

# Many-to-Many Graph Matching: a Continuous Relaxation Approach

Mikhail Zaslavskiy<sup>1,2,3,5</sup>, Francis Bach<sup>4</sup> and Jean-Philippe Vert<sup>1,2,3</sup>

<sup>1</sup>Centre for Computational Biology, Mines ParisTech, Fontainebleau, France

<sup>2</sup>Institut Curie and <sup>3</sup>INSERM, U900, Paris, F-75248 France

<sup>4</sup>INRIA-WILLOW project, Ecole Normale Supérieure, Paris, France

<sup>5</sup>Centre for Mathematical Morphology, Mines ParisTech, Fontainebleau, France

10 March 2010

## Abstract

Graphs provide an efficient tool for object representation in various computer vision applications. Once graph-based representations are constructed, an important question is how to compare graphs. This problem is often formulated as a graph matching problem where one seeks a mapping between vertices of two graphs which optimally aligns their structure. In the classical formulation of graph matching, only one-to-one correspondences between vertices are considered. However, in many applications, graphs cannot be matched perfectly and it is more interesting to consider many-to-many correspondences where clusters of vertices in one graph are matched to clusters of vertices in the other graph. In this paper, we formulate the many-to-many graph matching problem as a discrete optimization problem and propose an approximate algorithm based on a continuous relaxation of the combinatorial problem. We compare our method with other existing methods on several benchmark computer vision datasets.

## 1 Introduction

Graphs provide a convenient and efficient tool for object representation in various computer vision applications. An image or an object in an image can typically

be represented by a segmentation, contours, shock graph, or interest points (see, e.g., [2]). Once a graph representation is chosen, a fundamental question that often arises is that of *comparing* graphs in order to compare images or objects. In particular, it is important in many applications to assess quantitatively the similarity between graphs (e.g., for applications in supervised or unsupervised classification), and to detect similar parts between graphs (e.g., for identification of interesting patterns in the data).

Graph matching is one approach to perform these tasks. In graph matching, one tries to “align” two graphs by matching their vertices in such a way that most edges are conserved across matched vertices. Graph matching is useful both to assess the similarity between graphs (e.g., by checking how much the graphs differ after alignment), and to capture similar parts between graphs (e.g., by extracting connected sets of matched vertices). This graph matching framework has many applications in computer vision, e.g., to match 2D or 3D shapes [3–5], or to match deformable objects [6], where methods based on linear or projective transforms usually fail [7].

Classically, only one-to-one mappings are considered in graph matching. In other words, each vertex of the first graph can be matched to only one vertex of the second graph, and vice-versa<sup>1</sup>. This problem can be formulated as a discrete optimization problem, where one wishes to find a one-to-one matching which maximizes the number of conserved edges after alignment. This problem is NP-hard for general graphs, and remains impossible to solve *exactly* in practice for graphs with more than 30 vertices or so. Therefore much effort has been devoted to the development of approximate methods which are able to find a “good” solution in reasonable time. These methods can roughly be divided into two large classes. The first group consists of various local optimization algorithms on the set of permutation matrices, including  $A^*$ -Beam-search [8] and genetic algorithms. The second group consists in solving a continuous relaxation of the discrete optimization problem, such as the  $\ell_1$ -relaxation [9], the Path algorithm [10], various spectral relaxations [5, 11–14] or power methods [6].

In practice, we are sometimes confronted with situations where the notion of one-to-one mapping is too restrictive, and where we would like to allow the possibility to match groups of vertices of the first graph to groups of vertices of the second graph. We call such a mapping *many-to-many*. For instance, in computer vision, the same parts of the same object may be represented by different numbers

---

<sup>1</sup>Note that with the introduction of dummy nodes, one may match a vertex of the first graph to *up to one* vertex of the second graph (see, e.g., [3])

of vertices depending on the noise in the image or on the choice of object view, and it could be relevant to match together groups of vertices that represent the same part. From an algorithmic point of view, this problem has been much less investigated than the one-to-one matching problem. Some one-to-one matching methods based on local optimization over the set of permutation matrices have been extended to many-to-many matching, e.g., by considering the possibility to merge vertices and edges in the course of optimization [15, 16]. Spectral methods have also been extended to deal with many-to-many matching by combining the idea of spectral decomposition of graph adjacency matrices with clustering methods [12, 17]. However, while the spectral approach for one-to-one matching can be interpreted as a particular continuous relaxation of the discrete optimization problem [11], this interpretation is lost in the extension to many-to-many matching. In fact, we are not aware of a proper formulation of the many-to-many graph matching problem as an optimization problem solved by relaxation techniques.

Our main contribution is to propose such a formulation of the many-to-many graph matching problem as a discrete optimization problem, which generalizes the usual formulation for one-to-one graph matching (Section 2), and to present an approximate method based on a continuous relaxation of the problem (Section 3). The relaxed problem is not convex, and we solve it approximately with a conditional gradient method. We also study different ways to map back the continuous solution of the relaxed problem into a many-to-many matching. We present experimental evidence in Section 5, both on simulated and simple real data, that this formulation provides a significant advantage over other one-to-one or many-to-many matching approaches.

## 2 Many-to-many graph matching as an optimization problem

In this section we derive a formulation of the many-to-many graph matching problem as a discrete optimization problem. We start by recalling the classical expression of the one-to-one matching problem as an optimization problem. We then show how to extend the one-to-one formulation to the case of one-to-many matchings. Finally we describe how we can define many-to-many matchings via two many-to-one mappings.

**One-to-one graph matching.** Let  $G$  and  $H$  be two graphs with  $N$  vertices (if the graphs have different numbers of vertices, we can always add dummy nodes with

no connection to the smallest graph). We also denote by  $G$  and  $H$  their respective adjacency matrices, i.e, square  $\{0, 1\}$ -valued matrices of size  $N \times N$  with element  $(i, j)$  equal to 1 if and only if there is an edge between vertex  $i$  and vertex  $j$ .

A one-to-one matching between  $G$  and  $H$  can formally be represented by a  $N \times N$  permutation matrix  $P$ , where  $P_{ij} = 1$  if the  $i$ -th vertex of graph  $G$  is matched to the  $j$ -th vertex of graph  $H$ , and  $P_{ij} = 0$  otherwise. Denoting by  $\|\cdot\|_F$  the Frobenius norm of matrices, defined as  $\|A\|_F^2 = \text{tr}A^T A = (\sum_i \sum_j A_{ij}^2)$ , we note that  $\|G - PHP^T\|_F^2$  is twice the number of edges which are not conserved in the matching defined by the permutation  $P$ . The one-to-one graph matching problem is therefore classically expressed as the following discrete optimization problem:

$$\begin{aligned} \min_P \|G - PHP^T\|_F^2 \text{ subject to } P \in \mathcal{P}_{oto}, \text{ with} \\ \mathcal{P}_{oto} = \{P \in \{0, 1\}^{N \times N}, P1_N = 1_N, P^T 1_N = 1_N\}, \end{aligned} \quad (1)$$

where  $1_N$  denotes the constant  $N$ -dimensional vector of all ones. We note that  $\mathcal{P}_{oto}$  simply represents the set of permutation matrices. The convex hull of this set is the set of doubly stochastic matrices, where the constraint  $P \in \{0, 1\}^{N \times N}$  is replaced by  $P \in [0, 1]^{N \times N}$ .

**From one-to-one to one-to-many.** Suppose now that  $G$  has less vertices than  $H$ , and that our goal is to find a matching that associates each vertex of  $G$  with one or more vertices of  $H$  in such a way that each vertex of  $H$  is matched to a vertex of  $G$ . We call such a matching *one-to-many* (or many-to-one). The problem of finding an optimal one-to-many matching can be formulated as minimizing the same criterion as (1) but modifying the optimization set as follows:

$$\begin{aligned} \mathcal{P}_{otm}(N_G, N_H) = \{P \in \{0, 1\}^{N_G \times N_H}, \\ P1_{N_H} \leq k_{max}1_{N_G}, P1_{N_H} \geq 1_{N_G}, P^T 1_{N_G} = 1_{N_H}\}, \end{aligned}$$

where  $N_G$  denotes the size of graph  $G$ ,  $N_H$  denotes the size of graph  $H$ , and  $k_{max}$  denotes an optional upper bound on the number of vertices that can be matched to a single vertex. As opposed to the one-to-one matching case, each row sum of  $P$  is allowed to be larger than one, and the non-zero elements of the  $i$ -th row of  $P$  corresponds to the vertices of graph  $H$  which are matched to the  $i$ -th vertex of  $G$ .

Most of existing continuous relaxation techniques may be adopted for one-to-many matching. For example, [14] describes how spectral relaxation methods may be used in the case of one-to-many matching. Other techniques like convex

relaxation [10] may be used as well since the convex hull of  $\mathcal{P}_{otm}$  is also obtained by relaxing the constraint  $P \in \{0, 1\}^{N \times N}$  to  $P \in [0, 1]^{N \times N}$ .

**From one-to-many to many-to-many.** Now to match two graphs  $G$  and  $H$  under many-to-many constraints we proceed as if we matched these two graphs to a virtual graph  $S$  under many-to-one constraints, minimizing the difference between the transformed graph obtained from  $G$  and the transformed graph obtained from  $H$ . The idea of many-to-many matching as a double one-to-many matching is illustrated in Figure 1. Graph  $S$  (assumed to have  $L$  vertices) represents the graph

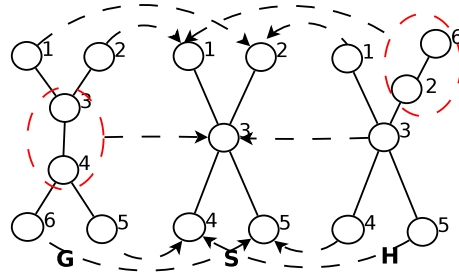


Figure 1: Many-to-many matching between  $G$  and  $H$  via many-to-one matching of both graphs to a virtual graph  $S$ .

of matched vertex clusters. Each vertex of  $S$  corresponds to a group of vertices of  $G$  and a group of vertices of  $H$  matched to each other. Let  $P_1 \in \mathcal{P}_{mto}(L, N_G)$  denote a many-to-one matching  $G \rightarrow S$ , and  $P_2 \in \mathcal{P}_{mto}(L, N_H)$  a many-to-one matching  $H \rightarrow S$ ; we propose to formulate the many-to-many graph matching problem as an optimization problem where we seek  $S$ ,  $P_1$  and  $P_2$  which minimize the difference between  $S$  and  $P_1 G P_1^\top$  and between  $S$  and  $P_2 H P_2^\top$ . The intermediate graph  $S$  may be squeezed out by considering directly the difference between  $P_1 G P_1^\top$  and  $P_2 H P_2^\top$ . We end up with the following objective function for the many-to-many graph matching problem:

$$F(P_1, P_2) = \|P_1 G P_1^\top - P_2 H P_2^\top\|_F^2, \quad (2)$$

where  $P_1 \in \mathcal{P}_{mto}(L, N_G)$  and  $P_2 \in \mathcal{P}_{mto}(L, N_H)$  denote two many-to-one mappings. The objective function (2) is similar to the objective function for the one-to-one case (1). In (1), we seek a permutation which makes the second graph  $H$  as similar as possible to  $G$ . In (2), we seek *combinations of merges and permutations* which makes  $G$  and  $H$  as similar as possible to each other. The only difference between both formulations is that in the many-to-many case we add the merging operation.

There are two slightly different ways of defining the set of matrices over which (2) is minimized. We can fix in advance the number of matching clusters  $L$ , which corresponds to the size of  $S$ , in which case the optimization set is  $P_1 \in \mathcal{P}_{mto}(L, N_G)$  and  $P_2 \in \mathcal{P}_{mto}(L, N_H)$ . An alternative way which we follow in the paper is to remove the constraint  $P_1 \mathbf{1}_{N_G} \geq \mathbf{1}_L$  from the definition of  $\mathcal{P}_{mto}(L, N_G)$ , in this case the method estimates itself the number of matching clusters (number of rows with non-zero sum). Finally, we thus formulate the many-to-many graph matching problem as follows:

$$\begin{aligned} \min_{P_1, P_2} & \|P_1 G_1^\top P_1^\top - P_2 H P_2^\top\|_F^2 \quad \text{subject to} \\ P_1 & \in \{0, 1\}^{N_K \times N_G}, P_1 \mathbf{1}_{N_G} \leq k_{max} \mathbf{1}_{N_K}, P_1^\top \mathbf{1}_{N_K} = \mathbf{1}_{N_G}, \\ P_2 & \in \{0, 1\}^{N_K \times N_H}, P_2 \mathbf{1}_{N_H} \leq k_{max} \mathbf{1}_{N_K}, P_2^\top \mathbf{1}_{N_K} = \mathbf{1}_{N_H}, \end{aligned} \quad (3)$$

where  $N_K = \min(N_G, N_H)$  represents the maximal number of matching clusters. This formulation is in fact valid for many kinds of graphs, in particular graphs may be directed (with asymmetric adjacency matrices), have edge weights (with real-valued adjacency matrices), and self-loops (with non-zero diagonal elements in the adjacency matrices). We also describe in Section 3 how this formulation can be modified to include information about vertex labels, which are important for computer vision (see, e.g., [3]).

### 3 Continuous relaxations of the many-to-many graph matching problem

The many-to-many graph matching problem (3) is a hard discrete optimization problem. We therefore need an approximate method to solve it in practice. In this section we propose an algorithm based on a continuous relaxation of (3). For that purpose we propose to replace the binary constraints  $P_1 \in \{0, 1\}^{N_K \times N_G}$ ,  $P_2 \in \{0, 1\}^{N_K \times N_H}$  by continuous constraints  $P_1 \in [0, 1]^{N_K \times N_G}$ ,  $P_2 \in [0, 1]^{N_K \times N_H}$ . Note that if we had a linear objective function in  $(P_1, P_2)$ , the continuous relaxation would be exact because we simply replace the optimization set by its convex hull. However, our objective function (3) is quartic, and its optimum is in general not an extreme point of the optimization set. To solve the relaxed optimization problem we propose to use the following version of the conditional gradient (a.k.a. Frank-Wolfe method [18]):

- Input: initial values  $P_1^0$  and  $P_2^0$ ,  $t = 0$ ,

- Do
  1. compute  $\nabla F(P_1^t, P_2^t)$
  2. find the minimum of  $\nabla F(P_1^t, P_2^t)^\top (P_1, P_2)$  w.r.t.  $(P_1, P_2)$
  3. perform line search in the direction of the optimum found in Step 2, assign the result to  $P_1^{t+1}, P_2^{t+1}, t = t + 1$
- Until  $|\Delta F| + \|\Delta P_1\|_F + \|\Delta P_2\|_F < \varepsilon$
- Output:  $P_1^t, P_2^t$ .

The minimization of a linear function in step 2, i.e.,  $\min_P \nabla F(P_1^t, P_2^t)^\top (P_1, P_2)$  is a version of the linear semi-assignment problem, and reduces to the classical linear assignment problem by adding dummy nodes. We then have to solve a linear assignment problem for a  $k_{max}(N_G + N_H) \times N_H$  matrix, which can be done efficiently by the Hungarian algorithm [19]. The solution of the line search step can be found in closed form since the objective function is a polynomial of the fourth order.

The conditional descent algorithm converges to a stationary point of (3) [18]. Because of the non-convex nature of the objective function, we can only hope to reach a local minimum (or more generally a stationary point) and it is important to have a good initialization. In our experiments we found that a good choice is the fixed “uniform” initialization, where we initialize  $P_1$  by  $\frac{1}{N_K} \mathbf{1}_{N_G} \mathbf{1}_{N_H}^\top$  and  $P_2$  by the identity matrix  $I$ . Another option would be to use a convex relaxation of one-to-one matching [10].

Algorithm complexity is mainly defined by two parameters:  $N = k_{max}(N_G + N_H)$  and  $\varepsilon$ . In general the number of iterations of the gradient descent scales as  $O(\frac{\kappa}{\varepsilon})$  where  $\kappa$  is the condition number of the Hessian matrix describing the objective function near a local minima [18].  $N$  has no direct influence on the number of iterations, but it defines the cost of one iteration, i.e., the complexity of the Hungarian algorithm  $O(N^3)$ .

**Projection.** Once we have reached a local optimum of the relaxed optimization problem, we still need to project  $P_1$  and  $P_2$  to the set of matrices with values in  $\{0, 1\}$  rather than in  $[0, 1]$ . Several alternatives can be considered. A first idea is to use the columns of  $P_1$  and  $P_2$  to define a similarity measure between the vertices of both graphs, e.g., by computing the dot products between columns. Indeed, the more similar the columns corresponding to two vertices, the more likely these vertices are to be matched if they are from different graphs, or merged if they are

from the same graph. Therefore a first strategy is to run a clustering algorithm (e.g., K-means or spectral clustering) on the column vectors of the concatenated matrix  $(P_1, P_2)$  and then use the resulting clustering to construct the final many-to-many graph matching.

An alternative to clustering is an incremental projection or forward selection projection, which uses the matching objective function at every step. Once  $P_1$  and  $P_2$  are obtained from the continuous relaxation, we take the pair of vertices  $(g, h)$  from the union of the graphs having the most similar column vectors in  $(P_1, P_2)$ . We then re-run the continuous relaxation with the new (linear) constraint that these two vertices remain matched. We then go on and find the most similar pair of vertices from the constrained continuous solution. This greedy scheme can be iterated until all vertices are matched.

In our experiments, the second approach produced better results. This is mainly due to the fact that when we just run a clustering algorithm we do not use the objective function, while when we use incremental projection we adapt column vectors of unmatched vertices according to earlier established matchings.

**Neighbor merging.** In many cases, it can be interesting to favor the merging of neighboring vertices, as opposed to merging of any sets of vertices. To that end we propose the following modification to (3):

$$F_N(P_1, P_2) = F(P_1, P_2) - \text{tr}G^\top P_1^\top P_1 - \text{tr}H^\top P_2^\top P_2.$$

The matrix product  $P_1^\top P_1$  is a  $N_G \times N_G$  matrix, with  $(i, j)$ -th entry equal to 1 if  $i$  and  $j$  are merged into the same cluster. Therefore, the new components in the objective function represent the number of pairs of adjacent vertices merged together in  $G$  and  $H$ , respectively.

**Local similarities.** Like the one-to-one formulation, we can easily modify the many-to-many graph matching formulation to include information on vertex pair-wise similarities by modifying the objective function as follows:

$$F_\lambda(P_1, P_2) = (1 - \lambda)F(P_1, P_2) + \lambda \text{tr}C^\top P_1^\top P_2, \quad (4)$$

where the matrix  $C \in \mathbb{R}^{N_G \times N_H}$  is a matrix of local dissimilarities between graph vertices, and parameter  $\lambda$  controls the relative impact of information on graph vertices and information on graph structures. The new objective function is again a polynomial of the fourth order, so our algorithm may still be used directly without any additional modifications.

## 4 Related methods

There exist two major groups of methods for many-to-many graph matching, which we briefly describe in this section. The first one consists of local search algorithms, generally used in the context of the graph edit distance, while the second one is composed of variants of the spectral approach.

**Local search algorithms.** Examples of this kind of approach are given in [15] and [16]. In the classical formulation of the graph edit distance, the set of graph edit operations consists of deletion, insertion and substitution of vertices and edges. Each operation has an associated cost, and the objective is to find a sequence of operations with the lowest total cost transforming one graph into another. In the case of many-to-many graph matching, this set of operations is completed by merging (and splitting if necessary) operations. Since the estimation of the optimal sequence is a hard combinatorial problem, approximate methods such as beam search [8] as well as other examples of best-first, breadth-first and depth-first searches are used.

**Spectral approach.** Caelli and Kosinov [12] discuss how spectral matching may be used for many-to-many graph matching. Their algorithm is similar to the Umeyama method [11] but instead of one-to-one correspondences, they search a many-to-many mapping by running a clustering algorithm. In the first step, the spectral decomposition of graph adjacency matrices is considered

$$G = V_G \Lambda_G V_G^\top, H = V_H \Lambda_H V_H^\top. \quad (5)$$

Rows of eigenvector matrices  $V_G$  and  $V_H$  are interpreted as spectral coordinates of graph vertices. Then vertices having similar spectral coordinates are clustered together by a clustering algorithm, and vertices grouped in the same cluster are considered to be matched.

Another example of spectral approach is given in [17] where, roughly speaking, the adjacency matrix is replaced by the matrix of shortest path distances, and then spectral decomposition with further clustering is used.

## 5 Experiments

In this section we compare the new method proposed in this paper with existing techniques (beam-search and spectral approach). We thus test three competitive approaches on several experiments: beam-search “Beam” (A\*-beam search from

[8]), the spectral approach “Spec” [12], and our new gradient descent method “Grad” (from Section 3).

## 5.1 Synthetic examples

In this section, we compare the three many-to-many graph matching algorithms on pairs of randomly generated graphs with similar structures. We generate graphs according to the following procedure: (1) generate a random graph  $G$  of size  $N$ , where each edge is present with probability  $p$ , (2) build a randomly permuted copy  $H$  of  $G$ , (3) randomly split the vertices in  $G$  (and in  $H$ ) by taking a random vertex in  $G$  (and in  $H$ ) and split it into two vertices (operation repeated  $M$  times), (4) introduce noise by adding/deleting  $\sigma \times p \times N^2$  random edges in both graphs.

As already mentioned, our principal interest here is to understand the behavior of graph matching algorithms as a function of the graph size  $N$ , and their ability to resist to structural noise. Indeed, in practice we never have identical graphs and it is important to have a robust algorithm which is able to deal with noise in graph structures. The objective function  $F(P_1, P_2)$  in (3) represents the quality of graph matching, so to compare different graph matching algorithms we plot  $F(P_1, P_2)$  as a function of  $N$  (Figure 2a), and  $F(P_1, P_2)$  as a function of  $\sigma$  (Figure 2b) for the three algorithms. In both cases, we observe that “Grad” significantly outperforms both “Beam” and “Spec”. “Beam” was run with beam width equal to 3, which represents a good trade-off between quality and complexity, “Spec” was run with projection on the first two eigenvectors with the normalization presented in [12]<sup>2</sup>. Figure 2c shows how algorithms scale in time with the graph size  $N$ . The “Spec” algorithm is the fastest one, but “Grad” has the same complexity order as “Spec” (corresponding curves are almost parallel lines in log-log scale, so both functions are polynomials with the same degree and different multiplication constants), these curves are coherent with theoretical values of algorithm complexity summarized in Section 3. The “Beam” algorithm is much slower, and it also has worse complexity order.

## 5.2 Chinese characters

In this section we *quantitatively* compare many-to-many graph matching algorithms as parts of a classification framework. We use graph matching algorithms

---

<sup>2</sup>“Spec” variants with three and more eigenvectors were also tested, but two eigenvectors produced almost the same matching quality and worked faster.

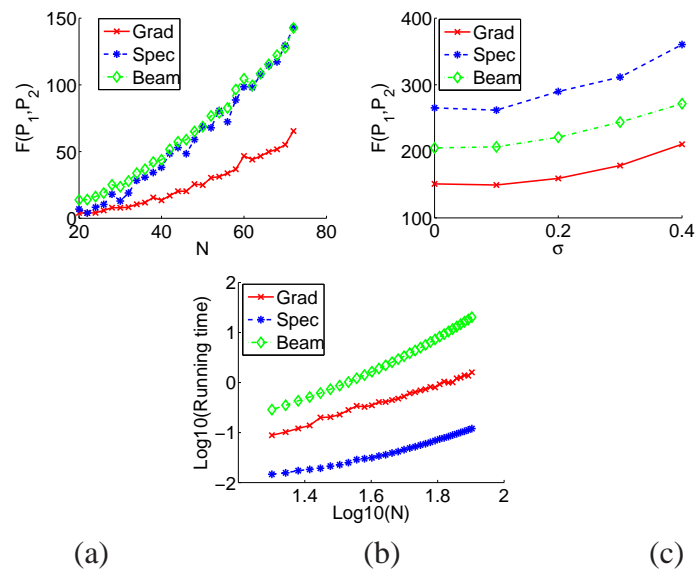


Figure 2: (a)  $F(P_1, P_2)$  (mean value over 30 repetitions) as a function of graph size  $N$ , simulation parameters:  $p = 0.1, \sigma = 0.05, M = 3$ . (b)  $F(P_1, P_2)$  (mean value over 30 repetitions) as a function of noise parameter  $\sigma$ , simulation parameters:  $N = 30, p = 0.1, M = 3$ . (c) Algorithm running time (mean value over 30 repetitions) as a function of  $N$  (log-log scale), other parameters are the same as in (a), “Beam” slope  $\approx 3.8$ , “Grad” slope  $\approx 2.5$ , “Spec” slope  $\approx 2.7$ .

to compute similarity/distance between objects of interest on the basis of their graph-based representations. As the classification problem, we chose the ETL9B dataset of Chinese characters. This dataset is well suited for our purposes, since Chinese characters may be naturally represented by graphs with variable non-trivial structures.

Figure 3 illustrates how “Grad” works on graphs representing Chinese characters. We see that our algorithm produces a good matching, although not perfect, providing a correspondence between “crucial” vertices. The characters rep-

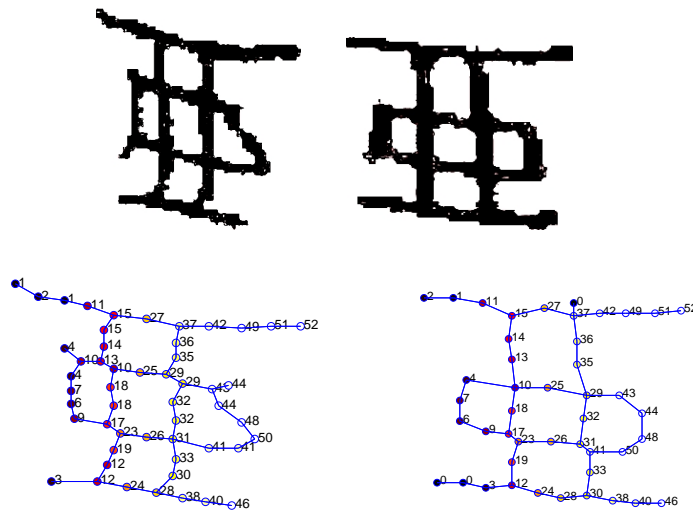


Figure 3: Different writings of the same Chinese character and the matching of the corresponding graphs made by “Grad”. Vertices having the same id’s are matched to each other.

resented in Figure 3 are however very easy to recognize, and most classification algorithms show a good performance on them; for example, “Grad” produces a classification error rate below 0.2%. To test graph matching algorithms on more challenging situations, we chose three “hard to classify” Chinese characters, i.e., three characters sharing similar graph structures, as illustrated in Table 1. We ran k-nearest neighbor (k-NN) with graph matching algorithms used as distance measures. The dataset consists of 600 images, 200 images of each class.

Table 1 shows classification results for the three many-to-many graph matching algorithms. In addition we report results for other popular approaches, namely, a SVM classifier with linear and Gaussian kernels, one-to-one matching with the Path algorithm (taken from [10]) and two versions of the shape context method [3],

with or without thin plate spline smoothing. The version named “shape context” computes polar histograms with further bipartite graph matching. To run the “shape context+tps” method we used code available online<sup>3</sup>.

Graph matching algorithms are run using information on vertex coordinates through (2). The elements of the matrix  $C$  are defined as  $C_{ij} = e^{-(x_i-x_j)^2-(y_i-y_j)^2}$ . The parameter  $\lambda$  in (2) as well as  $k$  (number of neighbors in k-NN classifier) are learned via cross-validation. We see that the “Grad” algorithm shows the best performance, outperforming other many-to-many graph matching algorithms as well as other competitive approaches.

Table 1: Top: chinese characters from three different classes. Bottom: classification results (mean and standard deviation of test error over cross-validation runs, with 50 repetitions of five folds)

違 遺 達

Method	error	STD
Linear SVM	0.377	± 0.090
SVM with Gaussian kernel	0.359	± 0.076
k-NN (one-to-one, Path)	0.248	± 0.075
k-NN (shape context)	0.399	± 0.081
k-NN (shape context+tps)	0.435	± 0.092
k-NN (Spec)	0.254	± 0.071
k-NN (Beam)	0.283	± 0.079
k-NN (Grad)	<b>0.191</b>	± 0.063

### 5.3 Deformable objects matching

One advantage of graph-based image alignment algorithms is that they can be used in problems with deformable objects. Figure 4 shows how the “Grad” algorithm aligns a pair of photos of spiders (for which the graphs have been constructed by hand). These photos are taken from completely different viewpoints, which is a significant difficulty for many existing image alignment approaches based on the grouping of superpixels, such as [20, 21]. Some methods generate various rotations or linear transforms of the same image and then take the best alignment (see,

<sup>3</sup><http://www.eecs.berkeley.edu/vision/shape/>

e.g., [1, 7]), but such approaches are not possible here because of deformations. Since image alignment should be rotation invariant we can not use the explicit vertex coordinates to construct the matrix  $C$  as it was done in the previous section. Instead, we use the shape context features [3]; namely, each vertex gets a feature vector representing the polar histogram of the vectors joining this vertex to the other graph vertices. To make the polar histogram rotation invariant, we align the polar histograms by taking as an origin for angle the direction to the center of mass of all graph vertices. Under such a setup, polar histograms are invariant with respect to rotations around the graph center of mass.

We see in Figure 4 that “Grad” figures out that the top of the first image corresponds to the bottom of the second image, for example, it groups two vertices representing the left part of the second spider head and matches them to one vertex of the left graph representing the same part in the first spider (vertices indexed by number 10).

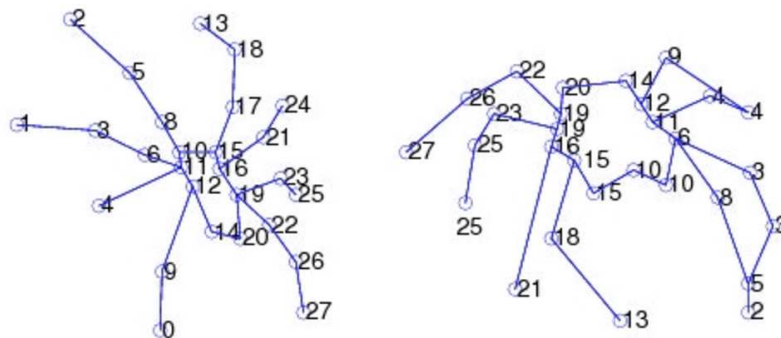


Figure 4: Illustration of rotation invariant matching made by “Grad”. Original spider photos with corresponding graph-based representations are given on the left. On the right, two spider graphs are aligned by “Grad”. Vertices with the same id’s are matched to each other.

## 5.4 Identification of object composite parts

While the pattern recognition framework is interesting and important for the comparison of different graph matching algorithms, it evaluates only one aspect of these algorithms, namely, their ability to detect similar graphs. A second and important aspect is their ability to correctly align vertices corresponding to the same parts of two objects. To test this capability, we performed the following series of experiments. We chose ten camel images from the MPEG7 dataset and we divided by hand each image into 6 parts: head, neck, legs, back, tail and body (Figure 5). This image segmentation automatically defines a partitioning of the corresponding graph shown in the column (c) in Figure 5: all graph vertices are labeled according to the image part which they represent. Figure 5 gives two illustrations of how this procedure works. A good graph matching algorithm should map vertices from corresponding image parts to each other, i.e., heads to heads, legs to legs, and so on. Therefore to evaluate the matching quality of the mapping, we use the following score. First, we match two graphs and then we try to predict vertex labels of one graph given the vertex labels of the second one. For instance, if vertex  $g_1$  of the first image is matched to vertices  $h_1$  and  $h_2$  representing the head of the second image, then we predict that  $g_1$  is of class “head”. The better the graph matching, the smaller the prediction error and vice-versa.

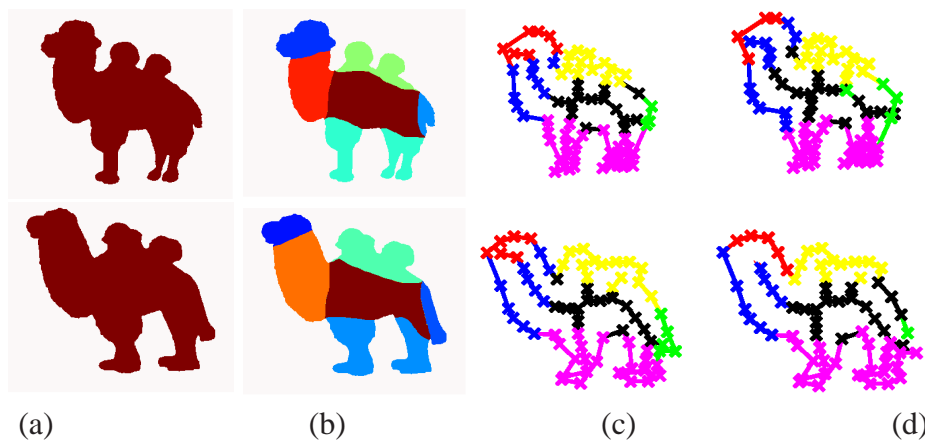


Figure 5: (a) Original images. (b) Manual segmentation (c) Graph-based representation (obtained automatically from subsampled contours and shock graphs) with induced vertex labels (d) Prediction of vertex labels on the basis of graph matching made by “Grad”. Best seen in color.

This experiment illustrates a promising application of graph matching algo-

rithms. Usually segmentation algorithms extract image parts on the basis of different characteristics such as changing of color, narrowing of object form, etc. With our graph matching algorithm, we can extract segments which does not only have a specific appearance, but also have a semantic interpretation defined by a user (e.g., through the manual labelling of a particular instance).

Table 2 presents mean prediction error over 45 pairs of camel images (we exclude comparison of identical images). Each pair has two associated scores: prediction error of the first image given the second one and vice-versa. We thus have 90 scores for each algorithm, which are used to compute means and standard deviations. Like in the previous sections, graph matching algorithms are run using information on vertex coordinates (using Eq. (2)), with  $C_{ij} = e^{-(x_i-x_j)^2-(y_i-y_j)^2}$ . The parameter  $\lambda$  in (2) as well as  $k$  (number of neighbors in k-NN classifier) are learned via cross-validation. Here, again we observe that the “Grad” algorithm works better than other methods.

Table 2: Identification of object composite parts: mean and standard deviation of prediction error (see text for details). Note that standard deviations are not divided by the square root of the sample size (therefore differences are statistically significant).

	Grad	Spec	Beam	One-to-one
Error	<b>0.303</b>	0.351	0.432	0.342
STD	0.135	0.095	0.092	0.094

## 6 Conclusion and Future work

The main contribution of this paper is the new formulation of the many-to-many graph matching problem as a discrete optimization problem and the approximate algorithm “Grad” based on a continuous relaxation. The success of the proposed method compared to other competitive approaches may be explained by two reasons. First, methods based on continuous relaxations of discrete optimization problems often show a better performance than local search algorithm due to their ability to better explore the optimization set with potentially large moves. Second, the “Grad” algorithm aims to optimize a clear objective function naturally representing the quality of graph matching instead of a sequence of unrelated steps.

Besides a natural application of graph matching as a similarity measure between objects with complex structures, graph matching can also be used for ob-

ject alignment. However, the structural noise usually encountered in graph-based representations have slightly hampered its application to natural images; but we believe that the many-to-many graph matching framework presented in this paper can provide an appropriate notion of robustness, which is necessary for computer vision applications. Of course, this requires the validation of our approach with graphs obtained from more cluttered images, which we are currently experimenting with.

## References

- [1] W. E. L. Grimson. *Object Recognition by Computer: The role of geometric constraints*. MIT Press, 1990. 14
- [2] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2003 2
- [3] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(4):509–522, 2002. 2, 6, 12, 14
- [4] Yefeng Z. and D. Doermann. Robust point matching for nonrigid shapes by preserving local neighborhood structures. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(4):643–649, April 2006. 2
- [5] M. Leordeanu and M. Hebert. A spectral technique for correspondence problems using pairwise constraints. In *International Conference of Computer Vision (ICCV)*, volume 2, pages 1482 – 1489, October 2005. 2
- [6] O. Duchenne, F. Bach, I. Kweon, and J. Ponce. A tensor-based algorithm for high-order graph matching. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition CVPR 2009*, pages 1980–1987, 20–25 June 2009. 2
- [7] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. of the ACM*, 24(6):381–395, 1981. 2, 14
- [8] M. Neuhaus, K. Riesen, and H. Bunke. Fast suboptimal algorithms for the computation of graph edit distance. In Dit-Yan Yeung, James T. Kwok, Ana

- L. N. Fred, Fabio Roli, and Dick de Ridder, editors, *SSPR/SPR*, volume 4109 of *Lecture Notes in Computer Science*, pages 163–172. Springer, 2006. 2, 9, 10
- [9] H.A. Almohamad and S.O. Duffuaa. A linear programming approach for the weighted graph matching problem. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(5):522–525, May 1993. 2
- [10] M. Zaslavskiy, F. Bach, and J.-P. Vert. A path following algorithm for the graph matching problem. Technical Report 00232851, HAL, 2008. To appear in *IEEE Trans. Pattern Anal. Mach. Intell.* 2, 5, 7, 12
- [11] S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(5):695–703, Sept. 1988. 2, 3, 9
- [12] T. Caelli and S. Kosinov. An eigenspace projection clustering method for inexact graph matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(4):515–519, April 2004. 2, 3, 9, 10
- [13] M. Carcassoni and E. Hancock. Spectral correspondence for point pattern matching. *Pattern Recogn.*, 36(1):193–204, January 2003. 2
- [14] T. Cour, P. Srinivasan, and J. Shi. Balanced graph matching. In *Advanced in Neural Information Processing Systems*, 2006. 2, 4
- [15] S. Berretti, A. Del Bimbo, and P. Pala. A graph edit distance based on node merging. In *Proc. of ACM International Conference on Image and Video Retrieval (CIVR)*, pages 464–472, Dublin, Ireland, July 2004. 3, 9
- [16] R. Ambauen, S. Fischer, and H. Bunke. Graph edit distance with node splitting and merging, and its application to diatom identification. In *GbrPR*, pages 95–106, 2003. 3, 9
- [17] Y. Keselman, A. Shokoufandeh, M. F. Demirci, and S. Dickinson. Many-to-many graph matching via metric embedding. In *CVPR*, pages 850–857, 2003. 3, 9
- [18] D. Bertsekas. *Nonlinear programming*. Athena Scientific, 1999. 6, 7
- [19] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research*, 2:83–97, 1955. 7

- [20] T. Cour and J. Shi. Recognizing objects by piecing together the segmentation puzzle. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition CVPR '07*, pages 1–8, 17–22 June 2007. 13
- [21] Zhuowen Tu, Xiangrong Chen, and Alan L. Yuille. Image parsing: Unifying segmentation, detection, and recognition. *Int. J. Comput. Vis.*, 2004. 13