

# 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> Center for Computational Biology, Mines ParisTech, Fontainebleau, France

<sup>2</sup> Institut Curie

<sup>3</sup> INSERM, U900, Paris, F-75248, France

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

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

**Abstract.** Graphs provide an efficient tool for object representation in various machine learning 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 two approximate algorithms based on alternative continuous relaxations of the combinatorial problem. We compare new methods with other existing methods on several benchmark datasets.

## 1 Introduction

The necessity to process data with complex structures has triggered the wide use of graph-based representation techniques in various applications domains. Graphs provide a flexible and efficient tools for data representation in computer vision (segmentation, contour and shock graphs), computational biology (biological networks), or chemoinformatics (representation of chemical compounds), to name just a few. A fundamental question when data are represented as graphs is to be able to *compare* graphs. In particular, it is important in many applications to be able 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 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).

---

\* Currently, Mikhail Zaslavskiy is with the bioinformatics group at Collectis S.A.

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 [2] and genetic algorithms. The second group consists in solving a continuous relaxation of the discrete optimization problem, such as the  $\ell_1$ -relaxation [3], the Path algorithm [4], various spectral relaxations [5, 6, 7, 8, 9] or power methods [10].

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 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 [11, 12]. 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 [13, 6]. However, while the spectral approach for one-to-one matching can be interpreted as a particular continuous relaxation of the discrete optimization problem [5], 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 two approximate methods based on different continuous relaxations of the problem (Sections 3.1 and 3.2). In both cases, the relaxed problems are not convex, and we solve them 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.

---

<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., [1]).

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

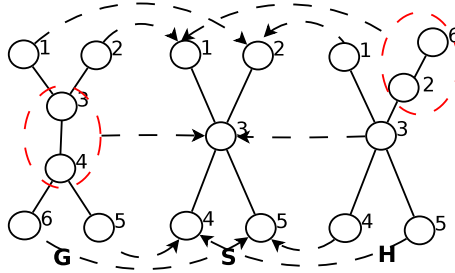
$$\begin{aligned} \mathcal{P}_{mto}(N_G, N_H) = \{P \in \{0, 1\}^{N_G \times N_H}, \\ P1_{N_H} = 1_{N_G}, P^T 1_{N_G} \leq k_{max} 1_{N_H}, P^T 1_{N_G} \geq 1_{N_H}\}, \end{aligned}$$

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 column sum of  $P$  is allowed to be larger than one, and the non-zero elements of the  $j$ -th column of  $P$  corresponds to the vertices of graph  $G$  which are matched to the  $j$ -th vertex of  $H$ .

Most of existing continuous relaxation techniques may be adopted for many-to-one matching. For example, [8] describes how spectral relaxation methods may be used in the case of many-to-one matching. Other techniques like convex relaxation [4] may be

used as well since the convex hull of  $\mathcal{P}_{mto}$  is also obtained by relaxing the constraint  $P \in \{0, 1\}^{N_G \times N_H}$  to  $P \in [0, 1]^{N_G \times N_H}$ .

**From many-to-one 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 many-to-one matching is illustrated in Figure 1. Graph  $S$  (assumed to have  $L$  vertices) represents the graph of matched vertex clusters.



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

Each vertex of  $S$  corresponds to a group of vertices of  $G$  and a group of vertices of  $H$  matched to each other. For example, in Figure 1, vertices  $g_3$  (vertex **3** of  $G$ ),  $g_4$  and  $h_3$  are matched to the same vertex  $s_3$ ; it means that in the final *many-to-many* matching between  $G$  and  $H$ ,  $g_3$  and  $g_4$  will be matched to  $h_3$ . 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  $P_1GP_1^\top$  and between  $S$  and  $P_2HP_2^\top$

$$\min_{P_1, P_2, S} \|P_1GP_1^\top - S\|_F^2 + \|S - P_2HP_2^\top\|_F^2. \tag{2}$$

Note that if we know  $P_1$  and  $P_2$ , then the optimal  $S$  is just  $\frac{1}{2}(P_1GP_1^\top + P_2HP_2^\top)$  (point in the middle between  $P_1GP_1^\top$  and  $P_2HP_2^\top$ ). Plugging this expression in (2) we obtain the following objective function for the many-to-many graph matching problem

$$F(P_1, P_2) = \|P_1GP_1^\top - P_2HP_2^\top\|_F^2, \tag{3}$$

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 (3) 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 (3), 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. Given matrices  $P_1$  and  $P_2$ , it is easy to retrieve the many-to-many matching between  $G$  and  $H$  by considering

$$M = P_1 P_2^\top .$$

Indeed,  $M$  is a  $N_G \times N_H$  binary matrix such that  $M_{ij} = 1$  if and only if  $g_i$  is matched to  $h_j$ .

There are two slightly different ways of defining the set of matrices over which (3) 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:

$$\begin{aligned} & \min_{P_1, P_2} \|P_1 G P_1^\top - P_2 H P_2^\top\|_F^2 \quad \text{subject to} \\ & P_1 \in \{0, 1\}^{N_K \times N_G}, P_1 \mathbf{1}_{N_G} \leq k_{max} \mathbf{1}_{N_K}, P_1^\top \mathbf{1}_{N_K} = \mathbf{1}_{N_G}, \\ & P_2 \in \{0, 1\}^{N_K \times N_H}, P_2 \mathbf{1}_{N_H} \leq k_{max} \mathbf{1}_{N_K}, P_2^\top \mathbf{1}_{N_K} = \mathbf{1}_{N_H}, \end{aligned} \quad (4)$$

where  $N_K = \min(N_G, N_H)$  represents the maximal number of matching clusters. Note that  $N_K$  is only an upper bound on the number of matching clusters, since it can not exceed the size of the smallest graph. On the other hand some of the columns of  $P_1$  and  $P_2$  may be empty, meaning that the corresponding clusters do not contain vertices, i.e., that these clusters do not exist. 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 3.1 how this formulation can be modified to include information about vertex labels, which are important for machine learning applications.

### 3 Continuous Relaxations of the Many-to-Many Graph Matching Problem

The many-to-many graph matching problem (4) is a hard discrete optimization problem. We therefore need an approximate method to solve it in practice.

#### 3.1 Method 1: Gradient Descent

In this section we propose an algorithm based on a continuous relaxation of (4). 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  $P_1 \in [0, 1]^{N_K \times N_G}$ ,  $P_2 \in [0, 1]^{N_K \times N_H}$ . Let  $\mathcal{K}$  denote the new continuous optimization set

$$\begin{aligned} \mathcal{K} &= \mathcal{K}_1 \times \mathcal{K}_2, \text{ where} \\ \mathcal{K}_1 &= \{P_1 \in [0, \mathbf{1}]^{N_K \times N_G}, P_1 \mathbf{1}_{N_G} \leq k_{max} \mathbf{1}_{N_K}, P_1^\top \mathbf{1}_{N_K} = \mathbf{1}_{N_G}\}, \\ \mathcal{K}_2 &= \{P_2 \in [0, \mathbf{1}]^{N_K \times N_H}, P_2 \mathbf{1}_{N_H} \leq k_{max} \mathbf{1}_{N_K}, P_2^\top \mathbf{1}_{N_K} = \mathbf{1}_{N_H}\}, \end{aligned} \quad (5)$$

To solve the relaxed optimization problem we propose to use the following version of the conditional gradient (a.k.a. Frank-Wolfe method [14]):

- 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.  $(P_1, P_2)$  i.e. over  $\mathcal{K}$
  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:  $P_1^t, P_2^t$ .

Step 2 consists in the minimization of a linear function over a convex optimization set. This problem can be solved by a generic linear programming solver in  $O((N_G + N_H)N_K)^{3.5}$ . The minimum of the gradient function is one of the extreme points of the continuous optimization set. Since our ultimate objective is to minimize  $F(P_1, P_2)$  over the discrete optimization set (4) it would be better to move towards one of the points of (4) and not just any extreme point of  $\mathcal{K}$ . The good news is that due to the special structure of the optimization set, the gradient minimization can be solved in  $O(\max(k_{max}N_K)^3)$  by reformulating  $\min_P \nabla F(P_1^t, P_2^t)^\top (P_1, P_2)$  as a linear assignment problem (see Appendix A). We then have to solve a linear assignment problem for a  $(N_G + N_H) \times k_{max} \min(N_H, N_G)$  matrix, which can be done efficiently by the Hungarian algorithm [15].

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 (4) [14]. 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  $\frac{1}{N_K} 1_{N_G} 1_{N_H}^\top$  and  $P_2$  by the identity matrix  $I$ . Another option would be to use a convex relaxation of one-to-one matching [4].

Algorithm complexity is mainly defined by two parameters:

$$N = \max(k_{max} \min(N_G, N_H), N_G + N_H) \text{ and } \varepsilon.$$

In general the number of iterations of the gradient descent scales as  $O(\frac{N}{\varepsilon})$  where  $\kappa$  is the condition number of the Hessian matrix describing the objective function near a local minima [14].  $N$  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 (4):

$$F_N(P_1, P_2) = F(P_1, P_2) - \text{tr}G^\top P_1^\top P_1 - \text{tr}H^\top P_2^\top P_2. \quad (6)$$

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 pairwise similarities by modifying the objective function as follows:

$$F_\lambda(P_1, P_2) = (1 - \lambda)F(P_1, P_2) + \lambda \text{tr}C^\top P_1^\top P_2, \quad (7)$$

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.

### 3.2 Method 2: SDP Relaxation

The second method consists in a relaxation of (4) to a quadratic semidefinite programming (SDP) problem.

First, we rewrite the objective function of (4) in an alternative form

$$\begin{aligned}
& \|P_1GP_1^T - P_2HP_2^T\|_F^2 = \\
& \text{tr}P_1G^T \underbrace{P_1^T P_1}_{M_1} GP_1^T + \text{tr}P_2H^T \underbrace{P_2^T P_2}_{M_2} HP_2^T - 2\text{tr}P_1G^T \underbrace{P_1^T P_2}_{M_{12}} HP_2^T = \\
& \text{tr}M_1G^T M_1G + \text{tr}M_2H^T M_2H - 2\text{tr}M_{21}G^T M_{12}H = \\
& \text{tr} \left[ \underbrace{\begin{pmatrix} M_1 & M_{12} \\ M_{21} & M_2 \end{pmatrix}}_M \underbrace{\begin{pmatrix} G^T & 0 \\ 0 & -H^T \end{pmatrix}}_{A^T} \underbrace{\begin{pmatrix} M_1 & M_{12} \\ M_{21} & M_2 \end{pmatrix}}_M \underbrace{\begin{pmatrix} G & 0 \\ 0 & -H \end{pmatrix}}_A \right] = \\
& \text{tr}MA^TMA = \text{vec}(M)(A^T \otimes A)\text{vec}(M).
\end{aligned} \tag{8}$$

Let  $F(M)$  denote our new objective function  $\text{vec}(M)(A^T \otimes A)\text{vec}(M)$ , now we have to minimize the quadratic function  $F(M)$  over the discrete set  $\mathcal{M}$  of binary matrices with a special structure. Since matrix  $M$  is a positive-semidefinite matrix (indeed,  $M$  can be expressed as  $P^T P$  where  $P$  is the matrix made of  $P_1$  and  $P_2$  stacked one below another), we can relax the optimization problem  $\min_{M \in \mathcal{M}} F(M)$  to the minimization of a quadratic function over the convex set of positive-semidefinite matrices

$$\min_{M \succeq 0} F(M). \tag{9}$$

Therefore the second method consists in the running of the Frank-Wolfe algorithm with an SDP solver to compute conditional gradient and further projection of the produced solution on  $\mathcal{M}$ .

Here again we can run a clustering algorithm using the output of the Frank-Wolfe algorithm (a real-valued positive-semidefinite matrix  $M$ ) as a similarity matrix between vertices of two graphs, or use the incremental projection strategy fixing on each step the most probable matching and adjusting the optimum given the new constraint.

To favor the neighbor merging and to introduce local vertex similarities, we can use the same terms that we used before. Note,  $P_1P_1^T = M_1$ ,  $P_2P_2^T = M_2$  and  $P_1P_2^T = M_{12}$ , therefore new terms in (6) and (7) can be expressed as linear functions of  $M$

$$\text{tr}G^T M_1 + \text{tr}H^T M_2 \text{ and } \text{tr}C^T M_{12}$$

The addition of these new terms to the objective function  $F(P)$  does not change its structure,  $F(P)$  stays a quadratic function, so we can use the same minimization procedure.

A serious drawback of the ‘‘SDP’’ approach lies in its complexity. The complexity of a generic SDP solver scales as  $O(m^{2.5})$  [16] where  $m$  is the number of variables of a given ‘‘SDP’’ problem. In our case,  $m$  is equal to  $(N_G + N_H)^2$  which means that the complexity of the gradient minimization step is at least  $O((N_G + N_H)^5)$ . Hence, it is almost impossible to run the ‘‘SDP’’ method on graphs with more than 30 vertices. To overcome this problem we propose to use an early stopping rule i.e. replace the exact SDP solution by an approximate one.



## 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 [11] and [12]. 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 [2] as well as other examples of best-first, breadth-first and depth-first searches are used.

**Spectral approach.** Caelli and Kosinov [6] discuss how spectral matching may be used for many-to-many graph matching. Their algorithm is similar to the Umeyama method [5] 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

$$G = V_G \Lambda_G V_G^\top, \quad H = V_H \Lambda_H V_H^\top. \quad (10)$$

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 [13] 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 methods proposed in this paper with existing techniques (beam-search and spectral approach). We thus test four competitive approaches in several experiments: beam-search “Beam” (A\*-beam search from [2]), the spectral approach “Spec” [6] and two new approaches: the gradient descent method “Grad” (from Section 3.1), and the “SDP” relaxation (Section 3.2). All four algorithms are implemented in matlab and are available at <http://cbio.ensmp.fr/graphm/mtmgm.html>. We use SeDuMi (<http://sedumi.ie.lehigh.edu/>) to perform the gradient minimization in the “SDP” method.

### 5.1 Synthetic Examples

In this section, we compare the four 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 functions 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 (4) 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 four algorithms. In both cases, we observe that “Grad” and “SDP” significantly outperform both “Beam” and “Spec” with the “SDP” algorithm working slightly better than “Grad”. “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 [6]<sup>2</sup>. To maximally accelerate the “SDP” method, the number of iterations in SeDuMi was set to one, actually, one iteration is enough to have a good matching quality.

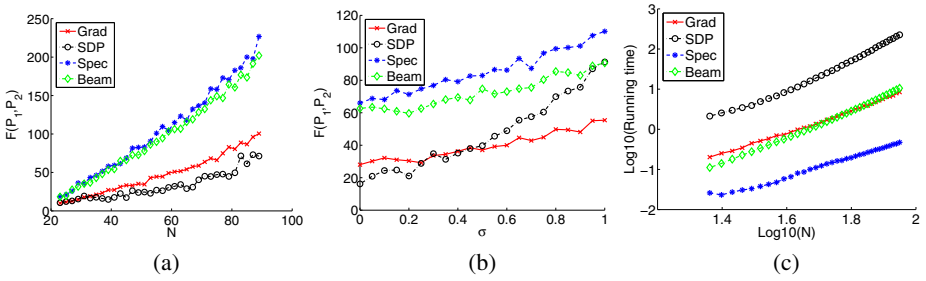
Figure 2c shows how algorithms scale in time with the graph size  $N$ . The “Spec” algorithm is the fastest one, but “Grad” has the same complexity order as “Spec” (corresponding curves are almost parallel lines in log-log scale, so both functions are polynomials with the same degree and different multiplication constants), these curves are coherent with theoretical values of algorithm complexity summarized in Section 3.1. The early stopping rule makes possible the use of the “SDP” algorithm on graphs with up to one hundred vertices, but it is still about 10 times slower than the “Grad” method, and almost 100 times slower than the “Spec” algorithm.

## 5.2 Chinese Characters

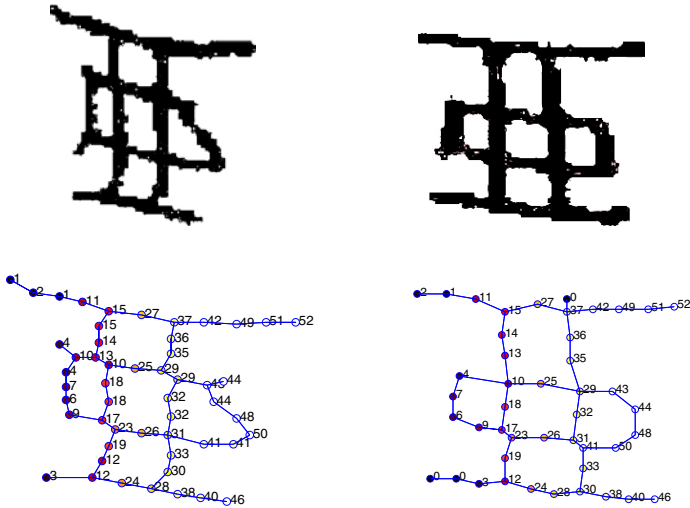
In this section we *quantitatively* compare many-to-many graph matching algorithms as parts of a classification framework. We use graph matching algorithms to compute similarity/distance between objects of interest on the basis of their graph-based representations. Our objective here is to see if with the new formulation of the many-to-many graph matching problem (4) and corresponding continuous relaxation algorithms improve the classification performance comparing to the existing state of the art algorithms. Since “Grad” and “Spec” are essentially two alternative approximate algorithms for the same discrete optimization problem with “Grad” working almost 10 times faster and providing the matching quality similar to that of the “SDP” method, we decided to test only the “Grad” algorithm. For example, to compute the similarity matrix of 600 graphs with in average 50 vertices, the running time of the “SDP” method would be about 280 hours ( $\approx 50$  seconds per pair).

---

<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.




**Fig. 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  $\approx 3.9$ ; “SDP” slope  $\approx 4.2$ , “Grad” slope  $\approx 2.9$ , “Spec” slope  $\approx 2.7$ .



**Fig. 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.

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 represented 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

**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)


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

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.

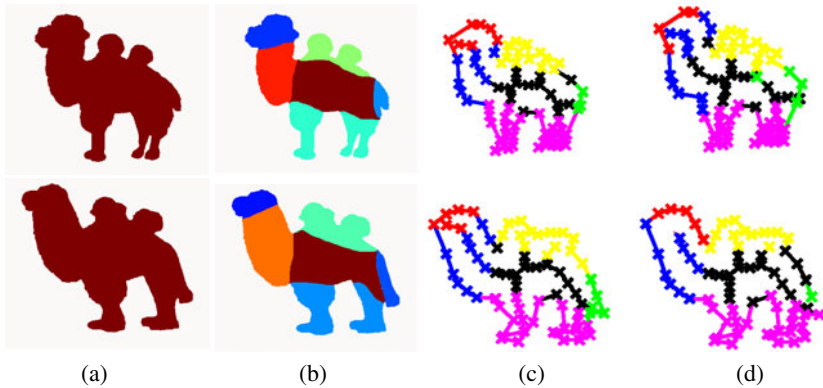
Table 1 shows classification results for the three many-to-many graph matching algorithms. In addition we report results for other popular approaches, namely, a SVM classifier with linear and Gaussian kernels, one-to-one matching with the Path algorithm (taken from [4]) and two versions of the shape context method [1], 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 (7). 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 (7) 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.

### 5.3 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 4). This image segmentation automatically defines a partitioning of the corresponding graph shown in the column (c) in Figure 4: all graph vertices are labeled according to the image part which they represent. Figure 4

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



**Fig. 4.** (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.

**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

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.

This experiment illustrates a promising application of graph matching algorithms. 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. (7)), with  $C_{ij} = e^{-(x_i-x_j)^2-(y_i-y_j)^2}$ . The parameter  $\lambda$  in (7)

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.

## 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 new approximate algorithms “Grad” and “SDP” based on two alternative continuous relaxation. The success of the proposed methods 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, “Grad” and “SDP” algorithms aim to optimize a clear objective function naturally representing the quality of graph matching instead of a sequence of unrelated steps. It is still difficult to run the “SDP” algorithm on real world datasets due to its time complexity, but we think a significant progress may be made by employing approximate SDP solvers. The hard limit on the number of iteration in SeDuMi is probably not the best way to find an approximate solution, we are planning to see other alternatives such as SDPA (<http://sdpa.indsys.chuo-u.ac.jp/sdpa/>) and CSDP (<https://projects.coin-or.org/Csdp/>). The reformulation of the gradient minimization as a linear assignment problem made possible the use of the “Grad” algorithm in large-scale graph matching problems. Probably, the special structure of the optimization set in the “SDP” relaxation may also represent an important clue for the further acceleration of the “SDP” algorithm. In particular, this direction seems to be interesting since “SDP” was able to find better matchings than “Grad” in numerical tests (Section 5.1). Another interesting direction is to try to construct a theoretical bound on the quality of the proposed approximate algorithms.

Besides a natural application of graph matching as a similarity measure between objects with complex structures, graph matching can also be used for object 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 many machine learning applications.

## Acknowledgments

This paper was supported in part by a grant from the Agence Nationale de la Recherche (MGA Project).

Mikhail Zaslavskiy thanks Collectis S.A.<sup>4</sup>, a genome engineering company, for kindly provided travel and registration grants.

## References

1. Belongie, S., Malik, J., Puzicha, J.: Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell.* 24(4), 509–522 (2002)

<sup>4</sup> [www.collectis.com](http://www.collectis.com)

2. Neuhaus, M., Riesen, K., Bunke, H.: Fast suboptimal algorithms for the computation of graph edit distance. In: Yeung, D.-Y., Kwok, J.T., Fred, A.L.N., Roli, F., de Ridder, D. (eds.) SSPR 2006 and SPR 2006. LNCS, vol. 4109, pp. 163–172. Springer, Heidelberg (2006)
3. Almohamad, H.A., Duffuaa, S.O.: A linear programming approach for the weighted graph matching problem. *IEEE Trans. Pattern Anal. Mach. Intell.* 15(5), 522–525 (1993)
4. Zaslavskiy, M., Bach, F., Vert, J.-P.: A path following algorithm for the graph matching problem. *IEEE Trans. Pattern Anal. Mach. Intell.* 31(12), 2227–2242 (2009)
5. Umeyama, S.: An eigendecomposition approach to weighted graph matching problems. *IEEE Trans. Pattern Anal. Mach. Intell.* 10(5), 695–703 (1988)
6. Caelli, T., Kosinov, S.: An eigenspace projection clustering method for inexact graph matching. *IEEE Trans. Pattern Anal. Mach. Intell.* 26(4), 515–519 (2004)
7. Carcassoni, M., Hancock, E.: Spectral correspondence for point pattern matching. *Pattern Recogn.* 36(1), 193–204 (2003)
8. Cour, T., Srinivasan, P., Shi, J.: Balanced graph matching. In: *Advanced in Neural Information Processing Systems* (2006)
9. Leordeanu, M., Hebert, M.: A spectral technique for correspondence problems using pairwise constraints. In: *International Conference of Computer Vision (ICCV)*, vol. 2, pp. 1482–1489 (October 2005)
10. Duchenne, O., Bach, F., Kweon, I., Ponce, J.: A tensor-based algorithm for high-order graph matching. In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition CVPR 2009*, June 20–25, pp. 1980–1987 (2009)
11. Berretti, S., Del Bimbo, A., Pala, P.: A graph edit distance based on node merging. In: *Proc. of ACM International Conference on Image and Video Retrieval (CIVR)*, Dublin, Ireland, July 2004, pp. 464–472 (2004)
12. Ambauen, R., Fischer, S., Bunke, H.: Graph edit distance with node splitting and merging, and its application to diatom identification. In: *GbRPR*, pp. 95–106 (2003)
13. Keselman, Y., Shokoufandeh, A., Demirci, M.F., Dickinson, S.: Many-to-many graph matching via metric embedding. In: *CVPR*, pp. 850–857 (2003)
14. Bertsekas, D.: *Nonlinear programming*. Athena Scientific (1999)
15. Kuhn, H.W.: The Hungarian method for the assignment problem. *Naval Research* 2, 83–97 (1955)
16. Nesterov, Y., Nemirovsky, A.: *Interior point polynomial methods in convex programming: Theory and applications*. SIAM, Philadelphia (1994)

## Appendix A

Here we present how the gradient minimization step in the “Grad” method can be reformulated as a linear assignment problem. Let  $\nabla F_1$  and  $\nabla F_2$  denote the gradient of the function  $F(P_1, P_2)$  with respect to matrices  $P_1$  and  $P_2$  ( $\nabla F_1$  is a  $N_K \times N_G$  matrix and  $\nabla F_2$  is a  $N_K \times N_H$  matrix). Recall that our objective is to minimize  $\text{tr} F_1^T P_1 + \text{tr} F_2^T P_2$  over (4). Note that two terms of the gradient are independent linear functions and can be minimized one by one. Let us consider the first term (we drop the subscript  $_1$  for simplicity)

$$\begin{aligned} \min_P \text{tr} \nabla F^T P \quad \text{subject to} \\ P \in \{0, 1\}^{N_K \times N_G}, P \mathbf{1}_{N_G} \leq k_{max} \mathbf{1}_{N_K}, P^T \mathbf{1}_{N_K} = \mathbf{1}_{N_G}. \end{aligned} \quad (11)$$

Matrix  $P$  is a  $N_K \times N_G$  binary matrix with up to  $k_{max}$  “ones” in each row, and one and only one “one” in each column. Let  $Q$  be a  $k_{max} N_K \times N_G$  binary matrix with up to one

“one” in each row (may be zero), and exactly one “one” in each column.  $Q$  may be seen as a splitted version of matrix  $P$ , the first  $k_{max}$  rows of the matrix  $Q$  correspond to the first row of  $P$ , then rows with indexes  $k_{max} + 1, \dots, 2k_{max}$  correspond to the second row of  $P$  and so on. We can always construct  $P$  from  $Q$  by merging corresponding rows, the reverse operation corresponds to splitting the rows of matrix  $P$ . We will write  $P \leftrightarrow Q$  to denote pairs  $(P, Q)$  which may be transformed to each other by merging/splitting operations, of course the same matrix  $P$  may correspond to many matrices  $Q$ 's. Now, let  $F_q$  denote a  $k_{max}N_K \times N_G$  real valued matrix constructed from the matrix  $F$  by duplicating every row  $k_{max}$  times i.e. first  $k_{max}$  rows of  $F_q$  are copies of the first row of  $F$  and so on.

This is easy to see, that if  $P \leftrightarrow Q$  then  $\text{tr}F_q^T Q = \text{tr}F^T P$  (the left side is just a splitted version of the right side) and therefore if  $P^* = \arg \min_P \text{tr}F^T P$  and  $Q^* \leftrightarrow P^*$  then  $Q^* = \arg \min_Q \text{tr}F_q^T Q$  and vice versa. Indeed, if  $Q^* \neq \arg \min_Q \text{tr}F_q^T Q$  then  $\exists Q^+$  such that  $\text{tr}F_q^T Q^+ < \text{tr}F_q^T Q^*$ , then we can construct  $P^+ \leftrightarrow Q^+$  such that  $\text{tr}F^T P^+ < \text{tr}F^T P^*$ .

We showed that  $\min_P F^T P$  is equivalent to  $\min_Q F_q^T Q$  which is nothing else than a linear assignment problem. Now, we can run the Hungarian algorithm to minimize  $F_q^T Q$  and then transform the optimal  $Q$ -solution to a  $P$ -solution by merging corresponding rows.