Noname manuscript No.
(will be inserted by the editor)

Learning the graph edit costs based on a learning model applied to sub-optimal graph matching

Pep Santacruz · Francesc Serratosa

Received: date / Accepted: date

Abstract Attributed graphs are used to represent patterns composed of several parts in pattern recognition. The nature of these patterns can be diverse, from images, to handwritten characters, maps or fingerprints. Graph edit distance has become an important tool in structural pattern recognition since it allows us to measure the dissimilarity of attributed graphs. It is based on transforming one graph into another through some edit operations such as substitution, deletion and insertion of nodes and edges. It has two main constraints: it requires an adequate definition of the costs of these operations and its computation cost is exponential with regard to the number of nodes. In this paper, we first present a general framework to automatically learn these edit costs considering graph edit distance is computed in a sub-optima way. Then, we specify this framework in two different models based on neural networks and probability density functions. An exhaustive practical validation on 14 public databases, which have different features such as the size of the graphs, the number of attributes or the number of graphs per class have been performed. This validation shows that with the learned edit costs, the accuracy is higher than with some manually imposed costs or other costs automatically learned by previous methods.

Pep Santacruz
Universitat Rovira i Virgili. Avinguda dels Pasos Catalans 26, Tarragona, 43005, Catalonia, Spain
E-mail: joseluis.santacruz@urv.cat

Francesc Serratosa
Universitat Rovira i Virgili. Avinguda dels Pasos Catalans 26, Tarragona, 43005, Catalonia, Spain https://orcid.org/0000-0001-6112-5913. Tel: +34 977 55 85 07
E-mail: francesc.serratosa@urv.cat

1 Introduction

Attributed graphs have been of crucial importance in pattern recognition throughout more than four decades, [5,35,36], since they have been used to model several kinds of problems. Interesting reviews of techniques and applications are [8,48,25,18]. If elements in pattern recognition are modelled through attributed graphs, error-tolerant graph-matching algorithms are needed that aim to compute a mapping between nodes of two attributed graphs that minimises some kind of objective function. To that aim, one of the most widely used methods to evaluate an error-correcting graph matching is graph edit distance (GED) [28,46,45,19,31].

GED is defined as the minimum amount of required distortion to transform one graph into another. To this end, a number of distortion or edit functions consisting of deletion, insertion, and substitution of nodes and edges are defined. To quantitatively evaluate the graph transformation, an edit cost is assigned to each edit operation according to the amount of distortion that it introduces in the transformation.

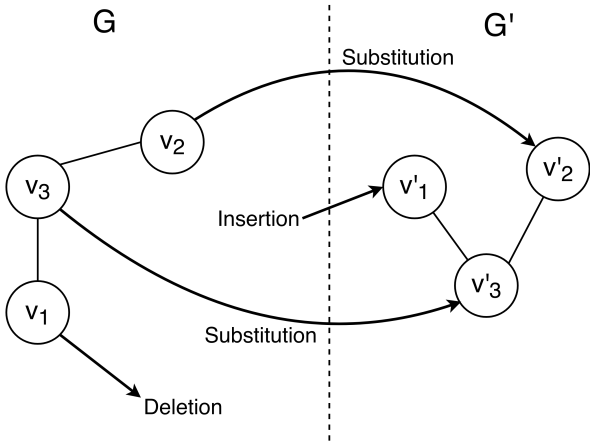


Fig. 1: Example of two graphs to be compared. The GED between G and G' is the cost of substituting two nodes, deleting a node, inserting a node, substituting an edge, deleting an edge and inserting an edge.

For instance, in Figure 1, $GED(G, G')$ is defined such that G is transformed into G' given the transformation highlighted by the arrows. This is done by substituting v_3 by v'_3 and v_2 by v'_2. Moreover, v_1 is deleted and v'_1 is inserted. The distance is the cost of these operations. That is, $SubsNode(v_3, v'_3) + SubsNode(v_2, v'_2) + DelNode(v_1) + InsNode(v'_3) + SubsEdge(v_3, v_2, v'_3, v'_2) + DelEdge(v_3, v_1) + InsEdge(v'_3, v'_1)$. Other transformations generate other costs and so, the distance becomes the minimum of the costs of all transformations. Thus, the structural and semantic dissimilarity of graphs is only correctly re-

flected by the GED if the underlying edit costs are defined appropriately and depending on the application at hand.

The aim of this paper is to learn these edit costs. Figure 2 shows our general scheme. In a first stage, the learning method presented in this paper deduces the edit costs that better fit the application at hand given the learning database. In a second stage, these costs are used the deduce the graph edit distances between graphs in a pattern recognition application. Since computing the GED is exponential with regard to the number of nodes, we present a framework that assumes a sub-optimal computation of it. When the edit costs have been learned or manually set, the sub optimal GED can be computed between pairs of attributed graphs by a graph matching algorithm. Note that it is needed the definition of the approximation of the GED in the learning stage to be the same as the one in the recognition stage.

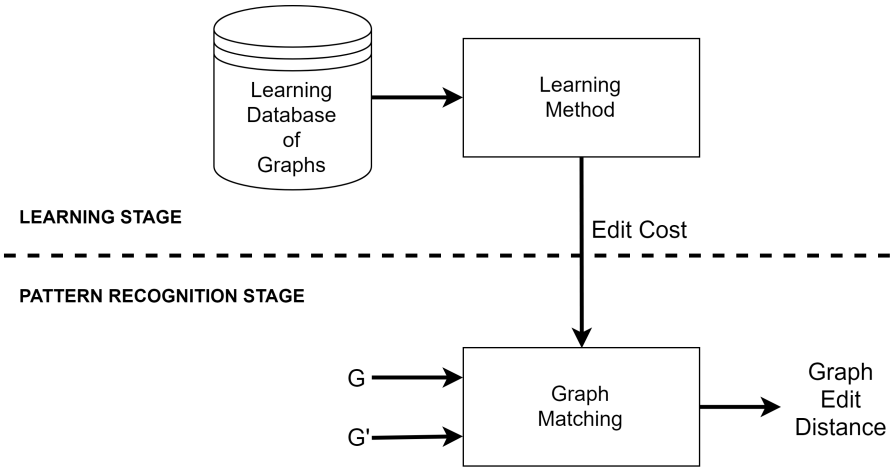


Fig. 2: General graph matching scheme.

Seven methods have been presented to learn the edit costs. The optimisation function (recognition ratio, correspondence accuracy or Dunn index)and the nature of their edit costs (Real number, Neural network, probability distribution or constant) are their main characteristics. Section 2.2 explains the features and differences of these methods.

This paper is structured as follows. In Section 2, we define the attributed graphs and the GED. We also summarise the learning methods applied to learn the edit costs and also the methods that embed graphs into vectors. In Section 3, we explain our general learning framework. In Section 4, we apply this framework to two specific models. In Section 5, we empirically compare our method to the seven methods commented before given fourteen databases. Section 6 concludes the paper.

This paper is based on a preliminary contribution presented in [38]. The current paper has been extended with respect to both the theoretic foundation and the experimental validation. On the one hand, the description of the novel approach as well as the underlying concepts are more detailed as in the preliminary paper. On the other hand, the experimental evaluation as well as the discussion has been substantially extended. In particular, we have increased the number of databases to fourteen and also, the number of compared methods to the seven ones previously commented.

2 Attributed graphs and graph edit distance

Suppose we have a pair of graphs, G and G'. Also suppose the a^{th} node in G is represented as v_a and the i^{th} node in G' is represented as v'_i. Similarly, the edge between the v_a node and the b_b node in G is represented as $e_{a,b}$. And finally, the edge between v'_i node and the v'_j node in G' is represented as $e'_{i,j}$.

The GED between two attributed graphs consists of finding the best combination of edit operations that transform one graph into another [44]. Three operations are considered on the nodes and also on the edges: Substitution, deletion and insertion. To quantify the quality of each edit operation, a cost is assigned to them depending on the attributes on the involved nodes or edges. $C_n^S(a,i)$ is the cost of substituting the node v_a by the node v'_i, $C_n^D(a)$ is the cost of deleting the node v_a and $C_n^I(i)$ is the cost of inserting the node v'_i. Similarly, $C_e^S(a,i,b,j)$ is the cost of substituting the edge $e_{a,b}$ by the edge $e'_{i,j}$, $C_e^D(a,b)$ is the cost of deleting the edge $e_{a,b}$ and $C_e^I(i,j)$ is the cost of inserting the edge $e'_{i,j}$. Thus, the GED is defined as the transformation from one graph into another that obtains the minimum cost.

This graph transformation can be defined through a node-to-node mapping f between nodes of both graphs. In this way, we represent the mapping from node v_a to node v'_i as $f(a) = i$. We suppose both graphs have the same number of nodes since they have been expanded with new nodes that have a special attribute. We call these new nodes as $Null$. To assure all possible combinations of node deletion, insertion and substitution are considered, each of the two graphs to be compared is extended by the number of nodes of the other graph. Thus, the final two graphs have the same number of nodes and f is imposed to be bijective. Note, the edit operations on the edges (deletion, insertion or substitution) are forced by the mapping of the nodes that the edges connect. For instance, an edge has to be deleted if a node connected to it is deleted. Given the mapping $f(a) = i$ from node v_a to node v'_i, we say that represents a node substitution if both nodes are not $Null$. Contrarily, if node v'_i is a $Null$ and v_a is not, we say that it represents a deletion. Finally, if node v_a is a $Null$ and v'_i is not, we say that it represents an insertion. Similarly happens with the edges. The cases that both nodes or both edges are $Null$ are not considered since they are defined as the costs are always zero. Both graphs have been extended with a $Null$ node to incorporate a node deletion and a node insertion.

In the rest of this section we first summarise how the GED is computed; then we list the learning models applied to learn the edit cost functions presented until now; finally we list the embedding methods that transform graphs into vectors.

2.1 Computing graph edit distance

Computing the GED is cast as a shortest path problem, often implemented as an A* search algorithm [23,17]. These types of problems are considered to be NP-complete [20]. Thus, the computational complexity of these methods is exponential in the number of nodes of the involved graphs. For this reason, greedy algorithms have been presented that deduce a sub-optimal distance and mapping between nodes in polynomial time. The main idea is to optimise local criteria instead of global criteria. For instance, the Graduated assignment [22], the Bipartite graph matching [32,39–41,34] or the Greedy edit distance algorithm [33,13]. Other algorithms seem to obtain more accurate correspondences at the expense of increasing the computational cost [1] or [4].

These greedy algorithms define a bi-dimensional matrix in which the number of rows or columns is related to the graph order and each cell represents the cost of mapping a pair of nodes (one per graph) and their local structures (for instance, the connected edges and the nodes connceted to these edges). Exceptionally, the Believe matching algorithm [37] deduces a node-to-node mapping in linear computational cost and without the computation of the bi-dimensional cost matrix, at the expense of needing an initial small set of node-to-node mappings.

All of these algorithms have the substitution, insertion and deletion costs of nodes and edges as input parameters. To this end, the edit cost between two graphs given a specific node-to-node mapping f between nodes of both graphs is defined as the addition of the substitution, deletion and insertion costs of their local structures:

$$EditCost_f(G,G') = \sum_{\forall Subs_f(v_a,v'_i)} C^S(a,i) + \sum_{\forall Del_f(v_a)} C^D(a) + \sum_{\forall Ins_f(v'_i)} C^I(i) \tag{1}$$

$C^S(a,i)$ denotes the cost of substituting the local structure S_a centred at node v_a by the local structure S'_i centred at node v_i. $C^D(a)$ denotes the cost of deleting the local structure S_a and $C^I(i)$ denotes the cost of inserting the local structure S'_i. These local structure costs depend on the structure itself and also on the costs on nodes and edges $C^S_n(a,i)$, $C^D_n(a)$, $C^I_n(i)$, $C^S_e(a,i,b,j)$, $C^D_e(a,b)$ and $C^I_e(i,j)$. Finally, $Subs_f$, Del_f and Ins_f represent the set of node substitution, deletion and insertion operations, respectively, carried out to transform G into G'. The sub-optimal graph edit distance is defined as:

$$GED_{sub-optimal}(G,G') = \min_{\forall f}(EditCost_f(G,G')) \tag{2}$$

2.2 Learning models for graph edit distance

Seven methods have been presented to learn the edit costs, which are summarised in Table 1. Considering the objective function in the learning method, most of them aims to maximise the recognition ratio or the correspondence accuracy. Only the first two presented methods have another optimisation function. Most of them assume the substitution costs are weighted Euclidean distances and learn the weighting parameters [7, 24, 12]. Another one, [10], considers the insertion and deletion costs as constants and then applies optimisation techniques to tune these two parameters. Finally, in [2], the optimisation method learns the weights of the weighted Euclidean distance and also the insertion and deletion constant costs. Nevertheless, there are two other papers that define the edit costs as functions. The first one [30] introduces a probabilistic model of the distribution of graph edit operations that allows them to derive edit costs. The second paper [29] is based on a self-organising map model 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 optimised with regard to Dunn's index [16]. Except for the algorithm in [10], these methods need the attributes on the nodes and edges to be real numbers. If it was not the case, the model has to be adapted, for instance, assigning a numerical value to each non-numerical value.

The method we present is close to methods [30, 29] since it synthesises a function instead of learning weights or constants. One of the drawbacks of these methods is the runtime of the learning algorithm since the algorithm is based on an iterative optimisation criterion in which in each iteration the GED between all combinations of graphs in the learning set is computed. Our method avoids this process.

Finally, the seven methods mentioned before deduce, in a sub-optimal way, the edit costs on nodes and edges, C_n^S, C_n^D , C_n^I, C_e^S, C_e^D and C_e^I. Our method is the first one that deduces, also in a sub-optimal way, the edit costs on the stars, C^S, C^D and C^I. This is because we assumed adding some structural information could make the learning method to return better correspondences although being sub-optimal the algorithm.

2.3 Embedding graphs into vectors

Graphs have some limitations when they are applied to machine learning due to their intrinsic relational representation. This is because some trivial mathematical operations used in the traditional numeric machine learning representations have not an equivalence in the graph domain. Given an arbitrary set of graphs, a possible way to address this problem is to define an embedding function from the graph domain to a vector space [21]. However, defining such embedding functions is extremely challenging, when the constraints on time efficiency and preserving the underlying structural information is concerned. Explicit graph embedding is based on defining a function that, given

Ref.	Authors	Objective function	Learning method
2005 [**29**]	Neuhaus Bunke	Average of 8 indices: Davies-Bouldin, Dunn, C, Goodman-Krusk, Calinski-Haraba, Rand, Jaccard, Fowlkes-Mallo	The method learns a neural network to define the substitution, deletion and insertion costs on nodes and edges.
2007 [**30**]	Neuhaus Bunke	Dunn Index	The method learns a probability density function to define the substitution, deletion and insertion costs on nodes and edges.
2009 [**7**]	Caetano McAulery Cheng Le Smola	Correspondence accuracy	The method learns the weights of the weighted Euclidean distance to define the substitution cost on nodes and edges. Insertion and deletion of nodes and edges are not learned and assumed to be constant
2012 [**24**]	Leordeanu Sukthankar Herbert	Recognition ratio	The same than [7].
2015 [**10**]	Cortés Serratosa	Correspondence accuracy	The method learns the deletion and insertion costs on nodes and edges as constants (Real numbers). The substitution cost is the Euclidean distance between the attributes on nodes or on edges.
2016 [**12**]	Cortés Serratosa	Correspondence accuracy	The same than [7].
2018 [**2**]	Algabli Serratosa	Correspondence accuracy	The method learns the weights of the weighted Euclidean distance to define the substitution cost and also learns the deletion and insertion costs as constants on nodes and edges.

Table 1: Learning models for graph edit distance.

a graph, generates a point in an Euclidean space. These embedding functions are divided into four classes. 1) Graph probing [26] that measures the frequency of specific substructures. 2) Spectral graph theory [6], which analyses the structural properties of graphs in terms of eigenvectors and eigenvalues. 3) Dissimilarity measurements [15], in which the function depends on its distance to a selected set of graphs. 4) Geometric deep learning [14], in which the embedding uses deep neural networks.

Our learning method uses an explicit graph embedding based on graph probing (Section 3.3). We selected this method since the structural and semantic information contained in a star is limited and the computational cost of embedding a star is only linear with respect to the number of outgoing edges per node.

We point out that in the neural-network research field, recursive neural networks [3] have been recently defined, which are neural networks that allow structured input and return a structure prediction. They have been successfully used, for instance, in learning sequences and tree structures. Moreover, non-parametric small random networks for structures [47] have also been recently presented, which are based on an implicit embedding of the graphs, realised via representing the graph as a binary relation. This paper is out of the scope of these new methods.

3 The proposed learning model

In this section, we first present a general scheme of our learning model and then we explain the main two steps of this scheme. For simplicity, we have defined the local structure as a star, although other structures could be used. Stars are composed of a central node, the adjacent edges and also the nodes connected to these edges.

There are two main differences between our model and the other seven models commented above.

- The first one is that we present a general framework that can be particularized in several techniques, but the other ones present directly one of these techniques. For this reason, in the rest of the current section, we present a general model to learn the functions that define edit costs of the GED. This model opens the door for the learning algorithms to be applied on our model to learn these costs. In Section 4, we present two concrete techniques of our model. The first one is based on a probability density function learned through a multi-distribution Gaussian and the second one is based on a linear model learned through a neural network.

- The second one is that our general framework learns the edit functions on the local structures of the graphs, that is, it learns functions $C^S(a,i)$, $C^D(a)$ and $C^I(i)$. Contrarily, the other ones learn the edit functions on the nodes and edges, that is, $C_n^S(a,i)$, $C_n^D(a)$, $C_n^I(i)$, $C_e^S(a,i,b,j)$, $C_e^D(a,b)$ and $C_e^I(i,j)$. We believe that including the local information makes the learning algorithm to deduce more accurate edit costs. The paper in [2] could be considered somehow

between both options since it learns the edit costs of nodes and edges ($C_n^S(a,i)$, $C_n^D(a)$, $C_n^I(i)$, $C_e^S(a,i,b,j)$, $C_e^D(a,b)$ and $C_e^I(i,j)$) but considering the stars of the nodes S_a and S'_i (the central node v_a or v'_i and their neighbouring edges and nodes).

3.1 General problem setting

Figure 3 shows the basic scheme of our learning method. We want to learn the substitution, insertion and deletion costs of stars C^S, C^D and C^I through a supervised learning method.

- Database: In machine learning, registers of databases are composed of an element and their class. In our case, the element of the p register in the database is composed of a pair of graphs $(G^p, G^{p\prime})$ and their ground-truth correspondence $\hat{f}^p$. These ground-truth correspondences $\hat{f}^p$ have been deduced by an external system (human or artificial) and they are considered to be the best mappings for our learning purposes. Note that these ground-truth correspondences are independent of the definition of the edit costs. The aim of the learning method is to define these edit costs as functions so that the deduced correspondences become close to the ground-truth correspondences $\hat{f}^p$ for all pairs of graphs $G^p, G^{p\prime}$.

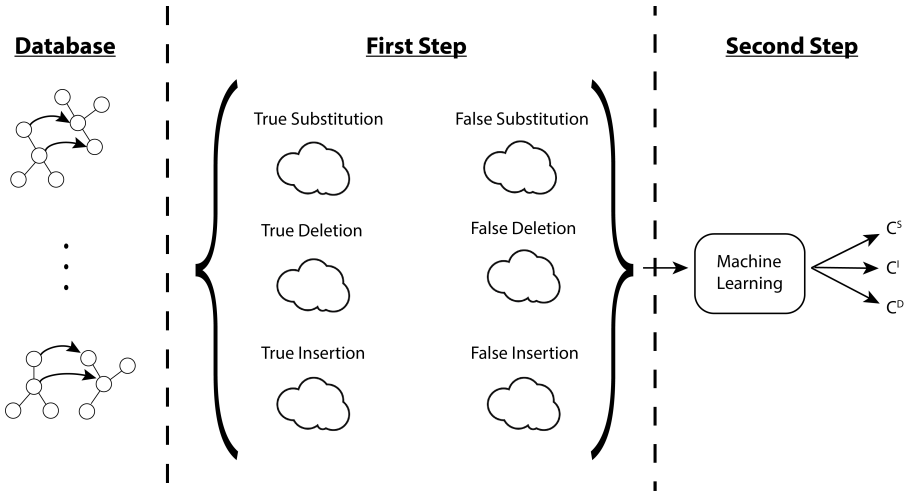


Fig. 3: General machine learning scheme composed of two steps. In the first one, six sets of stars are defined. In the second one, a supervised learning algorithm is used. The registers of the database are composed of pairs of graphs and their ground-truth correspondence.

This representation of the data in the databases is not new at all. It has been used by the learning methods [2,7,10,12]. Clearly, an expert is needed

to impose these correspondences, which could be a time-consuming task. This is the reason why methods [9,11] implemented a graphical human interactive application to make easier this task. In those cases, the human selected salient points of the images in which the graph nodes were located. Thus, this database structure has been used in [9,11] to locate a fleet of robots given only the images taken from their embedded cameras and some human interaction, when the completely automatic method failed. Another example of this database structure is the human identification through fingerprint matching. Given two fingerprints, a specialist decides which is the best correspondence between minutiae of these fingerprints. Thus, the specialist knows nothing about the GED nor edit costs and therefore the correspondence that the specialist decides is not influenced by these parameters. Note this data representation has the advantage that the data does not need to be previously classified (we do not need to know the "class" of the image the robot is looking at) but it is needed to map some salient points of pairs of images or the nodes of their graph representations.

- First step: In the first step of our scheme (Section 3.2), six different sets of stars are defined given the database registers composed of two graphs and their ground-truth correspondence. $TrueSubstitution$: The stars that are mapped by the ground-truth correspondences. $FalseSubstitution$: The stars that are not mapped. $TrueDeletion$: The stars that the ground-truth correspondences impose that have to be deleted. $FalseDeletion$: The stars that do not have to be deleted. $TrueInsertion$: The stars that the ground-truth correspondences impose that have to be inserted. $FalseInsertion$: The stars that do not have to be inserted. Stars of both graphs can appear in several sets.

- Second step: In the second step of our scheme (Section 3.3), our machine learning model deduces functions C^S, C^D and C^I, given the six sets of stars. It is based on embedding the stars into vectors and applying classical machine learning algorithms on these vectors. We like our graph matching algorithm to deduce node-to-node correspondences that would be close to the ground-truth correspondences.

Note several correspondences may be optimal for graph edit distance (Equation 2) in the learning databases. Thus, some of them might not be the ground-truth correspondence. If the learning method optimised the graph edit distance, then it might happen that the learned costs could generate correspondences that, although been optimal, are not the ground-truth correspondence. In our learning method, the ground-truth correspondence is implicitly considered in the embedding procedure and the optimisation procedure tends to deduce correspondences close to it. Thus, the fact of having several optimal correspondences does not influence on our method since they are considered as other correspondences that the method does not want to optimise.

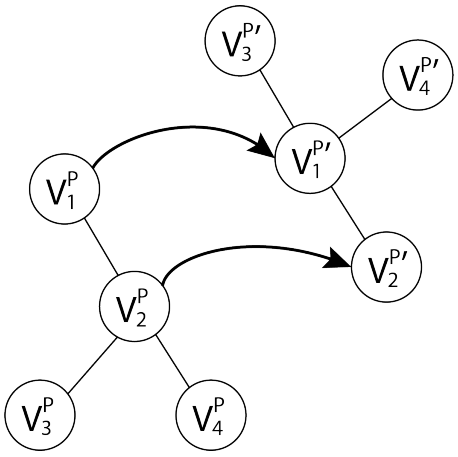


Fig. 4: Ground-truth correspondence $\hat{f}^p$ from G^p to $G^{p\prime}$. The non mapped nodes are deleted (the two ones that belong to G^p) or inserted (the two ones that belong to G'^p)

3.2 Defining the six sets of stars

In this section, we define the six sets of stars introduced in the previous section. The following rules concern to the whole registers in the database, supposing that the p register in the database is composed of a pair of graphs $(G^p, G^{p\prime})$ and their ground-truth correspondence $\hat{f}^p$. Moreover, we suppose that $S_a^p \in G^p$ and $S_i^{p\prime} \in G^{p\prime}$. The three first sets are defined by the explicit application of the ground-truth correspondences. The three last ones are defined by the combinations of mappings not allowed by the ground-truth correspondences.

$TrueSubstitution$: Composed of all pairs of stars $\{S_a^p, S_i^{p\prime}\}$ that the ground-truth correspondence imposes $Subs_f(v_a, v_i')$.

$TrueDeletion$: Composed of stars S_a^p that the ground-truth correspondence imposes $Del_f(v_a)$.

$TrueInsertion$: Composed of stars $S_i^{p\prime}$ that the ground-truth correspondence imposes $Ins_f(v_i')$.

$FalseDeletion$: Composed of all combinations of pairs of stars $\{S_a^p, S_j^{p\prime}\}$ that $j \neq i$ and also all combination of pairs of stars $\{S_b^p, S_i^{p\prime}\}$ that $b \neq a$ if the ground-truth correspondence imposes $Subs_f(v_a, v_i')$.

$FalseDeletion$: Composed of all stars S_p^a that the ground-truth correspondence imposes $Subs_f(v_a, v_i')$.

$FalseInsertions$: Composed of all stars $S_i^{p\prime}$ that the ground-truth correspondence imposes $Subs_f(v_a, v_i')$.

Figure 5 shows the classes of pairs of stars previously defined, given the substitutions, deletions and insertions of the example in Figure 4.

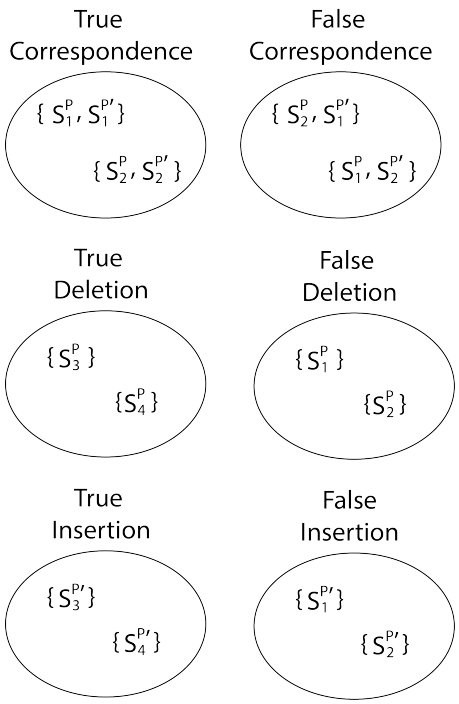


Fig. 5: Representation of the six classes given by the example in Figure 4.

3.3 Embedding stars into vectors

The aim of this paper is to present a model to learn costs C^S, C^D and C^I based on a classical machine-learning method. As commented in Section 2.3, classical machine learning methods need the prototypes to be represented as vectors (points in a vector space) instead of graphs. Nevertheless, the new papers published on this field define neural networks such that the input can be defined as a graph. Considering that we are not going to use recursive neural networks [3] neither small random networks [47], we first need to embed the stars into vectors and then model the edit costs as functions that the input parameters are vectors. Thus, the domain of these functions is a point in a vector space (in the case of deletions and insertions) or two points (in the case of substitutions) and the co-domain is a Real number. The embedding has to encode the stars into equal-sized vectors. For this reason, we define the embedding function as Φ that maps a star S_a into a point E_a in a T dimension space. It is given as $\phi(S_a) = E_a$. The value T is detailed above.

Considering the types of embedding functions summarised in Section 2.3 and that stars are small graphs with similar structure, we decided to define the embedding function Φ as a function in the first class: Graph probing. We discarded the embedding functions based on the spectral graph theory because

they need the graphs to be much larger than stars and also we discarded the functions based on deep learning because the number of samples were too low. Finally, the dissimilarity measurements methods were also discarded since they need a set of graphs (or stars in our case) to be used each time the embedding is computed.

Figure 6 graphically shows the embedding of the star S_a. The first N elements are the attributes on the nodes and the next one is the number of nodes of the star, n_{S_a}. The next cells are filled by the histograms generated by the attributes on the external nodes and the attributes on the external edges. Histograms $h_{v(i)}(S_a)$ and $h_{e(i)}(S_a)$ represent histograms generated by the i^{th} attribute on the nodes and edges, respectively. Moreover, N and M are the number of attributes on the nodes and edges, respectively. Finally, $\tilde{N}$ and $\tilde{M}$ are the number of bins of the node and edge histograms, respectively.

This representation has been inspired by the one presented in [26]. In that case, the model embedded a whole graph into a vector. Since we want to embed a star, which is a specific structure of a graph, we have somehow used their embedding model. Thus, we have that the number of dimensions of the Euclidean space is $T = N + 1 + \tilde{N} * N + \tilde{M} * M$.

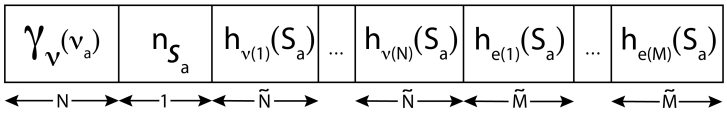


Fig. 6: Embedding of star S^a into the vector E^a .

Then, given the six sets defined in the previous section, our method defines three matrices as shown in Figure 7.

- The $SubstitutionMatrix$ has three main columns. The first one has the vectors E_a calculated by the embedding function given the first stars S_a of the pairs of stars in the sets $TrueSubstitution$ or $FalseSubstitution$. Similarly, the second column has the vectors $E_i{}'$ calculated by the embedding function given the second stars $S_i{}'$ of the pairs of stars in the same sets. Finally, there is a one in the third column if the pair of stars belongs to the $TrueSubstitution$ set and a zero if it belongs to the $FalseSubstitution$ set.

- The $DeletionMatrix$ has two main columns: The first one has the vectors E_a calculated by the embedding function given the the stars S_a in the sets $TrueSubstitution$ or $FalseSubstitution$. There is a one in the second column if S_a belongs to the $TrueDeletion$ set and there is a zero if it belongs to the $FalseDeletion$ set.

- Similarly occurs with the $InsertionMatrix$ but considering the stars S'_i in the sets $TrueInsertion$ or $FalseInsertion$.

$DeletionMatrix$, $InsertionMatrix$ and $SubstitutionMatrix$ are used to learn the edit functions C^D, C^I and C^S, respectively. The first column in the $DeletionMatrix$ and $InsertionMatrix$ and the first two columns in the

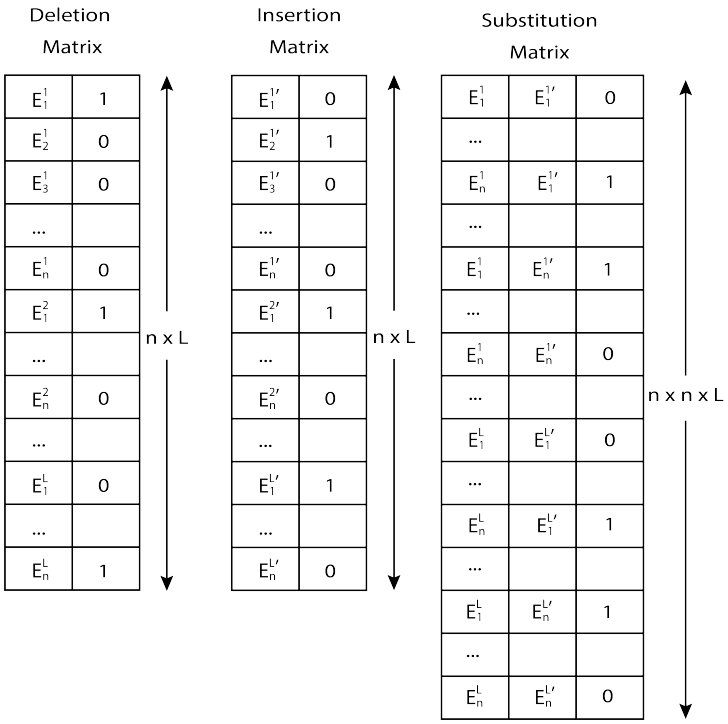


Fig. 7: The three matrices used to learn the edit costs. The last column of them is the information of the classes the points belong to.

$SubstitutionMatrix$ are the input data of the machine learning method, whereas the last column of them is the class to be learned. Note that these functions are learned independently.

4 Particularization of our learning framework

In the previous sections, we have presented a general framework to learn the edit functions. Although this framework could be concretized into different methods, we present, in this section, only two different particularizations. After these edit functions having been learned, several graph-matching algorithms could be applied that use these edit functions in the classification process. In the experimental evaluation, we computed the GED by the bipartite graph-matching algorithm [39]. In this case, adapting the algorithm only means how C^S, C^D and C^I are defined in it. In the original definition of the algorithm, these costs were computed considering that stars are graphs with a specific structure.

In the next two sub-sections, we show how we deduce the edit costs.

4.1 Neural Network

We model C^S by a regression function learned through an artificial neural network, NN^S. In the learning phase, the machine learning is fed by $SubstitutionMatrix$ (the first columns are the data and the last column is the ideal output of the substitution function). When the neural network NN^S has learned the regression function, the substitution cost $C^S(S_a, S'_i)$ is computed as one minus the output of this neural network, NN^S. We want the cost to be low if the output of the neural network is close to 1 (the ground-truth correspondence imposes that the stars have to be substituted). That is, the neural network regression deduces the stars have to be substituted.

$$C^S(S_a, S'_i) = 1 - Output(NN^S, [E_a, E'_i]) \tag{3}$$

We model C^D by another regression function based on an artificial neural network, NN^D, in a similar way than C^S. Nevertheless, in this case, we fed the machine learning by $DeletionMatrix$. The first columns are the data and the last column is the ideal output of the deletion function.

$$C^D(S_a) = 1 - Output(NN^D, [E_a]) \tag{4}$$

Similarly occurs with the insertion cost C^I but using the information of $InsertionMatrix$. The first columns are the data and the last column is the ideal output of the insertion function.

$$C^I(S'_i) = 1 - Output(NN^I, [E'_i]) \tag{5}$$

The method in [29] is similar to this particularisation except for it deduces the costs on nodes and edges but not the costs on stars. Thus, six matrices are defined, instead of our three matrices. Three matrices to define the costs of substitution, deletion and insertion on the nodes and three more for the costs on the edges. The embedding is simpler and it is only composed of the attributes on nodes and edges and there are not histograms. Then, in [29], we have $E^a = \lambda_v(v_a)$ and $E'^i = \lambda_v(v'_i)$. Similarly happens for the edges.

4.2 Probability density distribution

We define C^S by two probability density functions based on a mixture of Gaussians, $PdfTrue^S$ and $PdfFalse^S$. The first density function is modelled by columns that have the information of E^a and E'_i in the $SubstitutionMatrix$, but only using the rows that have a 1 in the last column (the ground-truth correspondence imposes that the stars have to be substituted). The second density function is modelled in a similar way but only using the rows that have a 0 in the last column (these combinations of stars do not appear in the ground-truth correspondence).

Thus, the substitution cost $C^S(S_a, S'_i)$ is defined as the subtraction of the probabilities obtained from these probability density functions (Equation 6).

Constant 1 is needed to assure the cost is always positive. We want the cost to be low if the probability obtained from the set $TrueSubstitution$ is high or the probability obtained from the set $FalseSubstitution$ is low.

$$C^S(S_a, S_i') = 1 - Prob(PdfTrue^S, [E_a, E_i']) + Prob(PdfFalse^S, [E_a, E_i']) \quad (6)$$

Functions C^D and C^I are modelled in a similar way. Nevertheless, matrices $DeletionMatrix$ and $InsertionMatrix$ are used. Thus, we have:

$$C^D(S_a) = 1 - Prob(PdfTrue^D, [E_a]) + Prob(PdfFalse^D, [E_a]) \quad (7)$$

$$C^I(S_i') = 1 - Prob(PdfTrue^I, [E_i']) + Prob(PdfFalse^I, [E_i']) \quad (8)$$

In the same way as the neural network case, the method in [30] is similar to this particularisation except for it deduces the costs on nodes and edges but not the costs on stars. Again, the embedding is simpler, and it is only composed of the attributes on nodes and edges and there are not histograms.

5 Experimental evaluation

The presented method has been validated using fourteen databases, which are available at [42]. All of them were used to test other learning methods and were previously published, for instance, the ones that belong to the public graph repository Tarragona_Graphs [27]. The main characteristic of these databases is that their registers are not only composed of a graph and its class, but they are composed of a pair of graphs and a ground-truth correspondence between them, as well as their class. This register structure is useful to analyse and develop graph matching algorithms and to learn their parameters in a broad manner. Tables 2, 3, and 4 shows the main features of the fourteen databases.

We have measured the quality of the learning algorithms through the accuracy of the returned correspondence. The accuracy of a graph matching given a pair of graphs is defined as the inverse of the normalised Hamming distance between the returned correspondence and the ground-truth correspondence. The returned accuracy of a learning method is the mean of all the accuracies computed in a database. We have experimentally validated our learning method given a methodology composed of two steps. In the first one, we have learned the edit costs given our method and other published ones. In the second one, we have applied the general scheme presented in Figure 2, in which the graph matching was the Bipartite graph matching [40], and we have deduced the accuracies of the returned correspondences. The accuracy is defined as the inverse of the normalised hamming distance between the returned correspondence and the ground-truth correspondences $\hat{f}^p$ for all pairs of graphs $(G^p, G^{p\prime})$. The returned accuracy of a learning method is the mean of all the accuracies computed in a database.

		First Experiment						
		House 90	***Hotel 90***	***House 1***	***Hotel 1***	***Noise***	***Rotate***	***Shear***
Graph	**Learn**	8	8	37	68	68	68	68
	Test	6	6	36	66	66	66	66
	Val.	8	8	74	66	66	66	66
Corr.	**Learn**	4	4	37	34	34	34	34
	Test	3	3	36	33	33	33	33
	Val.	4	4	37	33	33	33	33
Num. of classes		1	1	1	1	1	1	1
Num. of attrib		60	60	60	60	60	60	60
Description					*SIFT*			
Avg. num. nodes		30	30	30	30	35	35	35
Avg. num. edges		156	154	156	152.7	180.6	183.8	185.7
Max. nodes		30	30	30	30	35	35	35
Max. edges		158	156	158	158	184	184	186
Max. D./I.		0	0	0	0	0	0	0
Avg. null D./I.		0	0	0	0	0	0	0

Table 2: Main features of the fourteen databases, first experiment. The first row shows in which experiment the databases have been used.

Our method has been tested through four different specifications: Neural Network (NN), Probability density function (PDF), Neural Network without histograms (NNnoHis) and Probability density function without histograms (PDFnoHis). In the last two options, the embedded domain does not have the histograms. That is, the vector shown in Figure 6 is only composed of the first $N + 1$ bins. Tables 5 and 6 show the technical specifications that have achieved the best results. Considering the neural network experiments, symbol $-$ means that this experiment was no tested. The whole neural networks have three layers. The three numbers in a cell are the number of neurons of the input layer, the hidden layer and the output layer. Considering the probability density distributions, we only could synthesise them in the case of $PDFnoHis$ in the second round of experiments.

Tables 7, 8 and 9 show the accuracy of the databases in Tables 2, 3, and 4 given the four particularizations of our method and the methods in [45, 7, 24, 12, 2, 10, 30, 29]. The values in these methods have been extracted from the experimental sections of those papers, for this reason, some cells are not filled. We have also added the results presented in [27]. In that paper, authors present some accuracy results that the edit costs have been manually imposed

		Second Experiment		
		Letter High	***Letter Med***	***Letter Low***
Graph	**Learn**	750	750	750
	Test	750	750	750
	Val.	750	750	750
Corr.	**Learn**	37500	37500	37500
	Test	37500	37500	37500
	Val.	37500	37500	37500
Num. of classes		15	15	15
Num. of attrib		2	2	2
Description			*(x,y)*	
Avg. num. nodes		4.6	4.6	4.6
Avg. num. edges		6.2	6.4	9
Max. nodes		8	9	9
Max. edges		12	14	18
Max. D/I.		4	5	5
Avg. null D./I.		0.4	0.4	0.4

Table 3: Main features of the fourteen databases, second experiment. The first row shows in which experiment the databases have been used.

as constants, given two different configurations of the Bipartite graph matching algorithm. One configuration that the edit costs are computed through the stars and another one that the edit costs are computed through the degree (a star without the external nodes).

Note that our method implemented as a probability density function without the histograms in the embedded vector is almost similar to the method presented in [29], except for we also embed the number of edges that the star has and some implementation details that we do not know because they do not appear in that publication. Similarly happens with our method implemented as a neural network without the histograms in the embedded vector and the method presented in [30].

We have performed three rounds of experiments. The difference between them is the type of ground-truth correspondences that the databases have.

- In the first round of experiments, we have used the first seven databases in Tables 2, 3, and 4. In these databases, the ground-truth correspondences do not have deletion nor insertion operations. Then, it is not possible for the learning methods to learn these edit functions. Thus, they are useful to analyse how the learning algorithms deduce the substitution cost C^S without

		Third Experiment			
		Boat	***East Park***	***East South***	***Resid***
Graph	**Learn**	5	5	5	5
	Test	5	5	5	5
	Val.	-	-	-	-
Corr.	**Learn**	25	25	25	25
	Test	25	25	25	25
	Val.	-	-	-	-
Num. of classes		1	1	1	1
Num. of attrib		64	64	64	64
Description		*SURF*			
Avg. num. nodes		50	50	50	50
Avg. num. edges		278.4	276	278.8	276.4
Max. nodes		50	50	50	50
Max. edges		282	280	282	278
Max. D/I.		50	47	50	50
Avg. null D./I.		32	34	37	32

Table 4: Main features of the fourteen databases, third experiment. The first row shows in which experiment the databases have been used.

Number of neurons per layer				
Algorithm		***First Experiment***	***Second Experiment***	***Third Experiment***
NN	NN^S	1321-500-2	45-45-2	1409-500-2
	NN^D	-	23-23-2	705-352-2
	NN^I	-	23-23-2	705-352-2
NNnoHis	NN^S	122-122-2	5-5-2	130-130-2
	NN^D	-	3-3-2	65-65-2
	NN^I	-	3-3-2	65-65-2

Table 5: Configuration of the neural networks and probability density functions in the four specifications of our method: NN and NNnoHis. '-' means that we were not able to define the model.

Number of Gaussian distributions				
Algorithm		***First Experiment***	***Second Experiment***	***Third Experiment***
PDF	$PdfTrue^S$	-	2	-
	$PdfTrue^D$	-	1	-
	$PdfTrue^I$	-	1	-
PDFnoHis	$PdfTrue^S$	-	2	-
	$PdfTrue^D$	-	1	-
	$PdfTrue^I$	-	1	-

Table 6: Configuration of the neural networks and probability density functions in the four specifications of our method: PDF and PDFnoHis. '-' means that we were not able to define the model.

influence of the insertion and deletion costs, C^D and C^I. This first round of experiments is useful to fairly compare our framework to the methods [7, 24, 12, 30, 29], which only learn the substitution costs.

When the substitution costs have been learned, the pattern recognition model applies the graph matching algorithm with the learned substitution costs C^S and the insertion and deletion costs $C^D = Inf$ and $C^I = Inf$. These values are set to force the graph matching algorithm to return the correspondences without insertion and deletion operations, similarly to the ground-truth correspondences.

The first we realise is that our method with the neural network (NN) and the one in [30] (which is the same as our method but without embedding the number of edges per node and the histograms) obtains the maximum accuracy (all the node-to-node mappings are properly assigned). Moreover, the algorithm that computes the probability density method is not able to deduce the multimodal Gaussian function and returns "ill conditioned". We believe it is not possible to deduce the Gaussian functions due the graph nodes have a high number of attributes (thus the multimodal Gaussian has 1321 and 122 dimensions in the first experiment, and 1409 and 130 in the third experiment) but the number of correspondences per class is low (see Tables 2, 3, and 4).

- In the second round of experiments, we analyse the complete method (learning the substitution, deletion and insertion functions), but using graphs that have less attributes to void the "ill conditioned" in the probability density function. In this case, we have used the Letter databases presented in Tables 2, 3, and 4, which their graphs have only two attributes in the nodes and their correspondences have insertions and deletions. Our method has been compared to the methods that learn the substitution, deletion and insertion costs, which are [2, 10, 30, 29, 27].

	First Experiment						
Algorithm	***House 90***	***Hotel 90***	***House 1***	***Hotel 1***	***Noise***	***Rotate***	***Shear***
PDF	i.c.	i.c.	i.c.	i.c.	i.c.	i.c.	i.c.
NN	1	1	1	1	1	1	1
PDFnoHis	i.c.	i.c.	i.c.	i.c.	i.c.	i.c.	i.c.
NNnoHis	1	1	1	1	1	1	1
[29]	i.c.	i.c.	i.c.	i.c.	i.c.	i.c.	i.c.
[30]	1	1	1	1	1	1	1
[7]	0.85	0.90	-	-	0.67	0.98	0.57
[24]	-	-	1	0.98	-	-	-
[12]	0.77	0.79	1	1	0.81	0.45	0.82
[2]	1	1	0.88	0.97	0.99	1	1
[27] Star	-	-	-	-	-	-	-
[10] Degree	-	-	-	-	-	-	-
[10]	-	-	-	-	-	-	-
[45] $C_{vd} = 1$ $C_{ve} = 1$	-	-	-	-	-	-	-
[45]$C_{vd} = .5$ $C_{ve} = .5$	-	-	-	-	-	-	-

Table 7: Accuracies deduced by the methods referenced in the first column given the fourteen databases. The cells marked with a "−" are values not given in the original papers. "i.c" means "ill conditioned" (the learning method is not able to generate the Gaussian function). The first four rows show the accuracies of our method considering four adaptations (NN and PDF are Neural network and Probability density function methods: NNnoHis and PDFnoHis are the same adaptations but without the histograms in the embedding vector). First experiment.

Similarly to the first round of experiments, the neural network with histograms is the one that achieves the highest accuracy. Moreover, our method with the probability density functions continues not being able to deduce the Gaussian function. Nevertheless, our method without the information of the histograms is able to synthesise the Gaussian functions and return some results similar to other methods. In these databases, it seems as the histograms positively contribute to the learning process since our method with neural networks returns a higher accuracy than the one in [30]. Finally, authors of paper [27] claims that their presented results are the best ones that they had computed, given different combinations of edit costs. Our method obtains bet-

Algorithm	Second Experiment		
	Letter High	***Letter Med***	***Letter Low***
PDF	i.c.	i.c.	i.c.
NN	0.91	0.90	0.98
PDFnoHis	0.83	0.76	0.93
NNnoHis	0.89	0.87	0.97
[29]	0.80	0.75	0.90
[30]	0.86	0.87	0.91
[7]	-	-	-
[24]	-	-	-
[12]	-	-	-
[2]	0.82	0.70	0.85
[27] Star	0.89	0.90	0.97
[10] Degree	0.87	0.85	0.97
[10]	-	-	0.71
[45] $C_{vd} = 1$ $C_{ve} = 1$	-	-	-
[45] $C_{vd} = .5$ $C_{ve} = .5$	-	-	-

Table 8: Accuracies deduced by the methods referenced in the first column given the fourteen databases. The cells marked with a "−" are values not given in the original papers. "i.c" means "ill conditioned" (the learning method is not able to generate the Gaussian function). The first four rows show the accuracies of our method considering four adaptations (NN and PDF are Neural network and Probability density function methods: NNnoHis and PDFnoHis are the same adaptations but without the histograms in the embedding vector). Second experiment.

ter or similar results in the three databases, although the difference is not so important.

- In the third round of experiments, we analyse the complete method (learning the substitution, deletion and insertion functions), as we did in the second round of experiments, but graphs have a larger number of attributes. We used the databases in the last four columns in Tables 2, 3, and 4, in which graphs have 64 attributes. We compared our method to the ones presented in [45, 10, 30, 29]. Figure 8 shows the correspondence deduced by [10] given two images of the *Boat* database and two images of the *Letter* database. In both cases, there are correct and wrong node-to-node mappings.

	Third Experiment			
Algorithm	***Boat***	***East Park***	***South Park***	***Resid***
PDF	i.c.	i.c.	i.c.	i.c.
NN	0.14	0.15	0.18	0.17
PDFnoHis	i.c.	i.c.	i.c.	i.c.
NNnoHis	0.44	0.56	0.40	0.55
[29]	i.c.	i.c.	i.c.	i.c.
[30]	0.44	0.56	0.40	0.55
[7]	-	-	-	-
[24]	-	-	-	-
[12]	-	-	-	-
[2]	0.16	0.13	0.07	0.15
[27] Star	-	-	-	-
[10] Degree	-	-	-	-
[10]	0.22	0.21	0.20	0.20
[45] $C_{vd} = 1$ $C_{ve} = 1$	0.32	0.31	0.29	0.36
[45]$C_{vd} = .5$ $C_{ve} = .5$	0.34	0.34	0.29	0.45

Table 9: Accuracies deduced by the methods referenced in the first column given the fourteen databases. The cells marked with a "−" are values not given in the original papers. "i.c" means "ill conditioned" (the learning method is not able to generate the Gaussian function). The first four rows show the accuracies of our method considering four adaptations (NN and PDF are Neural network and Probability density function methods: NNnoHis and PDFnoHis are the same adaptations but without the histograms in the embedding vector). Second experiment.

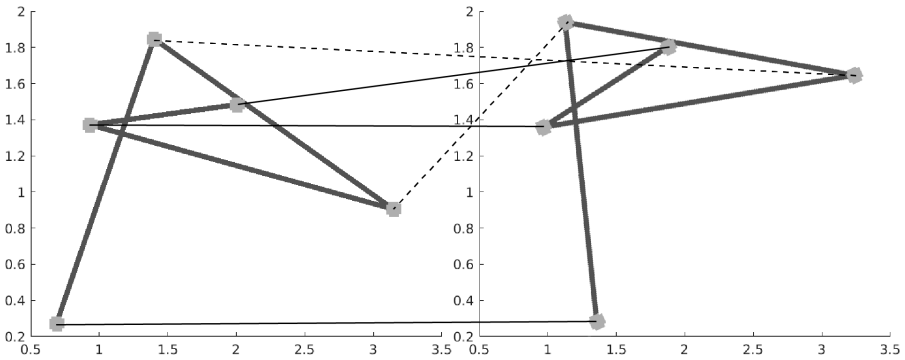
Our method based on neural networks but without using the information of the histograms is the one that obtains the best results. We suppose that, since the number of neighbours is very low (5 in average) and the number of attributes is 64, the histograms become too sparse to be useful for the learning algorithm (although fuzzy methods were tested on the histograms). As it happened in the first experiments, the learning algorithm is not able to deduce the probability density functions since the number of dimensions is too high and the number of correspondences is too low. In this experiment, the manually imposed edit costs (the last two rows in Tables 7, 8 and 9) achieved better results than the learning method presented in [10].

	First Experiment	Second Experiment	Third Experiment
Edit Operations	***SUB***	***INS, DEL, SUB***	***INS, DEL, SUB***
Type of graphs	***Many attributes***	***Few attributes***	***Many attributes***
Number of graphs	***Low***	***High***	***Low***
PDF	i.c.	i.c.	i.c.
NN	1	0.93	0.16
PDFnoHis	i.c.	0.84	i.c.
NNnoHis	1	0.91	0.48
[29]	i.c.	0.81	i.c.
[30]	1	0.88	0.40
[7]	0.79	-	-
[24]	0.99	-	-
[12]	0.80	-	-
[2]	0.97	0.79	0.12
[27] Star	-	0.92	-
[10] Degree	-	0.89	-
[10]	-	0.71	0.20
[45] $C_{vd} = 1$ $C_{ve} = 1$	-	-	0.32
[45] $C_{vd} = .5$ $C_{ve} = .5$	-	-	0.35

Table 10: Mean accuracies deduced by methods referenced in the first column given the fourteen databases (in bold the highest values per column). The blank cells are values not given in the original papers. "i.c" means "ill conditioned" (the learning method is not able to generate the Gaussian function). NN and PDF are neural network and probability density function methods. NNnoHis and PDFnoHis are the same adaptations but without the histograms in the embedding vector.

Table 10 summarises the experimental validation. In the first three rows, we highlight the edit operations considered by the ground truth correspondence (Insertion, Deletion and Substitution), the number of attributes on the nodes (two quality values: Many and Few) and the number of graphs used to learn the model (two quality values: Low and High). We consider these three features are crucial to be considered while deciding which method has to be used in a real application. Columns in the table represent the three rounds of experiments, which have been selected considering these features. Finally, values in the cells are the mean accuracies per each round of experiments.

(a) Two images of the *Boat* database. Wrong mappings appear in black lines and the correct ones in white lines.



(b) Two images of the *Letter* database. Wrong mappings appear in black dash lines and the correct ones in solid black lines.

Fig. 8: Two examples of mappings.

With these three rounds of experiments, we conclude:

- The neural network, NN, returns the best accuracies when the number of attributes is high or the number of graphs used to train the model is high (first and second round of experiments).

- Our method (neural network: NN or probability density function: PDF) cannot be used when we wish to learn the insertion, deletion and substitution costs, the number of attributes is high but the number of graphs in the learning stage is low (third round of experiments). In this case, NN returned poor results and PDF was not able to define the model.

- The neural network without embedding the histograms, NNnoHis, returns competitive accuracies in the three rounds of experiments.

- It seems as the histograms do not have to be included in the embedded vector when they are too sparse (graphs with many attributes and database with few graphs).

6 Conclusions

Edit costs functions are application dependent and usually set manually based on maximising the accuracy in the recognition process. Lately, some methods have been presented to automatically learn these functions. We have proposed a general framework to learn the substitution, deletion and insertion costs based on minimising the hamming distance between the deduced correspondences and the ground-truth correspondences. Moreover, we have concretised our framework on two models, one based on neural networks and the other one based on multimodal probability density functions. Our model has the main advantage that it is defined as a general framework, in which several learning methods can be adapted, and also it is defined such as the three most used edit operations, viz. substitution, deletion and insertion are independently learned.

We have tested our framework on fourteen public databases, in which their graphs and also their ground-truth correspondences have different characteristics, and we have empirically deduced that the neural network achieves the highest accuracy. Nevertheless, we conclude that deciding to include or discard the information of the histograms in the embedded vector depends on the sparsity of this vector. The probability density function method has obtained poor results. It may happen that it is due to the number of correspondences in the database is too low being the number of dimensions too high.

We leave, as a future work, to test our model with huge graphs and with larger number of graphs in the databases. To do so, we could use the algorithm presented in [43] that generates pairs of graphs with a ground-truth correspondence in almost linear cost. With these graphs, we could synthesise a large database of huge graphs and analyse the performance of the probability density function constructed with much more data. Moreover, using this new database, the classical neural network methods could be converted into a deep neural networks. Finally, other neural network-based approaches could be applied, like recursive neural networks or graph neural networks.

Acknowledgements This research is supported by the Spanish projects TIN2016-77836-C2-1-R and ColRobTransp MINECO DPI2016-78957-R AEI/FEDER EU; and also the European project AEROARMS, H2020-ICT-2014-1-644271.

References

1. Abu-Aisheh, Z., Raveaux, R., Ramel, J.: Anytime graph matching. Pattern Recognition Letters **84**, 215–224 (2016)
2. Algabli, S., Serratosa, F.: Embedding the node-to-node mappings to learn the graph edit distance parameters. Pattern Recognition Letters **112**, 353–360 (2018)
3. Bongini, M., Rigutini, L., Trentin, E.: Recursive neural networks for density estimation over generalized random graphs. IEEE Trans. Neural Netw. Learning Syst. **29**(11), 5441–5458 (2018). https://doi.org/10.1109/TNNLS.2018.2803523, https://doi.org/10.1109/TNNLS.2018.2803523
4. Bougleux, S., Brun, L., Carletti, V., Foggia, P., Gaüzère, B., Vento, M.: Graph edit distance as a quadratic assignment problem. Pattern Recognition Letters **87**, 38–46 (2017)

5. Bunke, H., Allermann, G.: Inexact graph matching for structural pattern recognition. Pattern Recogn. Lett. **1**(4), 245–253 (May 1983). https://doi.org/10.1016/0167-8655(83)90033-8, http://dx.doi.org/10.1016/0167-8655(83)90033-8
6. Caelli, T., Kosinov, S.: An eigenspace projection clustering method for inexact graph matching. IEEE Trans. Pattern Anal. Mach. Intell. **26**(4), 515–519 (2004)
7. Caetano, T.S., McAuley, J.J., Cheng, L., Le, Q.V., Smola, A.J.: Learning graph matching. IEEE transactions on pattern analysis and machine intelligence **31**(6), 1048–1058 (2009)
8. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. International journal of pattern recognition and artificial intelligence **18**(03), 265–298 (2004)
9. Cortés, X., Serratosa, F.: An interactive method for the image alignment problem based on partially supervised correspondence. Expert Systems With Applications **42**(1), 179–192 (2015)
10. Cortés, X., Serratosa, F.: Learning graph-matching edit-costs based on the optimality of the oracle's node correspondences. Pattern Recognition Letters **56**, 22–29 (2015)
11. Cortés, X., Serratosa, F.: Cooperative pose estimation of a fleet of robots based on interactive points alignment. Expert Systems With Applications **45**, 150–160 (2016)
12. Cortés, X., Serratosa, F.: Learning graph matching substitution weights based on the ground truth node correspondence. International Journal of Pattern Recognition and Artificial Intelligence **30**(02), 1650005 (2016)
13. Cortés, X., Serratosa, F., Riesen, K.: On the relevance of local neighbourhoods for greedy graph edit distance. In: S+SSPR. Lecture Notes in Computer Science, vol. 10029, pp. 121–131 (2016)
14. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: NIPS. pp. 3837–3845 (2016)
15. Duin, R.P.W., Pekalska, E.: The dissimilarity representation for structural pattern recognition. In: CIARP. Lecture Notes in Computer Science, vol. 7042, pp. 1–24. Springer (2011)
16. Dunn, J.C.: Well-separated clusters and optimal fuzzy partitions. Journal of Cybernetics **4**(1), 95–104 (1974). https://doi.org/10.1080/01969727408546059, https://doi.org/10.1080/01969727408546059
17. Ferrer, M., Serratosa, F., Riesen, K.: Improving bipartite graph matching by assessing the assignment confidence. Pattern Recognition Letters **65**, 29–36 (2015)
18. Foggia, P., Percannella, G., Vento, M.: Graph matching and learning in pattern recognition in the last 10 years. International Journal of Pattern Recognition and Artificial Intelligence **28**(01), 1450001 (2014)
19. Gao, X., Xiao, B., Tao, D., Li, X.: A survey of graph edit distance. Pattern Anal. Appl. **13**(1), 113–129 (Feb 2010). https://doi.org/10.1007/s10044-008-0141-y, http://dx.doi.org/10.1007/s10044-008-0141-y
20. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1990)
21. Gibert, J., Valveny, E., Bunke, H.: Graph embedding in vector spaces by node attribute statistics. Pattern Recognition **45**(9), 3072–3083 (2012)
22. Gold, S., Rangarajan, A.: A graduated assignment algorithm for graph matching. IEEE Trans. Pattern Anal. Mach. Intell. **18**(4), 377–388 (Apr 1996). https://doi.org/10.1109/34.491619, https://doi.org/10.1109/34.491619
23. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics **4**(2), 100–107 (July 1968). https://doi.org/10.1109/TSSC.1968.300136
24. Leordeanu, M., Sukthankar, R., Hebert, M.: Unsupervised learning for graph matching. International journal of computer vision **96**(1), 28–45 (2012)
25. Livi, L., Rizzi, A.: The graph matching problem. Pattern Analysis and Applications **16**(3), 253–283 (2013)
26. Luqman, M.M., Ramel, J.Y., Lladós, J., Brouard, T.: Fuzzy multilevel graph embedding. Pattern Recogn. **46**(2), 551–565 (Feb 2013). https://doi.org/10.1016/j.patcog.2012.07.029, http://dx.doi.org/10.1016/j.patcog.2012.07.029

27. Moreno-Garca, Carlos Francisco, C.X., Serratosa, F.: A graph repository for learning error-tolerant graph matching. Syntactic and Structural Pattern Recognition (2016)
28. Myers, R., Wilson, R.C., Hancock, E.R.: Bayesian graph edit distance. IEEE Trans. Pattern Anal. Mach. Intell. **22**(6), 628–635 (2000). https://doi.org/10.1109/34.862201, https://doi.org/10.1109/34.862201
29. Neuhaus, M., Bunke, H.: Self-organizing maps for learning the edit costs in graph matching. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **35**(3), 503–514 (2005)
30. Neuhaus, M., Bunke, H.: Automatic learning of cost functions for graph edit distance. Information Sciences **177**(1), 239–247 (2007)
31. Riesen, K.: Structural pattern recognition with graph edit distance. Advances in computer vision and pattern recognition, Cham (2015)
32. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image and Vision computing **27**(7), 950–959 (2009)
33. Riesen, K., Ferrer, M., Dornberger, R., Bunke, H.: Greedy graph edit distance. In: Proceedings of the 11th International Conference on Machine Learning and Data Mining in Pattern Recognition - Volume 9166. pp. 3–16. MLDM 2015, Springer-Verlag, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-319-21024-7_1, https://doi.org/10.1007/978-3-319-21024-7_1
34. Riesen, K., Fischer, A., Bunke, H.: On the impact of using utilities rather than costs for graph matching. Neural Processing Letters **48**(2), 691–707 (2018). https://doi.org/10.1007/s11063-017-9739-7, https://doi.org/10.1007/s11063-017-9739-7
35. Sanfeliu, A., Fu, K.S.: A distance measure between attributed relational graphs for pattern recognition. Systems, Man and Cybernetics, IEEE Transactions on **SMC-13** (06 1983). https://doi.org/10.1109/TSMC.1983.6313167
36. Sanfeliu, A., Alquézar, R., Andrade, J., Climent, J., Serratosa, F., Vergés-Llahí, J.: Graph-based representations and techniques for image processing and image analysis. Pattern Recognition **35**, 639–650 (2002)
37. Santacruz, P., Serratosa, F.: Error-tolerant graph matching in linear computational cost using an initial small partial matching. Pattern Recognition Letters (2018)
38. Santacruz, P., Serratosa, F.: Learning the sub-optimal graph edit distance edit costs based on an embedded model. In: S+SSPR. Lecture Notes in Computer Science, vol. 11004, pp. 282–292. Springer (2018)
39. Serratosa, F.: Fast computation of bipartite graph matching. Pattern Recognition Letters **45**, 244–250 (2014)
40. Serratosa, F.: Speeding up fast bipartite graph matching through a new cost matrix. International Journal of Pattern Recognition and Artificial Intelligence **29**, 1550010 (11 2014). https://doi.org/10.1142/S021800141550010X
41. Serratosa, F.: Computation of graph edit distance: Reasoning about optimality and speed-up. Image Vision Comput. **40**, 38–48 (2015)
42. Serratosa, F.: Graph databases. http://deim.urv.cat/ francesc.serratosa/databases/ (2015)
43. Serratosa, F.: A methodology to generate attributed graphs with a bounded graph edit distance for graph-matching testing. IJPRAI **32**(11), 1850038 (2018)
44. Serratosa, F.: Graph edit distance: Restrictions to be a metric. Pattern Recognition **90**, 250–256 (2019). https://doi.org/10.1016/j.patcog.2019.01.043, https://doi.org/10.1016/j.patcog.2019.01.043
45. Serratosa, F., Cortés, X.: Graph edit distance: Moving from global to local structure to solve the graph-matching problem. Pattern Recognition Letters **65**, 204–210 (2015)
46. Solé-Ribalta, A., Serratosa, F., Sanfeliu, A.: On the graph edit distance cost: Properties and applications. IJPRAI **26**(5) (2012)
47. Trentin, E., Iorio, E.D.: Nonparametric small random networks for graph-structured pattern recognition. Neurocomputing **313**, 14–24 (2018). https://doi.org/10.1016/j.neucom.2018.05.095, https://doi.org/10.1016/j.neucom.2018.05.095
48. Vento, M.: A long trip in the charming world of graphs for pattern recognition. Pattern Recognition **48**(2), 291–301 (2015)