Interactive Online Learning for Graph Matching using Active Strategies

Donatello Conte[a,*], Francesc Serratosa[b]

[a] *Laboratoire d'Informatique Fondamentale et Appliquée de Tours (LIFAT - EA6300) Tours, France*
[b] *Universitat Rovira i Virgili, Departament d'Enginyeria Informàtica i Matemàtiques Tarragona, Catalonia, Spain*

Abstract

In some pattern recognition applications, objects are represented by attributed graphs, in which nodes represent local parts of the objects and edges represent relationships between these local parts. In this framework, the comparison between objects is performed through the distance between attributed graphs. Usually, this distance is a linear equation defined by some cost functions on the nodes and on the edges of both attributed graphs. In this paper, we present an online, active and interactive method for learning these cost functions, which works as follows. Graphs are provided to the learning algorithm by pairs in a sequential order (online). Then, a correspondence between them is computed, and there is a strategy that, given the current pair of graphs and the computed correspondence, proposes which node-to-node mapping would most contribute to the learning process (active). Finally, the human can correct some node-to-node mappings if the human thinks they are wrong (interactive). This is the first learning method applied to graph matching that has the following two features: Being an online method and being active and interactive. These properties make our method useful in the cases that data does not arrive at once and when the human can interact on the system. Thus, given some human interactions the method would have to tend to gradually increase its accuracy. The results show

*Corresponding author
Email addresses: donatello.conte@univ-tours.fr (Donatello Conte), francesc.serratosa@urv.cat (Francesc Serratosa)

that with few interactions, we achieve better results than the offline learning state of the art methods that are currently available.

Keywords: Online learning, Active learning, Human interaction, Graph matching, Costs functions.

---

1. Introduction

Attributed graphs have found widespread applications in several research fields of structural pattern recognition [1, 2, 3, 4]. This is due to their ability to represent structured objects through unary and binary local entities. To compare the graphs, several distance measures between attributed graphs have been presented [1, 4], and the problem is usually called graph matching. Typically, the problem consists of finding the node-to-node assignment between a pair of attributed graphs that minimizes an objective function encoding local dissimilarities (a linear term) and structural dissimilarities (a quadratic term). To do so, it is necessary to define the cost functions between the linear terms and the quadratic terms of both attributed graphs, given the application at hand.

Note that a proper definition of these cost functions is crucial to achieve good classification or recognition results, and it is not an easy task. In most applications, cost functions are manually set in a trial and error process [3]. Moreover, they are usually defined as known distances, such as the Euclidean distance (in the case that the attributes are numbers), or other more complicated ones, such as the Levenshtein distance [4] (in the case the attributes are strings of characters). Use of learned cost functions would have to generate more application dependent functions that would increase the recognition accuracy. Moreover, it would not be necessary to waste time on a trial and error process to properly tune these cost functions.

Machine learning methods are broadly classified as either offline or online learning. In offline methods, learning the model is performed in a first stage. Then, use of this model occurs in a second stage in applications based on pattern recognition or clustering, among others. In contrast, for the online learning

methods, the use of the model is alternated with the learning of it. This happens when newly classified samples are available when the pattern recognition or clustering stages (among others) are in progress.

For instance, the systems that recognize people in a set of pictures select the ones in which a specific person appears, but at the same time, they keep learning how to recognize this person when new pictures appear that have been accepted by the user as containing this person.

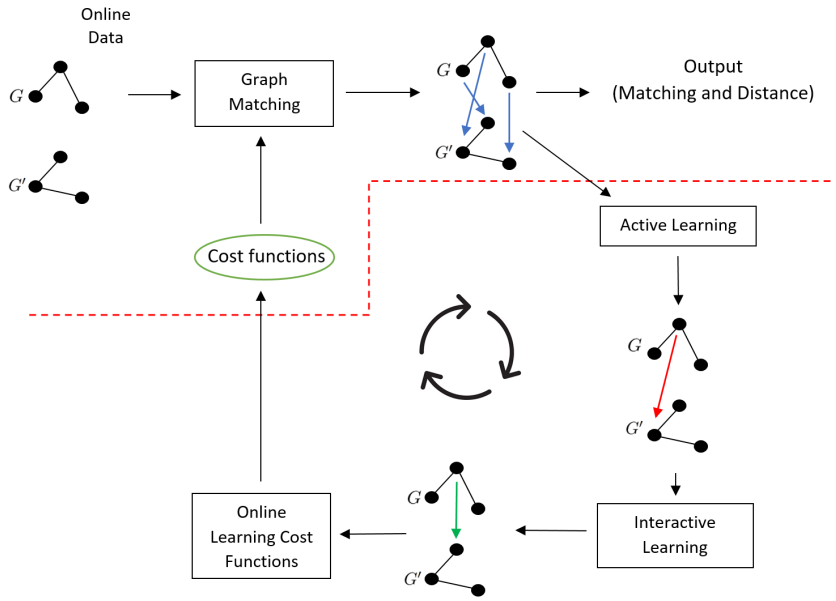


Figure 1: Online Active and Interactive Learning Framework: Our method updates the cost functions every time the human interacts. Above the dashed line is a classical graph matching with imposed cost functions.

Humans are very good at finding mappings between local sections of elements represented by structures or images given any level of abstraction. In fact, neuropsychological evidence affirms that there are four specific stages identified in the process of object recognition for a human being ([5, 6]). These stages involve processing of basic object components, such as color, depth, and form, grouping these basic components on the basis of similarity and matching with structural

descriptions in memory. Therefore mapping process is something innate in human beings. For instance, we can easily map the elements of two drawings of chemical compounds, or we can map the rear wheels of two motorbikes independent of how different the motorbikes are in the images. If these compounds or images are seen as specific graphs, this suggests that we are good at deducing the node-to-node mapping between two attributed graphs.

We propose a learning method that not only deduces the distance and node-to-node mapping between a pair of graphs but also has the ability to update the cost functions each time the human interacts. The aim of applying a learning strategy would be to deduce a more accurate distance and map the next time the graph matching is computed. To that aim, we have put a human in the loop. Because humans are good at deducing mappings between structures, in our method, the human validates the deduced node-to-node mapping or imposes a new one. We assume that this is a way to assure that the output of the matching method tends to be the one that the human desires, given some human interactions. We want to highlight that the human is not forced to interact each time the graph matching is performed. In contrast, the human only interacts when one considers it is necessary. This is an important feature of our method, as the human interaction could be much slower than the graph matching algorithm. Moreover, we have included an active method to reduce the effort asked of the human. Given the deduced node-to-node mapping, the system shows a pair of nodes (one per graph) that has been mapped. This pair of nodes is the one that has the greatest chance of being wrong. In this way, we do not ask the human to check the whole node-to-node mapping given a pair of graphs, but only one pair of nodes. This selected mapping is the most informative for updating the cost functions and therefore improving the following graph matching runs.

Figure 1 shows the main scheme of our method. The process line above the dashed line is a classical graph matching method. That is, given a pair of graphs and some imposed cost functions, the $GraphMatching$ module returns a distance and a node-to-node matching between the graphs (represented by

blue arrows in Figure 1). The *ActiveLearning* module receives this mapping and outputs a mapping composed of only a node of the first graph and a node of the second graph (highlighted in red in Figure 1). This mapping has a high chance of being wrong, considering the whole graphs and the whole node-to-node mapping. Then, the *InteractiveLearning* module, which is a human in our proposal, visualizes this mapping together with the original data, and corrects this mapping when it considers it to be wrong (the corrected mapping is highlighted in green in Figure 1). In this case, correcting means substituting one of the two nodes matched by the node-to-node mapping. Finally, the *LearningCostFunction* module updates the cost functions that encode the local and structural dissimilarities given the node-to-node mapping proposed by the human and the previous cost functions. Because this is an online learning method, we do not know how many pairs of graphs are going to be introduced into the system, or if some pairs of graphs are going to be presented to the system several times. Nevertheless, the method is ready at any time to compute a graph matching given the current cost functions.

The main contributions of this paper are:

- Presenting an online method to learn the cost functions, for the first time.
- Adapting the classical active strategies to the graph matching domain.
- Selective interaction of human onto the system.
- High learning capacity given few human interactions, shown in the experimental section.

The paper is organized as follows. In Section 2, we present the literature review of the basic used methods. Section 3 presents the basic definitions and notations used in the rest of paper. In Section 4, we describe our proposal to learn the cost functions for graph matching in an online, active and interactive way. In Section 5, we present the experimental results. Finally, Section 6 concludes the paper.

2. Literature Review

In this section, we first present the literature review of the online (Section 2.1), active (Section 2.2) and interactive (Section 2.3) learning methods in a general form. This is because our method is an adaptation of these classical methods into the graph matching field. Finally, in Section 2.4, we summarize the state of the art in offline machine learning methods applied to learn the graph-matching cost functions.

2.1. Online Machine Learning Methods

Online machine learning [7, 8] is a method of machine learning in which data become available in a sequential order and are used to update a predictor for future data, as the samples are obtained, to improve performance on the new data. Conversely, batch learning methods generate the best predictor by learning on the batch data at once. Online learning is a common technique used in areas of machine learning where it is computationally infeasible to train over the entire dataset. It is also used in situations where it is necessary for the algorithm to dynamically adapt to new patterns in the data, or when the data are generated as a function of time. An interesting example could be the classification of job advertisements [9], in which advertisements have to be constantly classified at the same time that new ones appear and other ones disappear.

Since its introduction, many papers have appeared that address various problems of online learning. As we will discuss in next Secion 2.2, online learning has been quickly associated with active querying strategies to decide which data are most useful for learning task [10]. In the papers [11, 12] the authors study the online heterogeneous transfer (OHT) learning problem, where the target data of interest arrive in an online manner, while the source data are from offline sources and can be easily annotated; the authors propose some techniques that exploit offline knowledge transfering it a online different domain. Other papers [13, 14] deal with the problems of limited query budget (when it is difficult to annotated date) and the problem of highly imbalance ratio between classes.

2.2. Active Machine Learning Methods

A machine learning algorithm can achieve a greater accuracy with fewer classified training examples if it can choose the data from which it learns [15, 16, 17]. In active machine learning, the learner queries some specific elements, and the answerer informs to which classes these elements belong. It is assumed that the answer is always correct. For this reason, the answerer, which might be another automatic system or a human annotator, is called an oracle.

In many modern machine-learning problems, active learning is well motivated if unclassified samples may be abundant, but finding the class is difficult, time-consuming or expensive to obtain [18]. Active learning has been applied in several fields, such as speech recognition [19], information extraction [20], robotics [21], transcription of text images [22] or object classification in general [23, 24, 25, 26].

All active learning scenarios involve evaluating the informativeness of unlabeled instances. There have been many proposed ways of formulating such query strategies [15]. The most commonly used ones, and therefore the ones applied in this paper, are Least Confident (LC) [27], Margin Sampling (MS) [28] and Maximum Entropy (ME) [29]. Least Confident [27] queries the element whose highest probability of belonging to a class is the lowest among all the elements. Margin Sampling [28] aims to incorporate the posterior probability of the second most likely labeled element. Maximum Entropy [29] queries the element with maximum Shannon Entropy [30] given the probabilities of belonging to the classes, because this element is less informative than others.

One of the latest paradigms of online learning is to optimise the regret function [31, 32]. Basically, this function deduces the weights to be learned by an online gradient descendent algorithm, in which the objective function is composed of the sum of a sensitivity function and a specificity function.

2.3. Interactive Machine Learning

Interactive Machine Learning is a specific type of Active Learning in which the human interacts with the system through a Human-Machine Interface (HMI).

The general idea is that the human receives the information of the state of the system through any source of information and then influences this system through any mechanism. The HMI is the space where interactions between humans and machines occur. The goal of this interaction is to allow effective operation of the machine from the human end, while the machine simultaneously feeds back information that aids the human to make decisions. Human interaction has been recently applied to image alignment for robotics pose and image alignment estimation [33, 34], 2D-camera calibration [35] or engineering drawing validation [36].

2.4. Learning Graph Matching Cost Functions

The aim of the learning methods applied to graph matching is to learn the cost functions on nodes and edges. To our knowledge, only seven papers have been published related to learning these cost functions [37, 38, 39, 40, 41, 42, 43]. An important feature of these methods is the nature of the cost functions the learning algorithm obtains. The method in [37] defines these costs as self-organizing maps, and the method in [38] defines them as probability density functions. In contrast, some methods [39, 40, 41] assume the cost functions are weighted Euclidean distances, and the aim is to determine the weights. Mathematically, they define the cost functions on nodes and edges as an inner product between the weights and the node or edge attributes. Finally, methods in [42, 43] suppose that the graph matching is solved through an approximation of the Graph Edit Distance [44, 45] and only learn the costs of inserting and deleting nodes and edges, which become to be constants.

The method we present deduces the cost functions through a supervised classifier; thus, we consider that it is located in the same nature as the methods from [37, 38, 39, 40, 41]. These methods are similar to our method when they learn application dependent functions. Nevertheless, the main differences are that we propose a general online machine learning model instead of an offline machine learning model. Moreover, our method is the first one that has online, active and interactive features. These features make it new and unique.

Reference [46] presents an active and interactive method applied to graph matching. Nevertheless, the aim is not to learn the cost functions but to assist the graph matching module to deduce a better node-to-node assignment. In that case, the human imposes some node-to-node mappings, as does our method, but they are used only to recompute the whole graph assignment, without learning the cost functions. That method could be used while matching large graphs (for instance, palmprint matching with graphs that have approximately 1000 nodes), in which it is practical for the interactive module to impose just a few correct node-to-node mappings. Our active methods were inspired by the methods presented in that paper, although several modifications have been considered.

3. Definitions and notation

Given a pair of graphs, G and G', the i^{th} vertex in G is represented as G_i and the a^{th} vertex in G' is represented as G'_a. Moreover, the edge between the i^{th} vertex and the j^{th} vertex in G is represented as $G_{i,j}$. Similarly, the edge between the a^{th} vertex and the b^{th} vertex in G' is represented as $G'_{a,b}$.

A correspondence f between graph G and G' is a bijective function that assigns one node of G to only one node of G'. We represent the mapping from node G_i to node G'_a as $a = f(i)$. Note that we suppose that both graphs have the same order. If this was not the case, then they could be expanded with new nodes that had a specific attribute. Then, a generic formulation of the graph matching problem consists of finding the optimal correspondence f^* given by the solution of the quadratic assignment problem (NP-hard problem) [47] (Eq. 1).

$$f^* = \underset{\forall f: G \to G'}{\arg\min} \left\{ \sum_{\forall G_i} \alpha_{i,f(i)} + \sum_{\forall G_{i,j}} \beta_{i,j,f(i),f(j)} \right\} \tag{1}$$

Functions $\alpha_{i,f(i)}$ and $\beta_{i,j,f(i),f(j)}$ represent the cost of mapping a pair of nodes (G_i and $G'_{f(i)}$) and a pair of edges ($G_{i,j}$ and $G'_{f(i),f(j)}$), respectively, and they are application dependent.

With the aim of decreasing the computational cost, some methods reduce

the quadratic assignment into a linear assignment [48, 49, 50, 51, 52], thus, they return a suboptimal correspondence between both graphs (Eq. 2).

$$\hat{f}^* = \underset{\forall f: G \to G'}{\arg\min} \left\{ \sum_{\forall G_i} \gamma_{i,f(i)} \right\} \tag{2}$$

Where $\gamma_{i,f(i)}$ is the cost between local structures centred at nodes G_i and $G'_{f(i)}$. In the literature, one of the most used local structures is called *Star*, which is composed of a central node, its edges and the nodes these edges connect to. In [53], several local structures are analyzed from the point of view of how close the deduced assignment is to the optimal one, and the runtime spent by the matching algorithm.

Two different approaches have been used to define $\gamma_{i,f(i)}$ when the local structures are *Stars*: The structural approach and the vectorial approach. The structural approach is based on assuming a *Star* is a graph [53]. Then, Equation 1 is applied where G and G' are the *Stars* centred at nodes G_i and $G'_{f(i)}$, respectively. In this case, $\gamma_{i,f(i)}$ equals to the cost that generates f^* computed through Equation 1. The vectorial approach converts each *Star* into a vector [54]. Then, $\gamma_{i,f(i)}$ is defined as a distance between vectors. The simplest case is defining it through the Euclidean distance. Nevertheless, it is usual to define it through a learned distance, for instance through a neural network. This is the method we use in this paper and it is explained in Section 4.4.

4. Proposed Framework

In this section, we describe our method, schematically represented in Figure 2. First, we present a general schema of the method (Section 4.1), and then we explain in detail the modules of our schema. The first one searches for the graph matching (Section 4.2), then the second one executes the active learning strategies (Section 4.3) and finally the module that learns the cost functions (Section 4.4).

4.1. General problem schema

The first step of our method is devoted to finding an assignment between a pair of graphs. As we described in Section 3, we work within the linear assignment framework based on Equation 2, solved using the Hungarian method [55]. Therefore, the graph matching step needs the cost function $\gamma_{i,a}$ and the graphs G and G' as inputs for the C cost matrix computation. It provides the cost matrix between all pair of nodes of G and G'. Then, a solver, such as the Hungarian method, is used to provide the (suboptimal) assignment $\hat{f}^*$.

Then, having the cost matrix and the assignment, the second step in our framework is the Active Learning module (Section 4.3), whose goal is to propose a single node-to-node assignment G_i and G'_a that is likely to be the worst one in the complete assignment $\hat{f}^*$. We tried three different active strategies. Then, the human visualizes both nodes with the original data (Section 2.3) and decides to confirm that they map or proposes a new pair of nodes, G_i^* and $G_a'^*$.

The last step is devoted to updating cost function $\gamma_{i,a}$ by adding the new correct node-to-node mapping G_i^* and $G_a'^*$ to the learning module. The first step of this module (detailed in Section 4.4) is to take the node assignment provided by the human and to embed it into a vector, considering the structure and attributes of the original graphs, G and G'. The online learning algorithm updates its internal parameters, considering an input buffer, and provides a new cost function $\gamma_{i,a}$ by using the new data as a new training set (in an incremental manner).

In the first interaction, the pair of nodes proposed to the user, G_i^* and $G_a'^*$, are randomly selected because the cost functions $\gamma_{i,a}$ are not initialized and the cost matrix C cannot be computed yet.

4.2. Graph matching

The graph matching algorithm is composed of two main steps. In the first, a square cost matrix C is computed so that each cell represents a combination $\gamma_{i,a}$. Rows represent nodes G_i and columns represent nodes G'_a, then $C(i,a) = \gamma_{i,a}$. In the second one, the assignment $\hat{f}^*$ is deduced as a solution of a linear

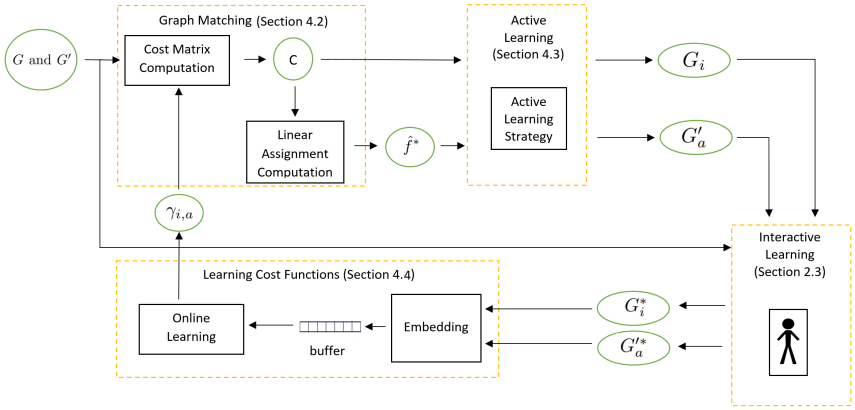


Figure 2: Online Active Learning: detailed framework (see text for explanation).

assignment problem applied to this cost matrix. In the first step, it is necessary to compute the n^2 (n is the size of the graphs) combinations of $\gamma_{i,a}$. Thus, the computational cost of this first step is $O(K \cdot n^2)$, where K is the cost of computing $\gamma_{i,a}$. How K effects the global cost was analyzed in [53]. Note in our schema, how $\gamma_{i,a}$ is defined is a parameter of the *Graph matching* module since it is generated in the *Learning cost functions* module (Section 4.4). The second step is exactly solvable in the worst-case in cubic time, $O(n^3)$, for instance, using the Hungarian method [55]. The fact that solution $\hat{f}^*$ appears to be suboptimal is due to the embedding the quadratic problem into a linear problem (converting Eq. 1 into Eq. 2), but it is not due to the solution of the linear problem itself.

Note that all the learning method is based on deducing the graph matching through a linear algorithm instead of a quadratic algorithm. That is, the graph correspondence is computed through Eq. 2 instead of Eq. 1. This is because the run-time of computing Equation 1 has been empirically demonstrated that does not fulfil the application temporal constraints. In our case, there is a human waiting for the system to compute the graph correspondence. We belief it would be impractical to ask for the human to wait more than several seconds for check-

Table 1: Average runtime and distance given four public databases computed through A^* and graph bipartite algorithm (BP) (from [48]).

				A^*		BP	
	# graphs	Avg size	Avg Degree	Time	Dist	Time	Dist
Alkane	150	8.9	1.8	1.29	15	≈ 0.001	35
Acyclic	183	8.2	1.8	6.02	17	≈ 0.001	35
MAO	68	18.4	2.1	—	—	≈ 0.001	105
PAH	94	20.7	2.4	—	—	≈ 0.001	138

ing the correctness of the correspondence. To validate these reasoning, Table 1 shows a comparison between computing Equation 1 through a A^* algorithm and computing Equation 1 through the bipartite graph matching ([48]). The data in Table 1 has been extracted from [56] and it shows the average run-time of computing the graph edit distance between all the graphs of the database and also the average graph edit distance, given four public databases. Considering the runtime, we realise that there is a huge difference between both algorithms. It is reported in [56] that the computation of these values was not possible in the MAO and PAH databases due to it would take months. Considering the distance, we deduce that the bipartite graph matching returns suboptimal correspondences that have a distance twice the optimal one. We conclude, from this experiment, that using the A* algorithm is not practical since the human would have to wait too much time waiting for the correspondence proposed by the system. Note that better correspondence returned, better system performs but returning the optimal correspondence is not a critical point since it is the user that, through interacting, corrects the correspondence until the system achieves the desired one.

4.3. Active Machine Learning strategies

In this section, we show how we have adapted the active strategies presented in section 2.2 to the graph matching problem. In the strategies we commented,

the active strategy proposes an element given a specific domain, and the oracle returns its class. In our case, the modus operandi has been adapted because we have an active module that proposes to the human only one node-to-node mapping given a pair of graphs and their assignment computed by the graph matching module. Then, the human returns the same node-to-node mapping whether he considers it correct. In contrast, the humans returns the correct one, in which one of the two nodes is the same as the one presented by the active module. In this way, the "element" in the classic active strategy becomes a node-to-node mapping in our method. Because the answerer has to return the "class" of the queried element, in our method, this output has been converted into a node-to-node mapping.

We propose three active strategies. The input parameters are the cost matrix C and the suboptimal assignment $\hat{f}^*$ generated by the graph matching algorithm (Section 3). The output is a pair of nodes G_i and G'_a from each graph. It is assumed that their mapping, although in $\hat{f}^*$, have the highest chance of being wrong, considering the cost matrix C and the node-to-node mapping $\hat{f}^*$. The computational cost of these strategies is quadratic with respect to the number of nodes of the graphs.

Least Confident (LC): This strategy searches for the element for which the classifier has more doubts to deduce its class; in our case, it is the node pair having the highest cost function among the pairs to be selected for node-to-node mapping. Therefore, this strategy is implemented in three steps. First, the best mappings of each node in G are selected according Equation 3.

$$C_1 = \min_{1 \leq a \leq n} \{C(1,a)\}, C_2 = \min_{1 \leq a \leq n} \{C(2,a)\}, \cdots, C_n = \min_{1 \leq a \leq n} \{C(n,a)\} \quad (3)$$

That is, the minimum costs in the cost matrix are selected for every row, and the vector $V = [C_1, C_2, \cdots, C_n]$ is computed. Second, given these mappings, the node in G that has the highest cost (the worst option) is selected; that is, the maximum value is computed given the obtained vector of the minimum values, i.e., $G_i = \arg\max\{V\}$. Third, the node in G' becomes the one to be mapped by the assignation of the selected node in G: $G'_a = \hat{f}^*(G_i)$.

Margin Sampling (MS): The idea behind this strategy is that node pairs with similar cost value are more ambiguous to be selected in the final matching. Thus, in our case, this strategy is implemented in five steps. First, vector V is computed as it is done in the first step of the Least Confident strategy. Second, the same operation is performed, but discarding the nodes from G' found in the previous step. Then, a new vector W is computed. Third, the margins of the nodes in G are computed. That is, the vectors obtained in the first two steps are subtracted, $Z = V - W$. Fourth, the node in G that has the minimum margin is selected, i.e. $G_i = \arg\min\{Z\}$. Fifth, in a similar way to the Least Confident strategy, $G'_a = \hat{f}^*(G_i)$.

Maximum Entropy (ME): The main idea of the method is to query the elements that are more difficult to classify because the Shannon entropy is high. This strategy is similar to the Least Confident strategy except the fact that nodes are selected according to Eq. 4.

$$C_i = -\sum_{1 \leq a \leq n} \{C(i,a) \cdot \log C(i,a)\} \tag{4}$$

Then, $V = [C_1, C_2, \cdots, C_n]$. Note that in this case, the values in the cost matrix have to be in the domain $[0, 1]$. Finally, G_i and $G'_a = \hat{f}^*(G_i)$ are deduced.

4.4. Learning the cost function

In this section, we explain the method we used to learn the cost function $\gamma_{i,a}$. Note that this function depends only on the semantic and structural information of nodes proposed by the human G_i^* and $G_a'^*$, from both graphs (Section 4.1).

This information is embedded into a vector because we want to use a common supervised learning algorithm, such as neural network or a support vector machine. Several embedding methods have been presented in the literature, for instance, there is one in [57] or two more recent proposals in [58, 59]. In our case, we present a simple one defined by the attribute nodes (semantic information) and the number of edges adjacent to the node (structural information) as we show in Figure 3. Others could be analyzed in a future work.

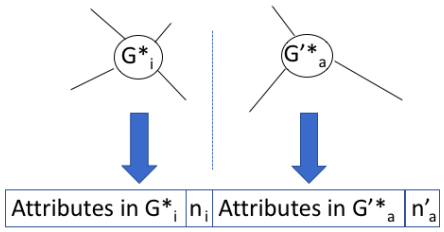


Figure 3: Embedding the local information of nodes G_i^* and $G_a'^*$ into a vector. n_i and n'_a are the number of edges of each node

When the interactive module returns that G_i^* and $G_a'^*$ have to be mapped, we want the cost function $\gamma_{i,a}$ to be zero. Moreover, we want the cost functions $\{\gamma_{i,b}\ ,\ a \neq b\}$ and the cost functions $\{\gamma_{j,a}\ ,\ j \neq i\}$ to be one (or a high value). This is because we want the linear assignment algorithm to return the mappings from G_i^* to $G_a'^*$ and discard the others.

Each time the interactive module proposes a node-to-node mapping (from G_i^* to $G_a'^*$), the supervised learning algorithm, which aims to learn $\gamma_{i,a}$, is fed the data shown in Figure 4. A is the input matrix and B is the expected outcome vector. The order of both graphs is n and n', respectively. Empty rows mean that the data are exactly the same as the upper cells. Rows with dots indicate the information of the next node in the graph. For instance, it represents that node G_j is different from node G_j or G_k. Similarly, it represents that node G'_a is different from node G'_b or G'_c. The repetition of the data (empty rows) is done to solve the unbalancing problems.

The buffer in the learning algorithm, shown in Figure 2, contains the data (matrices A and B) of M impositions. This buffer has been considered to enrich the information that the learning algorithm has to consider.

Note $\gamma_{i,a}$ has been defined as the output of a neural network, being the embedded information of a node and its local structure the input of this neural network. Another completely different paradigm could be to define $\gamma_{i,a}$ as a weighted Euclidean distance between the embedded vectors of nodes G_i and G'_a. In this case, the weights of the Euclidean distance would have to be learned

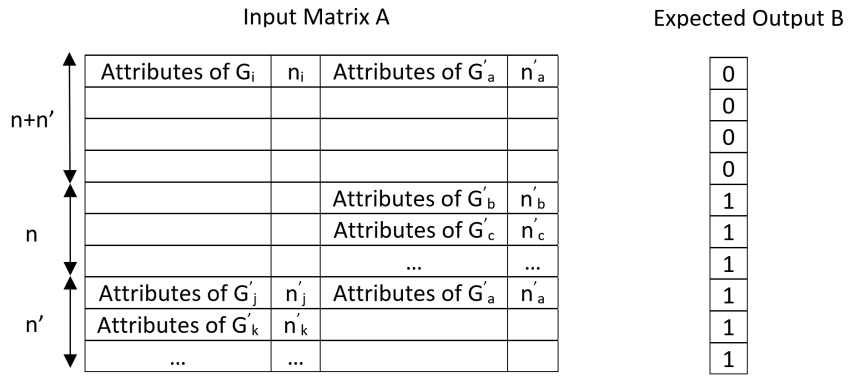


Figure 4: Input matrix A and expected output vector B generated by the imposition that G_i and G'_a have to be mapped. n and n' are the orders of graphs G and G'. Empty rows mean that the data are exactly the same as in the upper cells.

by an optimisation method, such as [31]. This paradigm could be analyzed in a future work.

5. Experiments and discussion

In this section we present the experimental evaluation of our mode. The section is divided as follows. In section 5.1, we provide a general description of the databases used in the experiments. In section 5.2, we describe the setup in which we conducted our experiments. In section 5.3, we show our experimental results. Finally, in section 5.4 we compare our results with the state of the art results.

5.1. Databases

In our experiments, we used five different databases. First, we used the House and Hotel databases, which are described in detail in [60] (Figure 5, left). They consist of two frame sequences corresponding to two different computer modeled objects; one is a House (111 frames) and the second is a Hotel (101 frames). The objects move and rotate through the frames. Each frame of these sequences has 30 identified and mapped salient points attributed by 60 Context

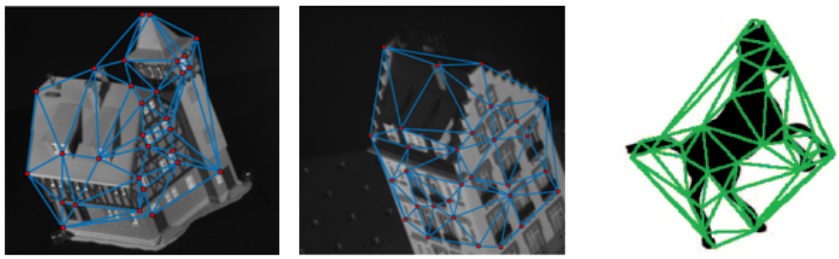


Figure 5: Examples of House-Hotel and Horse original images with salient points and edges.

Shape features. The graphs are built using the salient points as nodes and the edges by triangulating the nodes according to their position using the Delaunay algorithm [61].

The other databases are the Horse (Noise, Rotate and Shear) databases (Figure 5, right), which are based on an original drawing of a horse taken from [62]. The first paper that used these databases for graph-matching was [39]. Each graph has 35 nodes that represent hand-marked salient points representing a horse. Three databases of 199 graphs each (Noise, Shear and Rotate) have been artificially generated by applying the transformations that its name indicates. Each node is attributed by 60 Context Shape features, as in the House and Hotel databases, and the structure was built by triangulating the nodes using the Delaunay algorithm.

In all databases, the more temporally separated the frames are in the sequence, the more different the frames are, which consequently makes the graph-matching process more complex. Moreover, because the nodes are labeled, we can deduce the ground-truth correspondences between nodes of the graphs. In the House and Hotel databases, there are 1314 ground-truth correspondences in the learning set while in the Noise, Rotate and Shear, there are 300 of them.

5.2. Experimental setup

We build the training, validation and testing sets following the same configuration as in [39] and [40]. Thus, we divided each database into three different collections of graph pairs depending on the number of frames of separation for

each experiment: one to train the model, another to validate it, and the last one to test it. Each pair of graphs is separated by the same number of frames in order to keep the level of distortion. The number of frames of separation used in our experiments is 90 frames for the House and Hotel and 100 frames for the Noise, Shear and Rotate in order to compare our results with those previously published in the literature.

On the other hand, to learn and estimate the costs, we have implemented a fitting neural network for parameter regression, with a fully connected hidden layer and the sigmoid as the activation function. The input of network is the embedded representation of a pair of nodes, while the output is the assignment cost between this pair of nodes. We trained our network using the conjugate gradient backpropagation with the Powell-Beale restarts algorithm [63].

To feed the online backpropagation algorithm that trains the network for each interaction, we have implemented a circular buffer.

To determine the parameters of the model (number of neurons and buffer size), we performed some experiments with the validation set until we found the parameters that maximize the performance while minimizing the number of neurons and the buffer size. In addition, in section 5.3.3 we analyze the influence of these parameters on the performance in terms of accuracy and computational cost.

5.3. Results

In this section, we present the results achieved by our method. We split this section into different subsections. In section 5.3.1, we describe the metrics that we use to evaluate the results, while in section 5.3.2 we show the results achieved by our algorithm with respect to the number of interactions. Finally, in section 5.3.3, we show how the parameters configuration of our neural network affects the performance of the model.

5.3.1. Metrics

We used three metrics to analyze and compare the performance of our method:

- The normalized hamming distance between the correspondence deduced by the Bipartite graph matching algorithm (given the learned parameters) and the ground-truth correspondence (given by the database). Because the aim of the learning algorithm is to find the correspondences as similar as possible to the ground-truth, we want this distance to be as low as possible. Eq. 5 express this metric in formula, where $\hat{f}^*$ is the calculated matching, f is the ground truth matching, and N is the number of assignments.

$$NHD = \frac{1}{N}\sum_{i=1}^{N}\delta\left(\hat{f}^*(G_i) \neq f(G_i)\right) \tag{5}$$

- The percentage of node-to-node mappings proposed by the active module to the user that are different from the ground-truth correspondence. The active module only proposes node-to-node mappings that are part of the deduced correspondence (given the current parameters). Thus, we assume that if the active module selects the ones that are different from the ground-truth correspondence and the user corrects them, then the learning algorithm will properly learn the edit costs. Thus, the higher this metric is, the better the performance. In formula:

$$PWM = \frac{\left|\hat{f}^*(G)_{AM} \neq f(G)\right|}{N} \tag{6}$$

where $|\cdot|$ represent the cardinality of a set, $\hat{f}^*$ is the calculated matching, f is the ground truth matching, the subscript AM stands for Active Module and N is the number of assignments.

- The percentage of node-to-node mappings proposed more than once by the active module to the user during the learning phase. We assume that

proposing the same node-to-node mapping several times is less informative than proposing different ones. For this reason, the lower this metric is, the better the performance. Mathematical formula for this metric is shown in Eq. 7 where $\left|\hat{f}^*(G)_{\text{Repeated}}\right|$ is the number of mapping proposed more than onces by the Active module and N_{LP} is the number of node-to-node mappings proposed during the learning phase.

$$PRM = \frac{\left|\hat{f}^*(G)\right|_{\text{Repeated}}}{N_{LP}} \tag{7}$$

5.3.2. Performance evaluation

To perform the experiments in this section, we have implemented a neural network with a hidden layer of 30 neurons for all databases; the buffer has a capacity of 40 mappings for the House and Hotel and 250 for the Noise, Shear and Rotate databases.

Figure 6 and Figure 7 show the previously commented metrics with respect to the number of interactions and using the active learning strategies presented before. In Figure 6, we show the results of the metrics applied to the House and Hotel databases, and in Figure 7, we show the results of the metrics applied to the Noise, Shear and Rotate databases. As described in Figure 2, the usual modus operandi would be that at each interaction, the system proposes a node-to-node mapping to the user according to an active criterion, and the user corrects this mapping, based on whether the user considers it to be wrong. Nevertheless, to make the experimental part completely automatic, the human correction has been simulated by the selection of the node-to-node mapping proposed by the ground-truth correspondence in the database.

Both experiments present similar behaviors. Considering the normalized hamming loss (first row of Figure 6 and Figure 7), we see that our method (given the three active strategies presented before) can reduce the hamming distance to zero at some point (approximately 60 interactions for the House

and Hotel and approximately 120 for the Noise, Shear and Rotate) applying the LC or the MS strategy. Clearly, it is worth using the active strategies because without them (we have called it "Random"), the number of interactions needed to reduce the hamming distance is larger. Note that the reduction of interactions in the online learning methods is usually crucial. In this case, we consider that asking the user to perform 60 or 100 interactions is not a burden.

Considering the number of mappings that the human has to correct (second row of Figure 6 and Figure 7), we observe that the active strategies have a higher percentage than the "Random" strategy at the first interactions, but there is a point at which this relationship is inverted or has a similar value. We consider that having a high percentage of mappings to be corrected by the user is good because it makes the learning algorithm to learn faster. Moreover, it means that when we ask the user to look at the mappings, it is to correct them and influence the system. When the active strategy presents the user with a mapping that does not have to be corrected, the human has to look at the data, but the human's impositions are not used to influence the learning algorithm. Through this reasoning, the three active strategies properly achieve this aim. The "Random" strategy returns higher percentages from interaction 70 to the end in the House database. Note that in this case, the hamming distance is zero at this interaction in the active strategies. Then, it is not possible to improve the performance of the system, which makes the active strategy present mappings that do not have to be corrected.

Finally, considering the number of mappings that are presented to the user more than once (third row of Figure 6 and Figure 7), we realize that the three strategies have much more ability to return different mappings, which is good because they generate a higher chance to need to be corrected by the user. The active strategies present higher numbers of repeated matching than the "Random" strategy at the last interactions because it is much more difficult to increase the performance of the system when the hamming distance is almost zero.

Tables 2 and 3 show the normalized Area Under Curve (AUC) and the

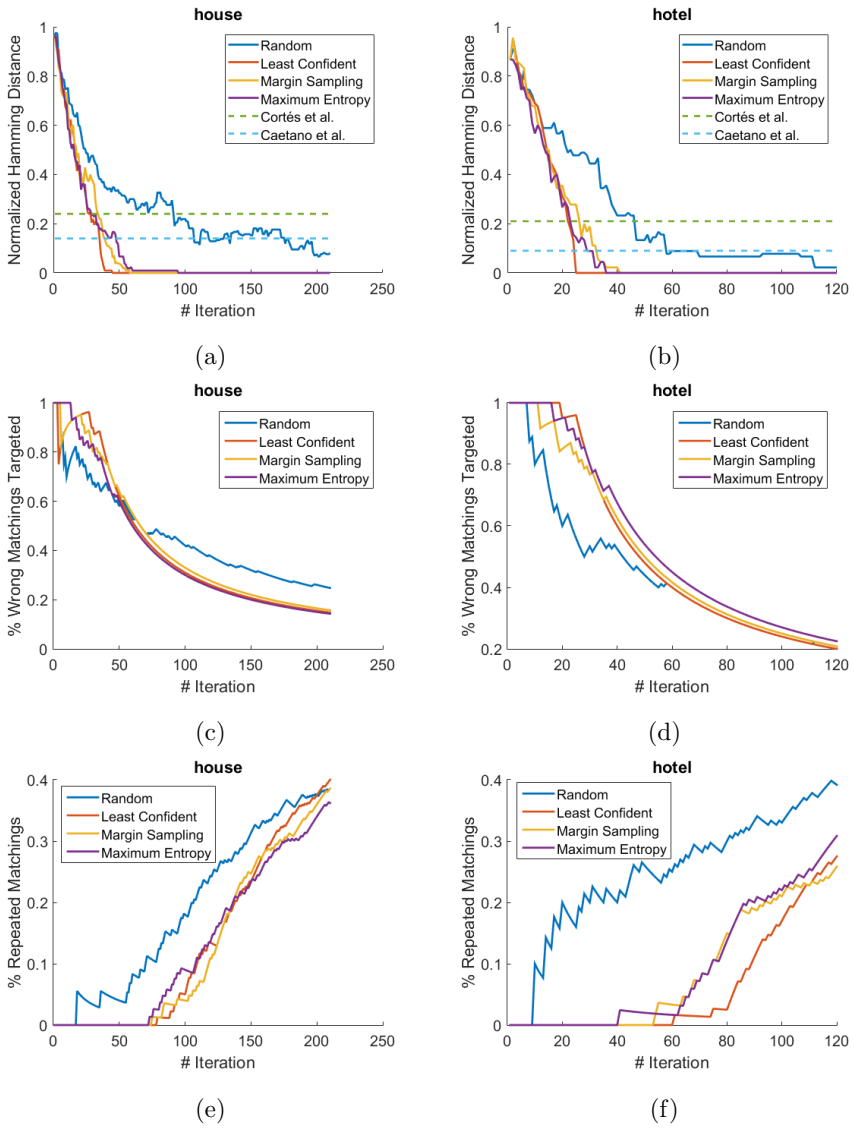


Figure 6: Hamming distance (a) and (b) (the lower the better). Percentage of matchings that need to be corrected deduced by the active algorithm (c) and (d) (the higher the better). Percentage of proposed repeated matchings (e) and (f) (the lower the better). Horizontal axes represent the number of interactions. Databases: House (*left*) and Hotel (*right*). "Random" means the active module selects a random node-to-node mapping from the ones deduced by the graph-matching algorithm.

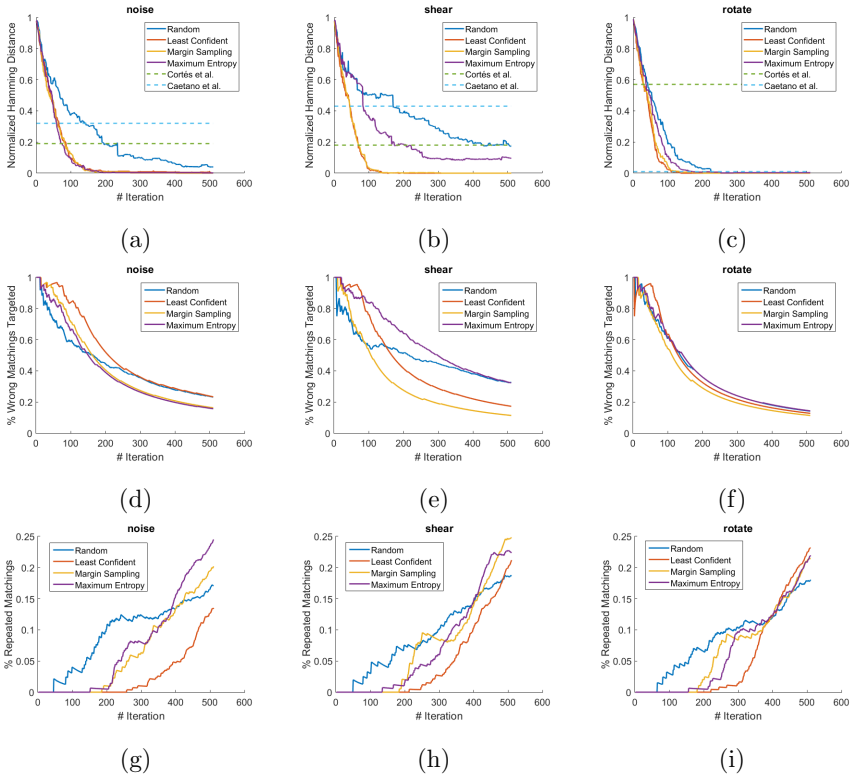


Figure 7: Hamming distance (a),(b) and (c) (the lower the better). Percentage of matchings that need to be corrected deduced by the active algorithm (d),(e) and (f) (the higher the better). Percentage of proposed repeated matchings (g),(h) and (i) (the lower the better). Horizontal axes represent the number of interactions. Databases: Noise (*left*), Shear (*center*) and Rotate (*right*). "Random" means the active module selects a random node-to-node mapping from the ones deduced by the graph-matching algorithm.

Table 2: Comparative table in terms of normalized Area Under Curve (AUC) and runtime per interaction (in seconds) using different active learning strategies in Hotel and House databases. Best results are underlined.

	House		Hotel	
Strategy	AUC	Runtime	AUC	Runtime
Random	0.26	<u>0.78</u>	0.24	0.92
LC	<u>0.08</u>	0.81	<u>0.10</u>	<u>0.88</u>
MS	0.10	0.86	15.02	0.13
ME	0.09	0.87	13.32	0.11

runtime spent to propose a matching to the user according to the different active learning strategies. The normalised AUC is computed as follows. The number of properly deduced mappings in the test set is computed each time the edit costs are recomputed. Then, this number is divided by the number of mappings. Finally, the Normalised AUC is defined as the addition of these values among all the iterations divided by the number of iterations.

Table 2 shows the achieved results in the House and Hotel databases, and Table 3 shows the results in the Noise, Shear and Rotate databases. The aim of the model is to minimize the AUC because we want to minimize the hamming distance with the minimum number of interactions. We observe that it is worth applying active strategies instead of proposing random pairs of nodes for each interaction. The performance between the different active learning strategies is quite similar; however, in the case of Noise, the ME strategy is not able to reduce the hamming distance to 0. We conclude that using an appropriate active learning strategy is crucial, not only because it reduces the number of interactions needed to decrease the hamming distance but also because it avoids falling into local minimums (see hamming distance in the Hotel, the House, the Noise and the Shear databases). We also realize that the runtime spent per interaction is quite similar, independent of the strategy. This is because the system spends much more time solving the graph-matching problem than

Table 3: Table comparing normalized Area Under Curve (AUC) and Runtime per interaction (in seconds) between different active learning strategies with our mode in Noise, Shear and Rotate databases. Best results are underlined.

	Noise		Shear		Rotate	
Strategy	AUC	Runtime	AUC	Runtime	AUC	Runtime
Random	0.23	2.75	0.37	<u>2.74</u>	0.12	<u>2.62</u>
LC	0.10	2.89	<u>0.07</u>	2.82	<u>0.07</u>	2.66
MS	0.10	2.82	0.08	3.11	0.08	2.79
ME	<u>0.09</u>	<u>2.69</u>	0.23	3.13	0.10	2.91

training the model or applying the active learning strategy.

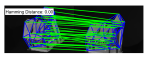
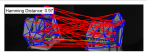
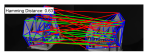
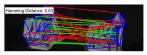
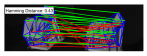
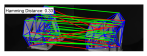
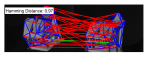
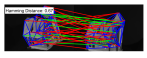
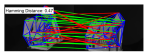
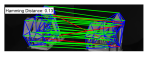
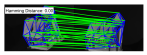
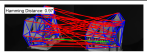
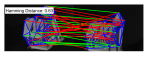
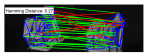
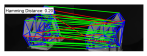
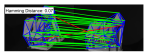
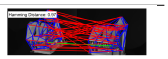
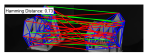
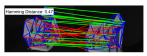
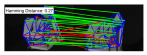
Finally, in Table 4 we show the heat-maps of the cost matrix computed with a pair of graphs of the House database after 1, 10, 20, 30 and 40 interactions using the proposed active strategies and the estimated graph matching. The ground-truth cost matrix (zeros on the diagonal and ones on the other positions) can be deduced because the ground-truth correspondences are provided by the database. We observe how the model tends to perform better when the number of interactions increases. We also observe that the "Random" strategy is the one that seems to work the worst.

5.3.3. Parameter Analysis

There are two main parameters that must be set when we design our model: the buffer size and the number of neurons in the hidden layer of the neural network. In this subsection, we discuss how these parameters effect the performance of the model.

Figure 8 shows the normalized hamming distance with respect to the number of interactions using several buffer sizes. When we increase the buffer size, the hamming distance tends to decrease (the hamming distances given 40 or 80 mappings are superimposed). This means that when we increase the data used to train the network at each interaction, the cost estimations improve until

Table 4: Heatmaps of the cost matrix (yellow: high costs, blue: low costs) and graph matchings (green: correct matchings, red: wrong matchings) using different strategies with respect to the number of interactions in the House dataset and 90 frames of separation.

Strategy	# Interactions	Cost Matrix	Graph Matching
Ground-Truth	-		
Random	1		
	10		
	20		
	30		
	40		
LC	1		
	10		
	20		
	30		
	40		
MS	1		
	10		
	20		
	30		
	40		
ME	1		
	10		
	20		
	30		
	40		

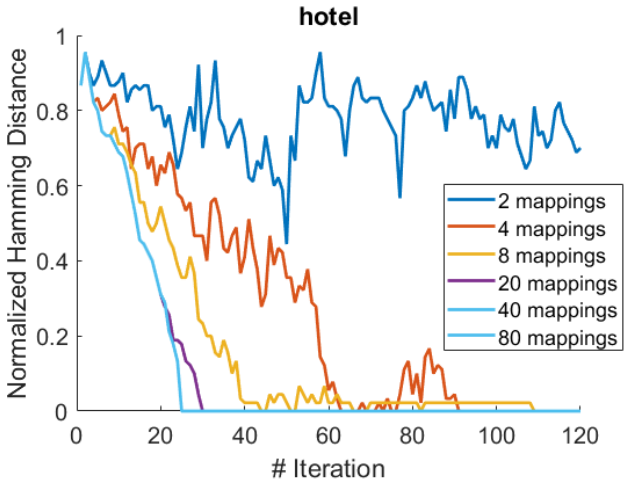


Figure 8: Hamming distance using different buffer sizes using the Least Confident strategy in the Hotel dataset with 90 frames of separation. A neural network with a hidden layer of 30 neurons was used in this experiment. The hamming distances given 40 and 80 mappings obtained the same results.

reaching a point in which there is no more improvement.

Table 5 shows the runtime and the Area Under the Curve (the lower the better) given several buffer sizes. As expected, the runtime spend solving the graph matching problem does not depend on the buffer size, while the learning algorithm depends on the amount of data used to train the model and so the buffer size. This is because when the buffer size increases, the training algorithm needs more interactions to find the weights that minimize the error. Finally, the Area Under the Curve decreases when the buffer size increases, which is congruent on the results shown in Figure 8.

Finally, Figure 9 shows the normalized hamming distance with respect to the number of interactions, while Table 6 shows the normalized Area Under the Curve and the runtime achieved with different neural networks. We have changed the number of neurons of the hidden layer in each run. The input layer has always 122 neurons, $2 * (Attributes + 1) = 2 * (60 + 1) = 122$, and the output layer has only one layer (the value of γ).

Table 5: Runtime per interaction (in seconds) and normalized Area Under the Curve given several buffer sizes in the Hotel dataset using the Least Confident strategy. A neural network with a hidden layer of 30 neurons was used in this experiment because it is the value that obtains the best results in Figure 9. Best results are underlined.

Buffer size	Runtime (graphs matching)	Runtime (learning)	AUC
2	0.50	<u>0.20</u>	0.78
4	0.49	<u>0.20</u>	0.28
8	0.49	<u>0.20</u>	0.17
20	0.49	0.25	0.12
40	<u>0.48</u>	0.32	<u>0.10</u>
80	0.49	0.42	<u>0.10</u>

We observe that with a neural network of 5, 10 and 20 neurons, the model returns a high Area Under the Curve, while if we increase the size to 30 or 40, the Area Under the Curve is significantly reduced. As expected, there are no differences in the graph matching runtime. Moreover, increasing the number of neurons makes the learning runtime increase, but the increase is not significant.

5.4. Comparison to the state-of-the-art

In Table 7, we show a comparative study between our best results applying the proposed active learning strategies and the state-of-the-art off-line methods published in the literature. Our method returns a lower hamming distance than [39, 40] given approximately 35 interactions for the House and Hotel databases and 100 interactions for the Noise, Rotate and Shear databases (see Figure 6 and Figure 7). Note that we are only using the ratio 35/1314 (2.7%) and 100/300 (33%) of the databases, respectively. In general, when we apply some of the proposed strategies, our method reaches the best results. Even when the performance is similar, as in [59], our model has the crucial advantage of using an online learning paradigm. This means that, as described above, it has the capacity to learn in each interaction, significantly minimizing the number

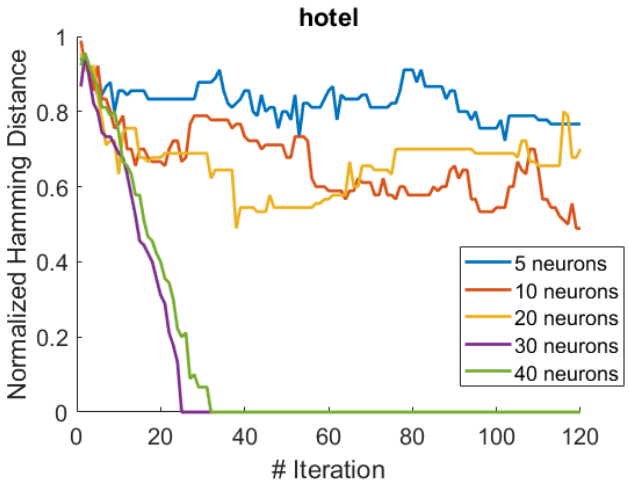


Figure 9: Hamming distance using different numbers of neurons in the hidden layer with the LC strategy in the Hotel dataset with 90 frames of separation.

Table 6: Runtime per interaction (in seconds) and normalized Area Under Curve given several number of neurons. in the Hotel dataset using the Lest Confident strategy. Buffer size of 40 mappings. Best results are underlined.

Number of neurons	Runtime (graphs matching)	Runtime (learning)	AUC
5	0.53	<u>0.33</u>	0.82
10	<u>0.50</u>	0.39	0.65
20	0.60	0.40	0.66
30	0.56	0.35	<u>0.10</u>
40	0.52	0.40	0.13

of node-to-node impositions necessary to obtain the best results. This is very important because manually annotating mappings between nodes of two graphs can be a very costly task.

Table 7: Comparison table of normalized hamming distance results. Best results are underlined.

Algorithm	*Strategy*	Database				
		House	Hotel	Noise	Shear	Rotate
Caetano et. al. [39]	-	0.14	0.09	0.32	<u>0</u>	0.42
Cortés et. al. [40]	-	0.24	0.21	0.19	0.55	0.18
Leordeanu et. al. [41]	-	0.01	0.05	-	-	-
Santacruz et. al. [58]	-	0.02	0.02	-	-	-
Cortés et. al. [59]	-	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>
Our proposal	Random	0.01	0.02	0.04	0.17	<u>0</u>
	LC	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>
	MS	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>
	ME	<u>0</u>	<u>0</u>	0.01	0.08	0.01

6. Conclusions

This paper proposes a model to learn the cost functions for graph matching in an online, active and interactive manner. It has the restriction that the graph matching has to be computed in a suboptimal way and modeled as a linear assignment problem. Moreover, node and edge attributes have to be structured as vectors.

A human (or another intelligent system) is needed to correct the node-to-node mappings that the active model proposes. This proposed mapping is supposed to have a high chance of being wrong. Putting a human in the loop could mean an important increase of runtime. Nevertheless, in some applications in which graphs arrive sequentially, the human effort could make the deduced graph

assignment be close to the one the human desires with only some interactions. In fact, this is what we deduce in the experimental section. We demonstrated that in the given databases, our method achieves higher accuracy than other off-line learning methods with a low number of interactions.

We have proposed three active strategies to present a node-to-node mapping to the human. The active module looks for mappings computed by the graph matching module that have a high chance of being wrong (different from the mapping that the human would decide), as are the ones whose correction could generate the highest impact on the learned cost function. This supposition was demonstrated empirically because when the ratio of selected wrong mapping decreases, the accuracy keeps stable. Thus, when the active module proposes correct mappings, the system does not learn.

In the experimental section, we have analyzed the influence of the three different active strategies and we have seen that the Least Confident one is the strategy with the lowest Area Under the Curve, although there are no significant differences with respect to the Margin Sampling strategy.

As a future work, we want to study the influence of different embeddings of the local information of the nodes. We could analyze the impact of these embeddings with regard to the runtime and the accuracy. Moreover, we could also present another graph matching paradigm in which the function to be optimized was composed of the first and second order terms. Then, the learning module would have to learn different functions. If this module was implemented through a neural network, we could analyze the use of more than one neural network. Finally, we could also apply online learning algorithms other than the neural network, such as online support vector machines.

The overall framework is based on linear assignment and bipartite approximation formulation of graph matching. This obviously limit the accuracy of solution (even if, is still acceptable in several problems). A new direction of research would be to reformulate the whole framework on the basis of quadratic formulation of graph matching. In this case all the learning process must be revised to take into account quadratic term of the formulation.

Acknowledgements

This research is supported by projects TIN2016-77836-C2-1-R and DPI2016-78957-R.

References

References

[1] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, International journal of pattern recognition and artificial intelligence 18 (03) (2004) 265–298.

[2] M. Vento, A long trip in the charming world of graphs for pattern recognition, Pattern Recognition 48 (2) (2015) 291–301.

[3] L. Livi, A. Rizzi, The graph matching problem, Pattern Analysis and Applications 16 (3) (2013) 253–283.

[4] P. Foggia, G. Percannella, M. Vento, Graph matching and learning in pattern recognition in the last 10 years, International Journal of Pattern Recognition and Artificial Intelligence 28 (01) (2014) 1450001.

[5] G. W. Humphreys, C. J. Price, M. J. Riddoch, From objects to names: A cognitive neuroscience approach, Psychological research 62 (2-3) (1999) 118–130.

[6] J. Ward, The student's guide to cognitive neuroscience, Psychology Press, 2015.

[7] P. Zhao, S. C. Hoi, Cost-sensitive online active learning with application to malicious url detection, in: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, 2013, pp. 919–927.

[8] C. Robert, Machine learning, a probabilistic perspective (2014).

[9] F. M. M. M. R Boselli, M Cesarini, Classifying online job advertisements through machine learning, Future Generation Computer Systems 86 (2018) 319–328.

[10] X. Zhang, T. Yang, P. Srinivasan, Online asymmetric active learning with imbalanced data, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 2055–2064.

[11] Y. Yan, Q. Wu, M. Tan, M. K. Ng, H. Min, I. W. Tsang, Online heterogeneous transfer by hedge ensemble of offline and online decisions, IEEE transactions on neural networks and learning systems 29 (7) (2017) 3252–3263.

[12] H. Wu, Y. Yan, Y. Ye, H. Min, M. K. Ng, Q. Wu, Online heterogeneous transfer learning by knowledge transition, ACM Transactions on Intelligent Systems and Technology (TIST) 10 (3) (2019) 1–19.

[13] Y. Zhang, P. Zhao, J. Cao, W. Ma, J. Huang, Q. Wu, M. Tan, Online adaptive asymmetric active learning for budgeted imbalanced data, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 2768–2777.

[14] Y. Zhang, P. Zhao, S. Niu, Q. Wu, J. Cao, J. Huang, M. Tan, Online adaptive asymmetric active learning with limited budgets, IEEE Transactions on Knowledge and Data Engineering.

[15] B. Settles, Active learning, Synthesis Lectures on Artificial Intelligence and Machine Learning 6 (1) (2012) 1–114.

[16] Q. Zhang, S. Sun, Multiple-view multiple-learner active learning, Pattern Recognition 43 (9) (2010) 3113–3119.

[17] S. B. Kotsiantis, I. Zaharakis, P. Pintelas, Supervised machine learning: A review of classification techniques, Emerging artificial intelligence applications in computer engineering 160 (2007) 3–24.

[18] T. M. Mitchell, Machine Learning, 1st Edition, McGraw-Hill, Inc., New York, NY, USA, 1997.

[19] A. Sanchis, A. Juan, E. Vidal, A word-based naïve bayes classifier for confidence estimation in speech recognition, IEEE Transactions on Audio, Speech, and Language Processing 20 (2) (2012) 565–574.

[20] P. H. Gosselin, F. Precioso, S. Philipp-Foliguet, Incremental kernel learning for active image retrieval without global dictionaries, Pattern Recognition 44 (10-11) (2011) 2244–2254.

[21] A. Baranes, P.-Y. Oudeyer, Active learning of inverse models with intrinsically motivated goal exploration in robots, Robotics and Autonomous Systems 61 (1) (2013) 49–73.

[22] V. Romero, A. H. Toselli, E. Vidal, Multimodal interactive handwritten text transcription, Vol. 80, World Scientific, 2012.

[23] R. Wang, S. Kwong, D. Chen, Inconsistency-based active learning for support vector machines, Pattern Recognition 45 (10) (2012) 3751–3767.

[24] D. Gorisse, M. Cord, F. Precioso, Salsas: Sub-linear active learning strategy with approximate k-nn search, Pattern Recognition 44 (10-11) (2011) 2343–2357.

[25] E. Lughofer, Hybrid active learning for reducing the annotation effort of operators in classification systems, Pattern Recognition 45 (2) (2012) 884–896.

[26] A. T. Da Silva, A. X. Falcão, L. P. Magalhães, Active learning paradigms for cbir systems based on optimum-path forest classification, Pattern Recognition 44 (12) (2011) 2971–2978.

[27] B. Settles, M. Craven, An analysis of active learning strategies for sequence labeling tasks, in: Proceedings of the conference on empirical methods in

natural language processing, Association for Computational Linguistics, 2008, pp. 1070–1079.

[28] A. Culotta, A. McCallum, Reducing labeling effort for structured prediction tasks, in: AAAI, Vol. 5, 2005, pp. 746–751.

[29] R. Hwa, Sample selection for statistical parsing, Computational linguistics 30 (3) (2004) 253–276.

[30] C. E. Shannon, A mathematical theory of communication, ACM SIGMOBILE Mobile Computing and Communications Review 5 (1) (2001) 3–55.

[31] P. Zhao, Y. Zhang, M. Wu, S. C. H. Hoi, M. Tan, J. Huang, Adaptive cost-sensitive online classification, IEEE Transactions on Knowledge and Data Engineering 31 (2) (2019) 214–228.

[32] M. Zinkevich, Online convex programming and generalized infinitesimal gradient ascent, in: Proceedings of the 20th international conference on machine learning (icml-03), 2003, pp. 928–936.

[33] X. Cortés, F. Serratosa, An interactive method for the image alignment problem based on partially supervised correspondence, Expert Systems With Applications 42 (1) (2015) 179–192.

[34] X. Cortés, F. Serratosa, Cooperative pose estimation of a fleet of robots based on interactive points alignment, Expert Systems With Applications 45 (2016) 150–160.

[35] G. Manzo, F. Serratosa, M. Vento, Online human assisted and cooperative pose estimation of 2d cameras, Expert systems with applications 60 (2016) 258–268.

[36] E. Rica, C. F. Moreno-García, S. Álvarez, F. Serratosa, Reducing human effort in engineering drawing validation, Computers in Industry 117 (2020) 103198.

[37] M. Neuhaus, H. Bunke, Self-organizing maps for learning the edit costs in graph matching, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 35 (3) (2005) 503–514.

[38] M. Neuhaus, H. Bunke, Automatic learning of cost functions for graph edit distance, Information Sciences 177 (1) (2007) 239–247.

[39] T. S. Caetano, J. J. McAuley, L. Cheng, Q. V. Le, A. J. Smola, Learning graph matching, IEEE transactions on pattern analysis and machine intelligence 31 (6) (2009) 1048–1058.

[40] X. Cortés, F. Serratosa, Learning graph matching substitution weights based on the ground truth node correspondence, International Journal of Pattern Recognition and Artificial Intelligence 30 (02) (2016) 1650005.

[41] M. Leordeanu, R. Sukthankar, M. Hebert, Unsupervised learning for graph matching, International journal of computer vision 96 (1) (2012) 28–45.

[42] X. Cortés, F. Serratosa, Learning graph-matching edit-costs based on the optimality of the oracle's node correspondences, Pattern Recognition Letters 56 (2015) 22–29.

[43] S. Algabli, F. Serratosa, Embedding the node-to-node mappings to learn the graph edit distance parameters, Pattern Recognition Letters 112 (2018) 353–360. doi:10.1016/j.patrec.2018.08.026.
URL https://doi.org/10.1016/j.patrec.2018.08.026

[44] K. Riesen, Structural pattern recognition with graph edit distance, Springer, 2015.

[45] F. Serratosa, Graph edit distance: Restrictions to be a metric, Pattern Recognition 90 (2018) 250–256.

[46] F. Serratosa, X. Cortés, Interactive graph-matching using active query strategies, Pattern Recognition 48 (4) (2015) 1364–1373.

[47] K. M. Anstreicher, Recent advances in the solution of quadratic assignment problems, Mathematical Programming 97 (1-2) (2003) 27–42.

[48] K. Riesen, H. Bunke, Approximate graph edit distance computation by means of bipartite graph matching, Image and Vision computing 27 (7) (2009) 950–959.

[49] F. Serratosa, Fast computation of bipartite graph matching, Pattern Recognition Letters 45 (2014) 244–250.

[50] P. Santacruz, F. Serratosa, Error-tolerant graph matching in linear computational cost using an initial small partial matching, Pattern Recognition Letters (2018) in press.

[51] F. Serratosa, Speeding up fast bipartite graph matching through a new cost matrix, International Journal of Pattern Recognition and Artificial Intelligence 29 (02) (2015) 1550010.

[52] F. Serratosa, Computation of graph edit distance: Reasoning about optimality and speed-up, Image and Vision Computing 40 (2015) 38–48.

[53] F. Serratosa, X. Cortés, Graph edit distance: Moving from global to local structure to solve the graph-matching problem, Pattern Recognition Letters 65 (2015) 204–210.

[54] P. Santacruz, F. Serratosa, Learning the graph edit costs based on a learning model applied to sub-optimal graph matching, Neural Processing Letters 51 (1) (2020) 881–904.

[55] J. Munkres, Algorithms for the assignment and transportation problems, Journal of the Society of Industrial and Applied Mathematics 5 (1) (1957) 32–38.

[56] S. Bougleux, L. Brun, V. Carletti, P. Foggia, B. Gaüzère, M. Vento, Graph edit distance as a quadratic assignment problem, Pattern Recognit. Lett.

87 (2017) 38–46. doi:10.1016/j.patrec.2016.10.001
URL https://doi.org/10.1016/j.patrec.2016.10.001

[57] M. M. Luqman, J.-Y. Ramel, J. Lladós, T. Brouard, Fuzzy multilevel graph embedding, Pattern Recogn. 46 (2) (2013) 551–565.

[58] P. Santacruz, F. Serratosa, Learning the sub-optimal graph edit distance edit costs based on an embedded model, in: Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), Springer, 2018, pp. 282–292.

[59] X. Cortés, D. Conte, H. Cardot, Learning edit cost estimation models for graph edit distance, Pattern Recognition Letters 125 (2019) 256 – 263.

[60] C. F. Moreno-García, X. Cortés, F. Serratosa, A graph repository for learning error-tolerant graph matching, in: Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), Springer, 2016, pp. 519–529.

[61] J. R. Shewchuk, Delaunay refinement algorithms for triangular mesh generation, Computational geometry 22 (1-3) (2002) 21–74.

[62] A. M. Bronstein, M. M. Bronstein, A. M. Bruckstein, R. Kimmel, Analysis of two-dimensional non-rigid shapes, International Journal of Computer Vision 78 (1) (2008) 67–88.

[63] M. J. D. Powell, Restart procedures for the conjugate gradient method, Mathematical Programming 12 (1977) 241–254.