.42	1.62	23.85	1.00	0.00	13.85	NA	NA	2.69	1.02	NA	NA
Fingerprint	10.22	4.93	0.00	1.00	23.54	165.49	NA	NA	3.95	1.36	NA	NA
Coil-DEL	0.00	2.32	9.46	1.00	NA	NA	NA	NA	NA	NA	NA	NA

model. Weights are learned using the gradient descent algorithm. Two types of graph models were investigated, the median graph and the large graph of a graph set. Classification results on 7 publicly available datasets demonstrated a large speed-up during the test phase (60 times faster in average) with a loss of accuracy of 6% on average compared to a 1-NN classifier based on an optimized graph distance. The results revealed that the median graph was the best graph prototype for serving as graph model within a neuron.

In future work, the first objective will be to overcome the main drawback of median graphs that require pre-defined weights to be computed. This could be achieved by looping the whole learning scheme. Learned weights could be used to compute better median graphs. Finally, in the near future, we plan to extend our work to multiple layers and consequently to learn mid-level graph-based representations.

References

Bakır, G.H., Zien, A., Tsuda, K., 2004. Learning to find graph pre-images, in: Rasmussen, C.E., Bülthoff, H.H., Schölkopf, B., Giese, M.A. (Eds.), Pattern Recognition, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 253–261.

Bougleux, S., Brun, L., Carletti, V., Foggia, P., Gaüzère, B., Vento, M., 2017. Graph edit distance as a quadratic assignment problem. Pattern Recognition Letters 87, 38–46.

Caetano, T.S., McAuley, J.J., Cheng, L., Le, Q.V., Smola, A.J., 2009. Learning graph matching. IEEE Trans. Pattern Anal. Mach. Intell. 31, 1048–1058.

Cho, M., Alahari, K., Ponce, J., 2013. Learning graphs to match, in: IEEE International Conference on Computer Vision, ICCV 2013, pp. 25–32.

Cortés, X., Serratosa, F., 2015. Learning graph-matching edit-costs based on the optimality of the oracle's node correspondences. Pattern Recogn. Lett. 56, 22–29.

Cortés, X., Serratosa, F., 2016. Learning graph matching substitution weights based on the ground truth node correspondence. IJPRAI 30.

Gaüzere, B., Brun, L., Villemin, D., 2012. Two new graphs kernels in chemoinformatics. Pattern Recogn. Lett. 33, 2038 – 2047.

He, K., Zhang, X., Ren, S., Sun, J., 2015. Deep residual learning for image recognition. CoRR abs/1512.03385. URL: http://arxiv.org/abs/1512.03385.

LeCun, Y., Bottou, L., Orr, G.B., Müller, K., 2012. Efficient backprop, in: Montavon, G., Orr, G.B., Müller, K. (Eds.), Neural Networks: Tricks of the Trade - Second Edition. Springer. volume 7700 of *Lecture Notes in Computer Science*, pp. 9–48.

Leordeanu, M., Hebert , M., Sukthankar, R., 2009. An integer projected fixed point method for graph matching and map inference, in: Proceedings Neural Information Processing Systems, pp. 1114–1122.

Leordeanu, M., Sukthankar, R., Hebert, M., 2012. Unsupervised learning for graph matching. International Journal of Computer Vision 96, 28–45.

Liu, Z., Qiao, H., 2014. GNCCP - graduated nonconvexityand concavity procedure. IEEE Trans. Pattern Anal. Mach. Intell. 36, 1258–1267.

Lozano, M.A., Escolano, F., 2013. Graph matching and clustering using kernel attributes. Neurocomputing 113, 177–194.

Moreno-García, C.F., Cortés, X., Serratosa, F., 2016. A graph repository for learning error-tolerant graph matching, in: Structural, Syntactic, and Statistical Pattern Recognition S+SSPR 2016, pp. 519–529.

Neuhaus, M., Bunke, H., 2005. Self-organizing maps for learning the edit costs in graph matching. IEEE Trans. Systems, Man, and Cybernetics, Part B 35, 503–514.

Neuhaus, M., Bunke, H., 2007. Automatic learning of cost functions for graph edit distance. Inf. Sci. 177, 239–247.

Raveaux, R., Burie, J.C., Ogier, J.M., 2013. Structured representations in a content based image retrieval context. J. Visual Communication and Image Representation 24, 1252–1268.

Raveaux, R., Martineau, M., Conte, D., Venturini, G., 2017. Learning graph matching with a graph-based perceptron in a classification context, in: Graph-Based Representations in Pattern Recognition - 11th IAPR-TC-15 International Workshop, GbRPR 2017, Proceedings, pp. 49–58.

Riesen, K., 2015. Structural Pattern Recognition with Graph Edit Distance - Approximation Algorithms and Applications. Advances in Computer Vision and Pattern Recognition, Springer.

Riesen, K., Bunke, H., 2008. Iam graph database repository for graph based pattern recognition and machine learning, in: SSPR. volume 5342 of *Lecture Notes in Computer Science*, pp. 287–297.

Riesen, K., Bunke, H., 2009. Approximate graph edit distance computation by means of bipartite graph matching. Image Vision Comput. 27, 950–959.

Riesen, K., Bunke, H., 2010. Graph Classification and Clustering Based on Vector Space Embedding. volume 77 of *Series in Machine Perception and Artificial Intelligence*. WorldScientific.

Riesen, K., Ferrer, M., 2016. Predicting the correctness of node assignments in bipartite graph matching. Pattern Recognition Letters 69, 8–14.

Rosenblatt, F., 1957. The perceptron – a perceiving and recognizing automaton. Cornell Aeronautical Laboratory. , Report 85–460–1.

Torresani, L., Kolmogorov, V., Rother, C., 2008. Feature correspondence via graph matching: Models and global optimization, in: Forsyth, D.A., Torr, P.H.S., Zisserman, A. (Eds.), Computer Vision - ECCV, Springer. pp. 596–609.