Learning Graphs to Match
Minsu Cho, Karteek Alahari, Jean Ponce

To cite this version:
Minsu Cho, Karteek Alahari, Jean Ponce. Learning Graphs to Match. ICCV - IEEE International Conference on Computer Vision, Dec 2013, Sydney, Australia. pp.25-32, 10.1109/ICCV.2013.11 . hal-00875105

HAL Id: hal-00875105
https://inria.hal.science/hal-00875105
Submitted on 21 Oct 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Abstract

Many tasks in computer vision are formulated as graph matching problems. Despite the NP-hard nature of the problem, fast and accurate approximations have led to significant progress in a wide range of applications. Learning graph models from observed data, however, still remains a challenging issue. This paper presents an effective scheme to parameterize a graph model, and learn its structural attributes for visual object matching. For this, we propose a graph representation with histogram-based attributes, and optimize them to increase the matching accuracy. Experimental evaluations on synthetic and real image datasets demonstrate the effectiveness of our approach, and show significant improvement in matching accuracy over graphs with pre-defined structures.

1. Introduction

Graphs are widely used as a general and powerful representation in a variety of scientific fields, including computer vision, and many problems can be formulated as attributed graph matching. Since graph matching is mathematically expressed as a quadratic assignment problem, which is NP-hard, most research has long focused on developing accurate and efficient approximate algorithms [8, 14, 32]. Much progress has been achieved recently in various applications of graph matching, such as shape analysis [27], image matching [12, 30], action recognition [33], and object categorization [3, 13].

For many tasks, however, a natural question arises: How can we obtain a good graph model for a target object to match? Recent studies have revealed that simple graphs with hand-crafted structures and similarity functions, typically used in graph matching, are insufficient to capture the inherent structure underlying the problem at hand. As a consequence, a better optimization of the graph matching objective does not guarantee better correspondence accuracy [5, 6]. Previous learning methods for graph matching tackle this issue by learning a set of parameters in the objective function [5, 21, 26, 30]. Although it is useful to learn such a matching function for two graphs of a certain class, a more apt goal would be to learn a graph model to match, which provides an optimal matching to all instances of the class. Such a learned graph would better model the inherent structure in the target class, thus resulting in better performance for matching.

In this paper, we propose to learn a graph model based on a particular, yet rather general, graph representation with histogram-based attributes for nodes and edges. To this end, we present a generalized formulation for graph matching, which is an extension of previous learning approaches (Sec. 2). We show that all attributes of the graph can be learned in a max-margin framework [31] (Sec. 3). The proposed method reconstructs a graph model inherent in a target class, and provides impressive matching performance, as demonstrated in our experiments on synthetic and real datasets (Sec. 4). Figure 1 illustrates the effectiveness of the learning approach presented in the rest of this paper on an example. Here, the learned graph model finds better correspondences than a graph with a learned matching function as well as a hand-crafted graph.
The problem of learning graphical models has been addressed in other contexts. Several methods learn the parameters in clique functions defined on Markov random fields [25, 28]. Other works learn the connectivity structure of the Markov networks by introducing sparsity on clique functions [11, 17]. In the context of certain graph matching applications, an iterative method that alternates between estimating parameters and punning some of the nodes and edges has been proposed [4, 19]. Our approach differs from these methods in the sense that it learns attributes for all nodes and edges in a max-margin framework, not limited to global parameters in clique functions or sparse selection of clique functions. The learned attributes, combined with weight parameters, turn out to be critical for matching.

2. Graph matching revisited

We begin by reviewing the standard graph matching formulation and elaborate on methods to learn its parameters. In this context, we generalize the standard formulation and also highlight related work.

2.1. Problem formulation

The objective of graph matching is to find correspondences between two attributed graphs \( G = (V, E, A) \) and \( G' = (V', E', A') \), where \( V \) represents a set of nodes, \( E \) a set of edges, and \( A \) a set of attributes of the nodes and edges. A solution of graph matching is defined as a subset of possible correspondences \( \mathcal{V} \subset V \times V' \), represented by a binary assignment matrix \( Y \in \{0, 1\}^{n \times n'} \), where \( n \) and \( n' \) denote the number of nodes in \( G \) and \( G' \), respectively. If \( \pi(i) \in V \) matches \( \pi(i') \in V' \), then \( Y_{i,a} = 1 \), and \( Y_{i,a} = 0 \) otherwise. We denote by \( y \in \{0, 1\}^{nn'} \), a column-wise vectorized replica of \( Y \). With this notation, graph matching problems can be expressed as finding the assignment vector \( y^* \) that maximizes a score function \( S(G, G', y) \) as follows:

\[
y^* = \arg \max_y S(G, G', y), \tag{1a}
\]

\[
s.t. \left\{ \begin{array}{l}
y \in \{0, 1\}^{nn'} \\
\sum_{i=1}^{n} y_{ia} \leq 1, \\
\sum_{a=1}^{n'} y_{ia} \leq 1,
\end{array} \right. \tag{1b}
\]

where Eq. (1b) induces the matching constraints, thus making \( y \) an assignment vector [6, 9, 14, 18].

The score function \( S(G, G', y) \) measures the similarity of graph attributes, and is typically decomposed into a first-order similarity function \( s_v(a_i, a'_i) \) for a node pair \( v_i \) in \( V \) and \( v'_i \) in \( V' \), and a second-order similarity function \( s_E(a_{ij}, a'_{ij}) \) for an edge pair \( e_{ij} \) in \( E \) and \( e'_{ij} \) in \( E' \). Similarity functions are usually represented by a symmetric similarity (or affinity) matrix \( A \), where a non-diagonal element \( A_{ij} = s_v(a_i, a'_i) \) contains the edge similarity of two correspondences \( (v_i, v'_i) \) and \( (v_j, v'_j) \), and a diagonal term \( A_{ii} = s_v(a_i, a'_i) \) represents the node similarity of a correspondence \( (v_i, v'_i) \). Thus, the score function of graph matching is defined as:

\[
S(G, G', y) = \sum_{y_{ia}=1} s_v(a_i, a'_i) + \sum_{y_{ia}=1} s_E(a_{ij}, a'_{ij})
\]

\[
= y^T A y. \tag{2}
\]

In essence, the score accumulates all the similarity values relevant to the assignment.

The formulation in Eq. (1) is referred to as an integer quadratic programming. More precisely, it is the quadratic assignment problem, which is known to be NP-hard. Due to its generality and flexibility, this formulation has been favored in recent graph matching research. Many efficient approximate algorithms have been proposed for the formulation [6, 9, 14, 20, 30] and its extensions [7, 12].

2.2. Learning parameters

In the context of scoring functions defined in Eq. (2), an interesting question is what can be learned to improve graph matching. To address this, we parameterize Eq. (2) as follows. Let \( \pi(i) = a \) denote an assignment of node \( v_i \) in \( G \) to node \( v'_a \) in \( G' \), i.e. \( y_{ia} = 1 \). A joint feature map \( \Phi(G, G', y) \) is defined by aligning the relevant similarity values of Eq. (2) into a vectorial form as:

\[
\Phi(G, G', y) = [\cdots; s_v(a_i, a'_i(\pi(i))); \cdots; s_E(a_{ij}, a'_{ij}(\pi(j))): \cdots].
\]

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

\[
S(G, G', y; \beta) = \beta \cdot \Phi(G, G', y), \tag{4}
\]

where \( \beta \) is a weight vector encoding the importance of node and edge similarity values. In the case of uniform weights, i.e. \( \beta = 1 \), it reduces to the conventional graph matching score function of Eq. (2): \( S(G, G', y) = S(G, G', y; 1) \). We refer to it as the discriminative weight formulation for graph matching. Note that the similarity functions \( s_v \) and \( s_E \) can take not only scalar-valued, but also vector-valued functions. Despite its apparent simplicity, this formulation covers a wide range of parameter learning approaches proposed for graph matching. They are equivalent to special cases of this formulation, which restrict \( \beta \) and assign the same vector for all \( s_v(a_i, a'_i) \) and \( s_E(a_{ij}, a'_{ij}) \). Caetano et al. [5] use a 60-dimensional similarity function \( s_v \) for appearance similarity and a simple binary similarity \( s_E \) for edges. Leordeanu et al. [21] do not use \( s_v \), and instead employ a multi-dimensional \( s_E \) for similarity of appearance, angle, and distance. The work of Torresani et al. [30] can be viewed as adopting 2-dimensional \( s_v \) and \( s_E \) functions for measuring appearance similarity, geometric compatibility, and occlusion likelihood.
different, all of them are essentially aimed at learning common weights for all the edge and node similarity functions. The discriminative weight formulation is more general in the sense that it can assign different parameters for individual nodes and edges. We will highlight this advantage with appropriate evidence in the experimental section. However, like previous approaches, it does not learn a graph model underlying the feature map Φ, and requires a reference graph G* at query time, whose attributes cannot be modified in the learning phase. We overcome this drawback by proposing an optimization framework to learn the reference graph.

3. Graph learning

To address the problem of learning graphs, we start with the discriminative weight formulation of Eq. (4). Instead of a reference graph used in the previous section, we consider a class-specific model graph G*. Our aim is to infer this graph, such that it produces the best matches with other instances of the class. Let y denote the optimal matching between the model graph G* and an input graph G, given by:

\[
\hat{y}(G; G^*, \beta) = \arg \max_{y \in \mathcal{Y}(G)} S(G^*, G, y; \beta),
\]

where β is a weight vector, \(\mathcal{Y}(G)\) defines the set of possible assignment vectors for the input graph G. Inspired by the max-margin framework [31], we learn the model graph G* and its weights β from labeled examples \(D = \{\langle G_1, y_1 \rangle, \ldots, \langle G_n, y_n \rangle\}\), where \(y_i \in \mathcal{Y}(G_i)\), by minimizing the following objective function:

\[
L_D(G^*, \beta) = r(G^*, \beta) + \frac{C}{n} \sum_{i=1}^{n} \Delta(y_i, \hat{y}(G_i; G^*, \beta)),
\]

In this objective function \(r\) is a regularization function, \(\Delta\) a loss function, and \(y_i\) denotes the ground truth assignment vector. The parameter C controls the relative importance of the loss term.

A critical question to address here is how to parametrize G* and β such that the optimization of Eq. (6) is feasible. We propose to parametrize both G* and β in a vectorial form, which then enables us to optimize the objective function in Eq. (6) efficiently, as shown below. We first separate the graph model G* from the joint feature map \(\Phi(G^*, G, y)\), so as to parameterize it separately. We assume that the similarity functions \(s_V\) and \(s_E\) are dot products of two attribute vectors:

\[
s_V(a^*_i, a_i) = a^*_i \cdot a_i, \quad s_E(a^*_{ij}, a_{ab}) = a^*_{ij} \cdot a_{ab},
\]

where \(a^*_i\) and \(a^*_{ij}\) correspond to the node and edge attributes of the model graph respectively. Further, we define the attribute vector \(\Theta(G^*)\) and the feature map \(\Psi(G, y)\) as:

\[
\Theta(G^*) = \begin{bmatrix} \cdots \mid a^*_i \mid \cdots \mid a^*_{ij} \cdots \end{bmatrix},
\]

\[
\Psi(G, y) = \begin{bmatrix} \cdots \mid a_{\pi(i)} \mid \cdots \mid a_{\pi(i)\pi(j)} \cdots \end{bmatrix},
\]

where \(\Theta(G^*)\) describes all the attributes of \(G^*\) and \(\Psi(G, y)\) represents the corresponding attributes of \(G\), according to the assignment \(y\). This enables the attributes of \(\Phi(G^*, G, y)\) to be factorized into \(\Theta(G^*)\) and \(\Psi(G, y)\), and thus to rewrite the score function as:

\[
S(G^*, G, y; \beta) = \beta \cdot \Phi(G^*, G, y) = \beta \cdot (\Theta(G^*) \odot \Psi(G, y)) = (\beta \odot \Theta(G^*)) \cdot \Psi(G, y),
\]

where \(\odot\) denotes the Hadamard (element-wise) product. Note that both terms of the attributes \(\Theta(G^*)\) and their weights \(\beta\) are now combined into a single vector \((\beta \odot \Theta(G^*))\). Thus, when the similarity functions are dot products, both the graph model attributes and their weights can be jointly expressed by a single vector. By substituting \(w = \beta \odot \Theta(G^*)\) into Eq. (5), we obtain a linear form for the optimal assignment:

\[
\hat{y}(G; w) = \arg \max_{y \in \mathcal{Y}(G)} w \cdot \Psi(G, y).
\]

In turn, this transforms the learning objective in Eq. (6) into a standard formulation of the structured support vector machine (SSVM):

\[
L_D(w) = \frac{1}{2} ||w||^2 + \frac{C}{n} \sum_{i=1}^{n} \Delta(y_i, \hat{y}(G_i; w)),
\]

where all the graph model attributes and their weights, to be learned, are represented by \(w\). This function can be minimized by various optimization approaches to estimate the parameters \(w\) [16, 29, 31]. Unlike other learning methods for graph matching [5, 21, 26, 30], this formulation allows us to combine graph learning, and learning a matching function into a coherent structured output framework. A related approach has been proposed to learn homography estimation in keypoint matching and tracking [15].

3.1. Histogram-attributed relational graph

In general, any graph representation satisfying the condition of dot product similarity of Eq. (7), which leads to the linearization in Eq. (10), can be learned with our approach. However, not all potential representations are effective in representing the data in the context of graph learning and matching performance. In this work, we propose a new histogram-attributed relational graph (HARG), wherein all node and edge attributes are represented by histogram distributions. The similarity value between two attributes in this graph is then computed as their dot product. The histogram attributes in this framework can be composed of a variety of features. In this work we chose to build them using angle and length for edge attributes, and local appearance for node attributes.

The histogram of log-polar bins edge attribute describes the geometric relationship between two interest points as
illustrated in Fig. 2. As widely done in computer vision [22, 23], we assume that each interest point can be assigned a characteristic scale and orientation. ¹ Consider an edge \( e_{ij} \) from node \( v_i \) (represented by point \( x_i \) in Fig. 2) to node \( v_j \) (\( x_j \) in the figure). The vector from \( x_i \) to \( x_j \) can be expressed in polar coordinates as \( (\rho_{ij}, \theta_{ij}) \). We transform this into a histogram-based attribute, which is invariant to the characteristic scale and orientation of \( v_i \). Two histograms — one for length and another for angle — are built and concatenated to quantize the edge vectors. For length, we use uniform bins of size \( n_L \) in the log space with respect to the position and scale of \( v_i \), making the histogram more sensitive to the position of nearby points. The log-distance histogram \( L_{ij} \) is constructed on the bins by a discrete Gaussian histogram centered on the bin for \( \rho_{ij} \):

\[
L_{ij}(k) = f_L(k-m),
\]

s.t. \( f_L(x) = N(0, \sigma_L), \rho_{ij} \in \text{bin}_{\rho}(m), \)

where \( N(\mu, \sigma) \) represents a discrete Gaussian window² of size \( \sigma \) centered on \( \mu \), and bin\(_{\rho}(m) \) denotes the \( m \)th log-distance bin from the center of \( v_i \). For angle, we use uniform bins of size \( 2\pi/n_P \). The polar-angle histogram \( P_{ij} \) is constructed on it in a similar way, except that a circular Gaussian histogram centered on the bin for \( \theta_{ij} \) with respect to the characteristic orientation of \( v_i \), is used:

\[
P_{ij}(k) = f_P(k-m),
\]

s.t. \( f_P(x) = N(0, \sigma_P) + N(\pm n_P, \sigma_P), \theta_{ij} \in \text{bin}_{\theta}(m), \)

where additional Gaussian terms in \( f_P(x) \) induce the circular bins for angle. The final histogram composed by concatenating the log-distance \( L_{ij} \), and the polar-angle \( P_{ij} \), histograms is defined as the attribute for edge \( e_{ij} \): \( a_{ij} = [L_{ij}; P_{ij}] \), which is asymmetric \( (a_{ij} \neq a_{ji}) \).³ In this work, we used \( n_L = 9, n_P = 18 \). Our representation has several advantages for visual matching. When used with local invariant features, it becomes geometrically invariant in scale and orientation. From the viewpoint of learning, notably, the nonparametric nature of histograms allows us to represent multi-modal distributions of distance and angle through the learning process.

For node attributes \( a_t \), describing the local appearance of node \( v_t \), we could adopt the histogram of gradient bins such as SIFT [22], HOG [10], and their variants, given their effectiveness. In our experiments, we used the SIFT descriptor.

¹When the interest points do not have characteristic scales and orientations, we fix them to 1 and 0 respectively.
²We used a window size \( \sigma_L = \sigma_P = 5 \) so that \( N(0, 5) = 1.0, N(\pm 1, 5) = 0.4578, N(\pm 2, 5) = 0.0439 \), and 0 otherwise.
³In the context of feature descriptors, shape-context [2] uses a histogram to represent the distribution of points in a two-dimensional log-polar space. In contrast, our histogram for edge attributes consists of two separate log-distance and polar-angle ones.

Figure 2: Histogram of log-polar bins for edge attributes. This histogram is represented by a discrete Gaussian window centered at a bin. (a) Log-distance \( \rho_{ij} \) (left) and its histogram with 9 bins (right). The log-distance \( \rho_{ij} \) of edge \( e_{ij} \) is measured relative to the scale of \( v_i \). (b) Polar-angle \( \theta_{ij} \) (left) and its histogram with 18 bins (right). The polar-angle \( \theta_{ij} \) of edge \( e_{ij} \) is measured from the characteristic orientation of \( v_i \), or from the horizontal line through \( v_i \) (shown as a green line), when there is no such orientation.

3.2. Loss functions

Another ingredient in the objective function Eq. (12) is the loss function \( \Delta(y, \hat{y}) \). It drives the learning process by measuring the quality of a predicted matching \( \hat{y} \) against its ground truth \( y \). We use the normalized Hamming loss, similar to [5], which is the fraction of mismatches between assignment vectors \( y \) and \( \hat{y} \),

\[
\Delta(y, \hat{y}) = 1 - \frac{1}{||y||^2_F} y \cdot \hat{y},
\]

where \( || \cdot ||_F \) is the Frobenius norm.

3.3. Optimization

Many approaches have been proposed to train SSVMs [16, 29, 31]. This problem amounts to solving a convex quadratic program with an exponentially large
number of constraints. Solutions for this optimization problem either: (i) reduce it to an equivalent polynomial-size reformulation (for certain decomposable loss functions), and use methods like SMO [29] or general-purpose solvers; or (ii) work with the original problem by considering a subset of constraints, and employing cutting plane [31] or stochastic subgradient methods. For solving the problem in Eq. (12), we use the efficient cutting plane method proposed by Joachims et al. [16]. This method differs from most other SVM training approaches by considering individual data points as well as their linear combinations as potential support vectors. This leads to a smaller set of cutting plane models, and thus more efficient training.

4. Experimental evaluation

In this section we conduct comparative evaluations on synthetic and real data. We observed that our histogram-based similarity function showed better or comparable matching performance than other similarity measures used in [6, 9, 21]. Hence, we chose to focus on the performance of learning using these attributes in all our experiments. A fully-connected graph is used as the initial graph for learning. We evaluate four methods: ‘w/o learning’, ‘SW-SSVM’, ‘SW-SPEC’, ‘DW-SSVM’, and our method ‘HARG-SSVM’. For w/o learning, we use a conventional graph matching method with uniform weighting. For SW-SSVM, SSVM learning is applied to shared weights on nodes and edges. Here, we learn 2 (angle and distance) parameters for edges, and 128 (SIFT) parameters for nodes. Although the similarity functions and the optimization algorithm are different, SW-SSVM is closely related to the method of [5]. SW-SPEC is the learning method of [21] for shared weights. DW-SSVM represents a discriminative weight learning approach based on the formulation discussed in Sec. 2.2, which learns individual weights for nodes and edges (2 parameters for each edge and 128 parameters for each node). HARG-SSVM is our graph learning approach proposed in Sec. 3. The SSVM objective is optimized with the same method [16] in all the experiments.

It should be noted that the methods [5, 21] were originally proposed to learn the weights of a graph matching function for two graphs in the same class. Our approach (HARG-SSVM), on the other hand, learns the graph model as well. The approaches are evaluated on three datasets, including a synthetic dataset, the CMU House/Hotel sequences, and an object class dataset.

4.1. Synthetic point sets

The goal of this experiment is to evaluate and compare the performance in a controlled setting. Here, we build on the widely used point set matching problem protocol [9, 18]. We define a source set \( P \) by \( n \) Gaussian distributions, each of which is centered at a random point in the 2-dimensional domain \([0, 1]^2\), with a random variance in \([0, 0.15]\). As shown in Fig. 3(a), an observable sample from the source set \( P \) consists of \( n \) inlier points, which come from \( N(p_i, \sigma_i) \), and \( n_o \) random outliers, generated from a uniform distribution in \([0, 1]^2\). Visually, this simulates deformation and clutter in the observations. We consider 100 sample point sets from the source set, where each point in the sample set has an assignment label to one of the \( n \) distributions in \( P \). Our task is to assign labels to new samples with graph matching. This problem setting resembles many real-world applications, and is equivalent to experiments in previous works [5, 21]. Since there is no unary information in the points, graph matching in this case relies solely on pairwise similarity. From each point set we construct a graph with our histogram-based attributes.
Table 1: Performance on synthetic point sets. Several learning approaches (shown in each row) are evaluated with the state-of-the-art graph matching algorithms (in columns). See text for details of the learning methods. ‘Sample’ or ‘Source’ refer to the type of reference graphs used for learning and matching. Given that in real problems it is unlikely to get observations without deformations and noise, the result with ‘Source’ corresponds to the upper bound for the methods we compare to. In contrast, our graph learning approach (HARG-SSVM) does not require such a reference graph, and consistently outperforms all the other learning approaches, including those with the source reference graphs. The error is measured by the average distance from true matching points.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>accuracy (%)</td>
<td>error</td>
<td>accuracy (%)</td>
<td>error</td>
<td>accuracy (%)</td>
</tr>
<tr>
<td>Sample</td>
<td>w/o learning</td>
<td>60.4</td>
<td>0.079</td>
<td>62.4</td>
<td>0.070</td>
</tr>
<tr>
<td></td>
<td>SW-SSVM</td>
<td>59.7</td>
<td>0.073</td>
<td>53.9</td>
<td>0.114</td>
</tr>
<tr>
<td></td>
<td>SW-SPEC</td>
<td>62.2</td>
<td>0.068</td>
<td>58.8</td>
<td>0.103</td>
</tr>
<tr>
<td></td>
<td>DW-SSVM</td>
<td>66.2</td>
<td>0.054</td>
<td>70.4</td>
<td>0.050</td>
</tr>
<tr>
<td>Source</td>
<td>w/o learning</td>
<td>70.0</td>
<td>0.044</td>
<td>76.1</td>
<td>0.034</td>
</tr>
<tr>
<td></td>
<td>SW-SSVM</td>
<td>70.9</td>
<td>0.042</td>
<td>68.1</td>
<td>0.061</td>
</tr>
<tr>
<td></td>
<td>SW-SPEC</td>
<td>71.8</td>
<td>0.040</td>
<td>68.6</td>
<td>0.053</td>
</tr>
<tr>
<td></td>
<td>DW-SSVM</td>
<td>73.9</td>
<td>0.035</td>
<td>77.7</td>
<td>0.030</td>
</tr>
<tr>
<td></td>
<td>HARG-SSVM</td>
<td>79.7</td>
<td>0.026</td>
<td>79.5</td>
<td>0.027</td>
</tr>
</tbody>
</table>

For our method (HARG-SSVM), we directly learn the model and use it to match with a test set. For all the other methods, since they do not learn a model, we take a reference point set, and use it to match with the test set. Here, two types of references are used: ‘Sample’ or ‘Source’. For ‘Sample’, we randomly select one of the training sample point sets, and for ‘Source’, the points from the source set are directly used. In other words, ‘source’ corresponds to an ideal reference graph without deformations and noise, i.e., a graph formed by red dots on the left image of Fig. 3(a).

We performed learning with 10 inliers (n = 10), 5 outliers (n_o = 5), and maximum variance of 0.15. A comparison of matching performance on 1000 test samples from 10 source sets is shown in Table 1. We also use different graph matching algorithms to account for dependency on the matching algorithm used. Note that our method HARG-SSVM consistently outperforms all the other methods, when a sample set is used as the reference graph. Furthermore, HARG-SSVM provides better results even when other methods use the source set as the reference graph. This is because our graph model additionally captures both the variation and the importance in edge attributes as shown in Fig. 3(b). The two methods, SW-SSVM and SW-SPEC, do not show a notable improvement compared to w/o learning because of the limitations of using shared weights. Given the fact that in realistic situations we cannot directly access the source of information, and can only sample from the source, this synthetic experiment suggests that our learning approach leads to improvement in such problems.

4.2. House/Hotel dataset

The CMU House/Hotel sequence is one of the most popular benchmark datasets for graph matching. We used the image sequences (House: 111 images and Hotel: 101 images) and the feature points from [5, 21]. In one of their experiments, they used 5 training images for learning, and tested on all the remaining pairs. Here, we learn a graph model using only 3 training images (#0, #50, #100) from the 5 images they used, and match it to all the other test images. Unlike [5, 21], we only rely on the edge attributes without any appearance descriptor. As shown in Table 2, learned HARG achieves perfect results without any mismatch, outperforming all the other methods. Figure 5 shows the learned graphs and some example results of matching to other frames. For images of identical objects, such as these House/Hotel sequences, only a few number of images are sufficient for our method to learn the model graph. In Table 2, we also compared with other learning approaches (SW-SSVM, DW-SSVM) using a reference graph (#0). In this case, DW-SSVM also provides near-perfect results, which suggests that learning individual parameters of edges is important for matching.

4.3. Object class datasets

We tested our method on learning graphs of visual object classes. This experiment is different from the previous ones for two reasons: First, the image contains not only background clutter but also significant intra-category, and viewpoint changes. Second, contrary to many related works [5, 21, 26, 30], we use a local feature detector without any additional manual pruning on cluttered images. In this case, a significant number of outliers exists, and thus graph matching becomes challenging.

We use the scale-invariant Hessian detector [24] to detect local regions as nodes. We then describe the edge and node attributes, as detailed in Sec. 3.1. Given a test image, we select kNN features (k = 50) for each node of the graph model based on dot product similarity, and construct...
Table 2: Matching performance comparison on the CMU House/Hotel sequences. The frame #0 was used as a reference graph in w/o learning, SW-SSVM, and DW-SSVM. The numbers in parentheses denote the number of training images used for each method. In the last three rows, we also report the published results for [5, 21] on 5 and 106 training images. Note that while we learn a graph model and match it to all the other images, they learned the parameters for matching, and applied them to match all possible pairs among all the other images.

<table>
<thead>
<tr>
<th>Method</th>
<th>House</th>
<th>Hotel</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o learning</td>
<td>99.6</td>
<td>78.7</td>
</tr>
<tr>
<td>SW-SSVM (3)</td>
<td>99.6</td>
<td>94.6</td>
</tr>
<tr>
<td>DW-SSVM (3)</td>
<td>99.8</td>
<td>100.0</td>
</tr>
<tr>
<td>HARG-SSVM (3)</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>SW-SPEC [21] (5)</td>
<td>99.8</td>
<td>94.8</td>
</tr>
<tr>
<td>SW-SSVM [5] (5)</td>
<td>&lt; 84</td>
<td>&lt; 87</td>
</tr>
<tr>
<td>SW-SSVM [5] (106)</td>
<td>&lt; 96</td>
<td>&lt; 90</td>
</tr>
</tbody>
</table>

(a) a learned House graph and its matching examples
(b) a learned Hotel graph and its matching examples

Figure 5: Results on the CMU House/Hotel sequence. For each sequence, a graph model is learned using 3 images, and tested on all the other images (108 for House, 98 for Hotel). From left to right, the learned graph and two matching examples are shown. In the graph models, darker lines denote edges with higher weights. For comparison with other methods, see Table 2. (Best viewed in color.)

We presented a novel graph learning approach with a histogram-based representation and an SSVM framework. In synthetic and real data experiments, we demonstrated that the proposed method effectively learns an inherent model graph from a training set, and provides good generalization to unseen instances for matching. In future work, we plan to explore sparse graph representation for better efficiency.

5. Conclusion

We presented a novel graph learning approach with a histogram-based representation and an SSVM framework. In synthetic and real data experiments, we demonstrated that the proposed method effectively learns an inherent model graph from a training set, and provides good generalization to unseen instances for matching. In future work, we plan to explore sparse graph representation for better efficiency.

Acknowledgments. The authors would like to thank Olivier Duchenne and Francis Bach for helpful discussions, and Marius Leordeanu for providing his code and data. This research was supported in part by the ERC advanced grant VideoWorld, Institut Universitaire de France, and the Quaero programme funded by the OSEO.

References


---

4The dataset consists of 109 Face, 50 Duck, 66 Wine bottle, 40 Motorbike, and 40 Car images, and is available from our project website [1].
Table 3: Matching performance on 5 object classes from Caltech-256 and PASCAL VOC2007 datasets. For each class, the average performance on 20 random splits of the data is reported. Our method (HARG-SSVM) shows the best matching results on all the 5 classes. For more details, see text and our project website [1].

<table>
<thead>
<tr>
<th>Method</th>
<th>Face</th>
<th>Motorbike</th>
<th>Car</th>
<th>Duck</th>
<th>Wine bottle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc. (%)</td>
<td>error</td>
<td>Acc. (%)</td>
<td>error</td>
<td>Acc. (%)</td>
</tr>
<tr>
<td>w/o learning</td>
<td>66.6</td>
<td>0.205</td>
<td>44.1</td>
<td>0.226</td>
<td>34.1</td>
</tr>
<tr>
<td>SW-SSVM</td>
<td>75.3</td>
<td>0.142</td>
<td>48.6</td>
<td>0.211</td>
<td>40.3</td>
</tr>
<tr>
<td>SW-SPEC</td>
<td>78.7</td>
<td>0.133</td>
<td>47.2</td>
<td>0.212</td>
<td>42.1</td>
</tr>
<tr>
<td>DW-SSVM</td>
<td>84.3</td>
<td>0.102</td>
<td>54.2</td>
<td>0.189</td>
<td>50.8</td>
</tr>
<tr>
<td>HARG-SSVM</td>
<td>93.9</td>
<td>0.070</td>
<td>71.4</td>
<td>0.134</td>
<td>71.9</td>
</tr>
</tbody>
</table>

Figure 4: Learned graph models and their matching results. For each class, the learned graph model, its learned appearance in node attributes, and matching results are shown. In the graph model, bigger circles represent stronger nodes, and darker lines denote stronger edges. In the second and the fifth columns, to better visualize node attributes, we show the edge responses based on the learned SIFT attributes. For each model, some matching examples with high scores are shown. The results show that the learned graph model enables robust matching in spite of deformation and appearance changes. (Best viewed in pdf.)