A Path Following Algorithm for the Graph Matching Problem

Mikhail Zaslavskiy, Francis Bach, and Jean-Philippe Vert

Abstract—We propose a convex-concave programming approach for the labeled weighted graph matching problem. The convex-concave programming formulation is obtained by rewriting the weighted graph matching problem as a least-square problem on the set of permutation matrices and relaxing it to two different optimization problems: a quadratic convex and a quadratic concave optimization problem on the set of doubly stochastic matrices. The concave relaxation has the same global minimum as the initial graph matching problem, but the search for its global minimum is also a hard combinatorial problem. We, therefore, construct an approximation of the concave problem solution by following a solution path of a convex-concave problem obtained by linear interpolation of the convex and concave formulations, starting from the convex relaxation. This method allows to easily integrate the information on graph label similarities into the optimization problem, and therefore, perform labeled weighted graph matching. The algorithm is compared with some of the best performing graph matching methods on four data sets: simulated graphs, QAPLib, retina vessel images, and handwritten Chinese characters. In all cases, the results are competitive with the state of the art.

Index Terms—Graph algorithms, graph matching, convex programming, gradient methods, machine learning, classification, image processing.

✦

---

1 INTRODUCTION

THE graph matching problem is among the most important challenges of graph processing and plays a central role in various fields of pattern recognition. Roughly speaking, the problem consists in finding a correspondence between vertices of two given graphs, which is optimal in some sense. Usually, the optimality refers to the alignment of graph structures and, when available, of vertices labels, although other criteria are possible as well. A nonexhaustive list of graph matching applications includes document processing tasks like optical character recognition [1], [2], image analysis (2D and 3D) [3], [4], [5], [6], or bioinformatics [7], [8], [9].

During the last decade, many different algorithms for graph matching have been proposed. Because of the combinatorial nature of this problem, it is very hard to solve it exactly for large graphs, however, some methods based on incomplete enumeration may be applied to search for an exact optimal solution in the case of small or sparse graphs. Some examples of such algorithms may be found in [10], [11], [12].

- M. Zaslavskiy is with the Centre for Computational Biology and the Centre for Mathematical Morphology, Mines ParisTech, 35 rue Saint-Honore, 77305 Fontainebleau cedex, France, and the Institute Curie—INSER-M—U900, 26 rue d'Ulm, 75248 Paris cedex 05, France.
  E-mail: mikhail.zaslavskiy@ensmp.fr.
- F. Bach is with the INRIA—Willow Project Team, Laboratoire d'Informatique de l'Ecole Normale Supérieure (CNRS/ENS/INRIA UMR 8548), 23 avenue d'Italie, 75214 Paris, France. E-mail: francis.bach@inria.fr.
- J.-P. Vert is with the Centre for Computational Biology, Mines ParisTech, 35 rue Saint-Honore, 77305 Fontainebleau cedex, France, and the Institut Curie—INSERM—U900, 26 rue d'Ulm, 75248 Paris cedex 05, France.
  E-mail: Jean-Philippe.Vert@mines.org.

Another group of methods includes approximate algorithms that are supposed to be more scalable. The price to pay for the scalability is that the solution found is usually only an approximation of the optimal matching. Approximate methods may be divided into two groups of algorithms. The first group is composed of methods, which use spectral representations of adjacency matrices or equivalently embed the vertices into a euclidean space, where linear or nonlinear matching algorithms can be deployed. This approach was pioneered by Umeyama [13], while further different methods based on spectral representations were proposed in [3], [4], [5], [14], [15]. The second group of approximate algorithms is composed of algorithms which work directly with graph adjacency matrices and typically involve a relaxation of the discrete optimization problem. The most effective algorithms were proposed in [6], [16], [17], [18].

An interesting instance of the graph matching problem is the matching of labeled graphs. In that case, graph vertices have associated labels, which may be numbers, categorical variables, etc. The important point is that there is also a similarity measure between these labels. Therefore, when we search for the optimal correspondence between vertices, we search a correspondence which matches not only the structures of the graphs but also vertices with similar labels. Some widely used approaches for this application only use the information about similarities between graph labels. In vision, one such algorithm is the shape context algorithm proposed in [19], which involves an efficient algorithm of node label construction. Another example is the BLAST-based algorithms in bioinformatics such as the Inparanoid algorithm [20], where correspondence between different protein networks is established on the basis of BLAST scores between pairs of proteins. The main advantages of all these methods are their speed and simplicity. However, the

main drawback of these methods is that they do not take into account information about the graph structure. Some graph matching methods try to combine information on graph structures and vertex similarities, examples of such method are presented in [7], [18].

In this paper, we propose an approximate method for labeled weighted graph matching, based on a convex-concave programming approach, which can be applied for matching of graphs of large sizes. Our method is based on a formulation of the labeled weighted graph matching problem as a quadratic assignment problem (QAP) over the set of permutation matrices, where the quadratic term encodes the structural compatibility and the linear term encodes local compatibilities. We propose two relaxations of this problem, resulting in one quadratic convex and one quadratic concave minimization problem on the set of doubly stochastic matrices. While the concave relaxation has the same global minimum as the initial QAP, solving it is also a hard combinatorial problem. We find a local minimum of this problem by following a solution path of a family of convex-concave minimization problems, obtained by linearly interpolating between the convex and concave relaxations. Starting from the convex formulation with a unique local (and global) minimum, the solution path leads to a local optimum of the concave relaxation. We refer to this procedure as the PATH algorithm.[1] We perform an extensive comparison of this PATH algorithm with several state-of-the-art matching methods on small simulated graphs and various QAP benchmarks, and show that it consistently provides state-of-the-art performances while scaling to graphs of up to a few thousand vertices on a modern desktop computer. We further illustrate the use of the algorithm on two applications in image processing, namely the matching of retina images based on vessel organization, and the matching of handwritten Chinese characters.

The rest of the paper is organized as follows: Section 2 presents the mathematical formulation of the graph matching problem. In Section 3, we present our new approach. Then, in Section 4, we present the comparison of our method with Umeyama's algorithm [13] and the linear programming approach [16] on the example of artificially simulated graphs. In Section 5, we test our algorithm on the QAP benchmark library, and compare obtained results with the results published in [18] for the QBP algorithm and graduated assignment algorithms. Finally, in Section 6, we present two examples of applications to the real-world image processing tasks.

2   PROBLEM DESCRIPTION

A graph $G = (V, E)$ of size N is defined by a finite set of vertices $V = \{1, \ldots, N\}$ and a set of edges $E \subset V \times V$. We consider only undirected graphs with no self-loop, i.e., such that $(i, j) \in E \Longrightarrow (j, i) \in E$ and $(i, i) \notin E$ for any vertices $i, j \in V$. Each such graph can be equivalently represented by a symmetric adjacency matrix A of size $|V| \times |V|$, where A_{ij} is equal to one if there is an edge between vertices i and

j, and zero otherwise. An interesting generalization is a weighted graph which is defined by association of nonnegative real values w_{ij} (weights) to all edges of graph G. Such graphs are represented by real-valued adjacency matrices A with $A_{ij} = w_{ij}$. This generalization is important because in many applications, the graphs of interest have associated weights for all their edges, and taking into account, these weights may be crucial in construction of efficient methods. In the following, when we talk about "adjacency matrix," we mean a real-valued "weighted" adjacency matrix.

Given two graphs G and H with the same number of vertices N, the problem of matching G and H consists in finding a correspondence between vertices of G and vertices of H, which aligns G and H in some optimal way. We will consider in Section 3.8 an extension of the problem to graphs of different sizes. For graphs with the same size N, the correspondence between vertices is a permutation of N vertices, which can be defined by a permutation matrix P, i.e., a $\{0, 1\}$-valued $N \times N$ matrix with exactly one entry 1 in each column and each row. The matrix P entirely defines the mapping between vertices of G and vertices of H, P_{ij} being equal to 1 if the ith vertex of G is matched to the jth vertex of H, and 0 otherwise. After applying the permutation defined by P to the vertices of H, we obtain a new graph isomorphic to H, which we denote by $P(H)$. The adjacency matrix of the permuted graph $A_{P(H)}$ is simply obtained from A_H by the equality $A_{P(H)} = PA_HP^T$.

In order to assess whether a permutation P defines a good matching between the vertices of G and those of H, a quality criterion must be defined. Although other choices are possible, we focus in this paper on measuring the discrepancy between the graphs after matching, by the number of edges (in the case of weighted graphs, it will be the total weight of edges), which are present in one graph and not in the other. In terms of adjacency matrices, this number can be computed as

$$F_0(P) = \|A_G - A_{P(H)}\|_F^2 = \|A_G - PA_HP^T\|_F^2, \qquad (1)$$

where $\|.\|_F$ is the Frobenius matrix norm defined by $\|A\|_F^2 = \text{tr} A^T A = (\sum_i \sum_j A_{ij}^2)$. A popular alternative to the Frobenius norm formulation (1) is the 1-norm formulation obtained by replacing the Frobenius norm by the 1-norm $\|A\|_1 = \sum_i \sum_j |A_{ij}|$, which is equal to the square of the Frobenius norm $\|A\|_F^2$ when comparing $\{0, 1\}$-valued matrices, but may differ in the case of general matrices.

Therefore, the problem of graph matching can be reformulated as the problem of minimizing $F_0(P)$ over the set of permutation matrices. This problem has a combinatorial nature and there is no known polynomial algorithm to solve it [21]. It is therefore very hard to solve it in the case of large graphs and numerous approximate methods have been developed.

An interesting generalization of the graph matching problem is the problem of *labeled* graph matching. Here, each graph has associated labels to all its vertices and the objective is to find an alignment that fits well graph labels and graph structures at the same time. If we let C_{ij} denote the cost of fitness between the ith vertex of G and jth vertex

of H, then the matching problem based only on label comparison can be formulated as follows:

$$\min_{P \in \mathcal{P}} \; \mathrm{tr}(C^T P) = \sum_{i=1}^{N} \sum_{j=1}^{N} C_{ij} P_{ij}, \qquad (2)$$

where $\mathcal{P}$ denotes the set of permutation matrices. A natural way of unifying (2) and (1) to match both the graph structure and the vertices' labels is then to minimize a convex combination [18]:

$$\min_{P \in \mathcal{P}} \; (1 - \alpha) F_0(P) + \alpha \mathrm{tr}(C^T P), \qquad (3)$$

which makes explicit, through the parameter $\alpha \in [0, 1]$, the trade-off between cost of individual matchings and faithfulness to the graph structure. A small α value emphasizes the matching of structures, while a large α value gives more importance to the matching of labels.

2.1 Permutation Matrices

Before describing how we propose to solve (1) and (3), we first introduce some notations and bring to notice some important characteristics of these optimization problems. They are defined on the set of permutation matrices, which we denoted by $\mathcal{P}$. The set $\mathcal{P}$ is a set of extreme points of the set of doubly stochastic matrices, that is, matrices with nonnegative entries and with row sums and column sums equal to one: $\mathcal{D} = \{A : A1_N = 1_N, A^T 1_N = 1_N, A \geq 0\}$, where 1_N denotes the N-dimensional vector of all ones [22]. This shows that, when a linear function is minimized over the set of doubly stochastic matrices $\mathcal{D}$, a solution can always be found in the set of permutation matrices. Consequently, minimizing a linear function over $\mathcal{P}$ is in fact equivalent to a linear program and can thus be solved in polynomial time by, e.g., interior point methods [23]. In fact, one of the most efficient methods to solve this problem is the Hungarian algorithm, which uses a specific primal-dual strategy to solve this problem in $O(N^3)$ [24]. Note that the Hungarian algorithm allows to avoid the generic $O(N^7)$ complexity associated with a linear program with N^2 variables.

At the same time, $\mathcal{P}$ may be considered as a subset of orthonormal matrices $\mathcal{O} = \{A : A^T A = I\}$ of $\mathcal{D}$ and in fact $\mathcal{P} = \mathcal{D} \cap \mathcal{O}$. An (idealized) illustration of these sets is presented in Fig. 1: The discrete set $\mathcal{P}$ of permutation matrices is the intersection of the convex set $\mathcal{D}$ of doubly stochastic matrices and the manifold $\mathcal{O}$ of orthogonal matrices.

2.2 Approximate Methods: Existing Works

A good review of graph matching algorithms may be found in [25]. Here, we only present a brief description of some approximate methods that illustrate well ideas behind two subgroups of these algorithms. As mentioned in Section 1, one popular approach to find approximate solutions to the graph matching problem is based on the spectral decomposition of the adjacency matrices of the graphs to be matched. In this approach, the singular value decompositions of the graph adjacency matrices are used:

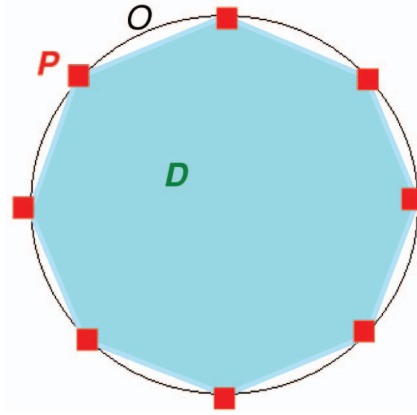$$A_G = U_G \Lambda_G U_G^T, \quad A_H = U_H \Lambda_H U_H^T,$$

Fig. 1. Relation between three matrix sets. $\mathcal{O}$—set of orthogonal matrices, $\mathcal{D}$—set of doubly stochastic matrices, and $\mathcal{P} = \mathcal{D} \cap \mathcal{O}$—set of permutation matrices.

where the columns of the orthogonal matrices U_G and U_H consist of eigenvectors of A_G and A_H, respectively, and Λ_G and Λ_H are diagonal matrices of eigenvalues.

If we consider the rows of eigenvector matrices U_G and U_H as graph node coordinates in eigenspaces, then we can match the vertices with similar coordinates through a variety of methods [5], [13], [15]. However, these methods suffer from the fact that the spectral embedding of graph vertices is not uniquely defined. First, the unit norm eigenvectors are at most defined up to a sign flip and we have to choose their signs synchronously. Although it is possible to use some normalization convention, such as choosing the sign of each eigenvector in such a way that the biggest component is always positive, this usually does not guarantee a perfect sign synchronization, in particular, in the presence of noise. Second, if the adjacency matrix has multiple eigenvalues, then the choice of eigenvectors becomes arbitrary within the corresponding eigensubspace, as they are defined only up to rotations [26].

One of the first spectral approximate algorithms was presented by Umeyama [13]. To avoid the ambiguity of eigenvector selection, Umeyama proposed to consider the absolute values of eigenvectors. According to this approach, the correspondence between graph nodes is established by matching the rows of $|U_G|$ and $|U_H|$ (which are defined as matrices of absolute values). The criterion of optimal matching is the total distance between matched rows, leading to the optimization problem

$$\min_{P \in \mathcal{P}} \; \| |U_G| - P |U_H| \|_F$$

or, equivalently,

$$\max_{P \in \mathcal{P}} \; \mathrm{tr}(|U_H| |U_G|^T P). \qquad (4)$$

The optimization problem (4) is a linear program on the set of permutation matrices, which can be solved by the Hungarian algorithm in $O(N^3)$ [27], [28].

The second group of approximate methods consists of algorithms that work directly with the objective function in (1) and typically involve various relaxations to optimizations problems that can efficiently be solved. An example of such an approach is the linear programming method proposed by Almohamad and Duffuaa in [16]. They

considered the 1-norm as the matching criterion for a permutation matrix $P \in \mathcal{P}$:

$$F_0'(P) = \|A_G - PA_HP^T\|_1 = \|A_GP - PA_H\|_1.$$

The linear program relaxation is obtained by optimizing $F_0'(P)$ on the set of doubly stochastic matrices $\mathcal{D}$ instead of $\mathcal{P}$:

$$\min_{P \in \mathcal{D}} F_0'(P), \qquad (5)$$

where the 1-norm of a matrix is defined as the sum of the absolute values of all the elements of a matrix. A priori the solution of (5) is an arbitrary doubly stochastic matrix $X \in \mathcal{D}$, so the final step is a projection of X on the set of permutation matrices (we let denote $\Pi_{\mathcal{P}}X$ the projection of X onto $\mathcal{P}$):

$$P^* = \Pi_{\mathcal{P}}X = \arg\min_{P \in \mathcal{P}} \|P - X\|_F^2$$

or, equivalently,

$$P^* = \arg\max_{P \in \mathcal{P}} X^T P. \qquad (6)$$

The projection (6) can be performed with the Hungarian algorithm, with a complexity cubic in the dimension of the problem. The main disadvantage of this method is that the dimensionality (i.e., number of variables and number of constraints) of the linear program (6) is $O(N^2)$, and therefore, it is quite hard to process graphs of size more than 100 nodes.

Other convex relaxations of (1) can be found in [18] and [17]. In the next section, we describe our new algorithm which is based on the technique of convex-concave relaxations of the initial problems (1) and (3).

3 CONVEX-CONCAVE RELAXATION

Let us start the description of our algorithm for unlabeled weighted graphs. The generalization to labeled weighted graphs is presented in Section 3.7. The graph matching criterion we consider for unlabeled graphs is the square of the Frobenius norm of the difference between adjacency matrices (1). Since permutation matrices are also orthogonal matrices (i.e., $PP^T = I$ and $P^TP = I$), we can rewrite $F_0(P)$ on $\mathcal{P}$ as follows:

$$F_0(P) = \|A_G - PA_HP^T\|_F^2 = \|(A_G - PA_HP^T)P\|_F^2$$
$$= \|A_GP - PA_H\|_F^2.$$

The graph matching problem is then the problem of minimizing $F_0(P)$ over $\mathcal{P}$, which we call **GM**:

$$\textbf{GM}: \min_{P \in \mathcal{P}} F_0(P). \qquad (7)$$

3.1 Convex Relaxation

A first relaxation of **GM** is obtained by expanding the convex quadratic function $F_0(P)$ on the set of doubly stochastic matrices $\mathcal{D}$:

$$\textbf{QCV}: \min_{P \in \mathcal{D}} F_0(P). \qquad (8)$$

The **QCV** problem is a convex quadratic program that can be solved in polynomial time, e.g., by the Frank-Wolfe algorithm [29] (see Section 3.5 for more details). However, the optimal value is usually not an extreme point of $\mathcal{D}$, and

therefore, not a permutation matrix. If we want to use only **QCV** for the graph matching problem, we therefore have to project its solution on the set of permutation matrices, and to make, e.g., the following approximation:

$$\arg\min_{\mathcal{P}} F_0(P) \approx \Pi_{\mathcal{P}} \arg\min_{\mathcal{D}} F_0(P). \qquad (9)$$

Although the projection $\Pi_{\mathcal{P}}$ can be made efficiently in $O(N^3)$ by the Hungarian algorithm, the difficulty with this approach is that if $\arg\min_{\mathcal{D}} F_0(P)$ is far from $\mathcal{P}$, then the quality of the approximation (9) may be poor: In other words, the work performed to optimize $F_0(P)$ is partly lost by the projection step, which is independent of the cost function. The PATH algorithm that we present later can be thought of as an improved projection step that takes into account the cost function in the projection.

3.2 Concave Relaxation

We now present a second relaxation of **GM**, which results in a concave minimization problem. For that purpose, let us introduce the diagonal degree matrix D of an adjacency matrix A, which is the diagonal matrix with entries $D_{ii} = d(i) = \sum_{i=1}^{N} A_{ij}$ for $i = 1, \ldots, N$, as well as the Laplacian matrix $L = D - A$. A having only nonnegative entries, it is well known that the Laplacian matrix is positive semidefinite [30]. We can now rewrite $F_0(P)$ as follows:

$$F_0(P) = \|A_GP - PA_H\|_F^2$$
$$= \|(D_GP - PD_H) - (L_GP - PL_H)\|_F^2$$
$$= \|D_GP - PD_H\|_F^2 \qquad (10)$$
$$\quad - 2\text{tr}[(D_GP - PD_H)^T(L_GP - PL_H)]$$
$$\quad + \|L_GP - PL_H\|_F^2.$$

Let us now consider more precisely the second term in this last expression:

$$\text{tr}[(D_GP - PD_H)^T(L_GP - PL_H)]$$
$$= \underbrace{\text{tr}PP^TD_G^TL_G}_{\sum d_G^2(i)} + \underbrace{\text{tr}L_HD_H^TP^TP}_{\sum d_H^2(i)} - \underbrace{\text{tr}P^TD_G^TPL_H}_{\sum d_G(i)d_{P(H)}(i)}$$
$$\quad - \underbrace{\text{tr}D_H^TP^TL_GP}_{\sum d_{P(H)}(i)d_G(i)} \qquad (11)$$
$$= \sum (d_G(i) - d_{P(H)}(i))^2 = \|D_G - D_{P(H)}\|_F^2$$
$$= \|D_GP - PD_H\|_F^2.$$

By plugging (11) into (10), we obtain

$$F_0(P) = \|D_GP - PD_H\|_F^2 - 2\|D_GP - PD_H\|_F^2$$
$$\quad + \|L_GP - PL_H\|_F^2$$
$$= -\|D_GP - PD_H\|_F^2 + \|L_GP - PL_H\|_F^2$$
$$= -\sum_{i,j} P_{ij}(D_G(j) - D_H(i))^2 + \text{tr}(\underbrace{PP^T}_{I}L_G^TL_G)$$
$$\quad + \text{tr}(L_H^T\underbrace{P^TP}_{I}L_H) - 2\underbrace{\text{tr}(P^TL_G^TPL_H)}_{\text{vec}(P)^T(L_H^T \otimes L_G^T)\text{vec}(P)} \qquad (12)$$
$$= -\text{tr}(\Delta P) + \text{tr}(L_G^2) + \text{tr}(L_H^2)$$
$$\quad - 2\text{vec}(P)^T(L_H^T \otimes L_G^T)\text{vec}(P),$$

1) **Initialization**:
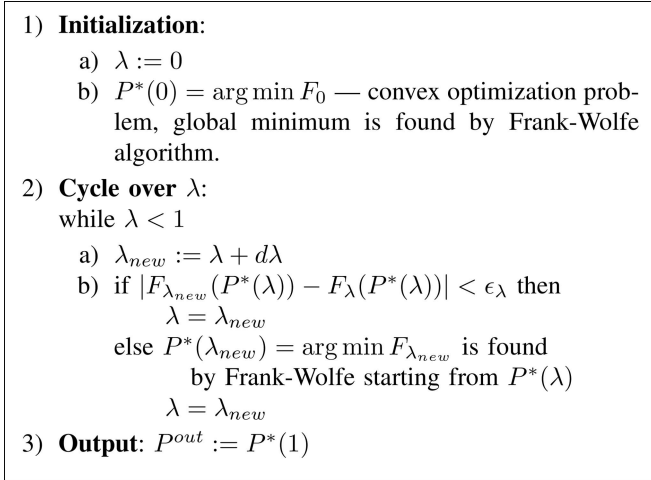   a) $\lambda := 0$
   b) $P^*(0) = \arg\min F_0$ — convex optimization problem, global minimum is found by Frank-Wolfe algorithm.

2) **Cycle over λ**:
   while $\lambda < 1$
   a) $\lambda_{new} := \lambda + d\lambda$
   b) if $|F_{\lambda_{new}}(P^*(\lambda)) - F_\lambda(P^*(\lambda))| < \epsilon_\lambda$ then
      $\lambda = \lambda_{new}$
      else $P^*(\lambda_{new}) = \arg\min F_{\lambda_{new}}$ is found
         by Frank-Wolfe starting from $P^*(\lambda)$
      $\lambda = \lambda_{new}$

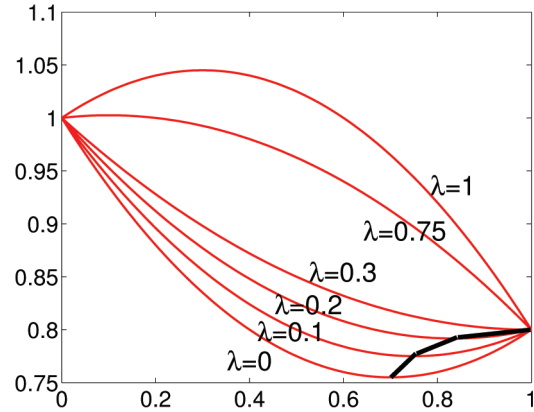3) **Output**: $P^{out} := P^*(1)$

Fig. 2. Schema of the PATH algorithm.

Fig. 3. Illustration for path optimization approach. F_0 ($\lambda = 0$)—initial convex function, F_1 ($\lambda = 1$)—initial concave function, and bold black line—path of function minima $P^*(\lambda)$ ($\lambda = 0 \ldots 0.1 \ldots 0.2 \ldots 0.3 \ldots 0.75 \ldots 1$).

where we introduced the matrix $\Delta_{i,j} = (D_H(j,j) - D_G(i,i))^2$ and used $\otimes$ to denote the Kronecker product of two matrices (definition of the Kronecker product and some important properties may be found in Appendix B).

Let us denote $F_1(P)$ the part of (12), which depends on P:

$$F_1(P) = -\text{tr}(\Delta P) - 2\text{vec}(P)^T(L_H^T \otimes L_G^T)\text{vec}(P).$$

From (12), we see that the permutation matrix which minimizes F_1 over $\mathcal{P}$ is the solution of the graph matching problem. Now, minimizing $F_1(P)$ over $\mathcal{D}$ gives us a relaxation of (7) on the set of doubly stochastic matrices. Since graph Laplacian matrices are positive semidefinite, the matrix $L_H \otimes L_G$ is also positive semidefinite as a Kronecker product of two symmetric positive semidefinite matrices [26]. Therefore, the function $F_1(P)$ is concave on $\mathcal{D}$ and we obtain a concave relaxation of the graph matching problem

$$\textbf{QCC}: \min_{P \in \mathcal{D}} F_1(P). \tag{13}$$

Interestingly, the global minimum of a concave function is necessarily located at a boundary of the convex set, where it is minimized [31], so the minimum of $F_1(P)$ on $\mathcal{D}$ is in fact in $\mathcal{P}$.

At this point, we have obtained two relaxations of **GM** as nonlinear optimization problems on $\mathcal{D}$: The first one is the convex minimization problem **QCV** (8), which can be solved efficiently but leads to a solution in $\mathcal{D}$ that must then be projected onto $\mathcal{P}$, and the other is the concave minimization problem **QCC** (13), which does not have an efficient (polynomial) optimization algorithm but has the same solution as the initial problem **GM**. We note that these convex and concave relaxations of the graph matching problem can be more generally derived for any quadratic assignment problems [32].

3.3 PATH Algorithm

We propose to approximately solve **QCC** by tracking a path of local minima over $\mathcal{D}$ of a series of functions that linearly interpolate between $F_0(P)$ and $F_1(P)$, namely,

$$F_\lambda(P) = (1 - \lambda)F_0(P) + \lambda F_1(P),$$

for $0 \le \lambda \le 1$. For all $\lambda \in [0,1]$, F_λ is a quadratic function (which is, in general, neither convex nor concave for λ away

from zero or one). We recover the convex function F_0 for $\lambda = 0$ and the concave function F_1 for $\lambda = 1$. Our method searches sequentially local minima of F_λ, where λ moves from 0 to 1. More precisely, we start at $\lambda = 0$, and find the unique local minimum of F_0 (which is, in this case, its unique global minimum) by any classical QP solver. Then, iteratively, we find a local minimum of $F_{\lambda+d\lambda}$ given a local minimum of F_λ by performing a local optimization of $F_{\lambda+d\lambda}$ starting from the local minimum of F_λ, using, for example, the Frank-Wolfe algorithm. Repeating this iterative process for $d\lambda$ small enough, we obtain a path of solutions $P^*(\lambda)$, where $P^*(0) = \arg\min_{P \in \mathcal{D}} F_0(P)$ and $P^*(\lambda)$ is a local minimum of F_λ for all $\lambda \in [0,1]$. Noting that any local minimum of the concave function F_1 on $\mathcal{D}$ is in $\mathcal{P}$, we finally output $P^*(1) \in \mathcal{P}$ as an approximate solution of **GM**.

Our approach is similar to graduated nonconvexity [33]: This approach is often used to approximate the global minimum of a nonconvex objective function. This function consists of two parts, the convex component and nonconvex component, and the graduated nonconvexity framework proposes to track the linear combination of the convex and nonconvex parts (from the convex relaxation to the true objective function) to approximate the minimum of the nonconvex function. The PATH algorithm may indeed be considered as an example of such an approach. However, the main difference is the construction of the objective function. Unlike [33], we construct two relaxations of the initial optimization problem, which lead to the same value on the set of interest ($\mathcal{P}$), the goal being to choose convex/concave relaxations that approximate in the best way the objective function on the set of permutation matrices.

The pseudocode for the PATH algorithm is presented in Fig. 2. The rationale behind it is that among the local minima of $F_1(P)$ on $\mathcal{D}$, we expect the one connected to the global minimum of F_0 through a path of local minima to be a good approximation of the global minima. Such a situation is, for example, shown in Fig. 3, where in one-dimension, the global minimum of a concave quadratic function on an interval (among two candidate points) can be found by following the path of local minima connected to the unique global minimum of a convex function.

More precisely, and although we do not have any formal result about the optimality of the PATH optimization method (beyond the lack of global optimality, see Appendix A), we can mention a few interesting properties of this method as follows:

- We know from (12) that for $P \in \mathcal{P}$, $F_1(P) = F_0(P) - \kappa$, where $\kappa = \text{tr}(L_G^2) + \text{tr}(L_H^2)$ is a constant independent of P. As a result, it holds for all $\lambda \in [0, 1]$ that, for $P \in \mathcal{P}$:

$$F_\lambda(P) = F_0(P) - \lambda\kappa.$$

  This shows that if for some λ the global minimum of $F_\lambda(P)$ over $\mathcal{D}$ lies in $\mathcal{P}$, then this minimum is also the global minimum of $F_0(P)$ over $\mathcal{P}$, and therefore, the optimal solution of the initial problem. Hence, if, for example, the global minimum of F_λ is found on $\mathcal{P}$ by the PATH algorithm (for instance, if F_λ is still convex), then the PATH algorithm leads to the global optimum of F_1. This situation can be seen in Fig. 3, where, for $\lambda = 0.3$, F_λ has its unique minimum at the boundary of the domain.

- The suboptimality of the PATH algorithm comes from the fact that, when λ increases, the number of local minima of F_λ may increase and the sequence of local minima tracked by PATH may not be global minima. However, we can expect the local minima followed by the PATH algorithm to be interesting approximations for the following reason. First, observe that if P_1 and P_2 are two local minima of F_λ for some $\lambda \in [0, 1]$, then the restriction of F_λ to (P_1, P_2) being a quadratic function it has to be concave and P_1 and P_2 must be on the boundary of $\mathcal{D}$. Now, let λ_1 be the smallest λ such that F_λ has several local minima on $\mathcal{D}$. If P_1 denotes the local minima followed by the PATH algorithm and P_2 denotes the "new" local minimum of F_{λ_1}, then necessarily the restriction of F_{λ_1} to (P_1, P_2) must be concave and have a vanishing derivative in P_2 (otherwise, by continuity of F_λ in λ, there would be a local minimum of F_λ near P_2 for λ slightly smaller than λ_1). Consequently, we necessarily have $F_{\lambda_1}(P_1) < F_{\lambda_1}(P_2)$. This situation is illustrated in Fig. 3, where, when the second local minimum appears for $\lambda = 0.75$, it is worse than the one tracked by the PATH algorithm. More generally, when "new" local minima appear, they are strictly worse than the one tracked by the PATH algorithm. Of course, they may become better than the PATH solution when λ continues to increase.

Of course, in spite of these justifications, the PATH algorithm only gives an approximation of the global minimum in the general case. In Appendix A, we provide two simple examples when the PATH algorithm, respectively, succeeds and fails to find the global minimum of the graph matching problem.

3.4 Numerical Continuation Method Interpretation

Our path following algorithm may be considered as a particular case of numerical continuation methods (sometimes called path following methods) [34]. These allow to estimate curves given in the following implicit form:

$$T(u) = 0, \text{ where T is a mapping } T : R^{K+1} \to R^K. \quad (14)$$

In fact, our PATH algorithm corresponds to a particular implementation of the so-called Generic Predictor Corrector Approach [34] widely used in numerical continuation methods.

In our case, we have a set of problems $\min_{P \in \mathcal{D}} (1 - \lambda)F_0(P) + \lambda F_1(P)$ parameterized by $\lambda \in [0, 1]$. In other words, for each λ, we have to solve the following system of Karush-Kuhn-Tucker (KKT) equations:

$$(1 - \lambda)\nabla_P F_0(P) + \lambda\nabla_P F_1(P) + B^T\nu + \mu_S = 0,$$
$$BP - 1_{2N} = 0,$$
$$P_S = 0,$$

where S is a set of active constraints, i.e., of pairs of indices (i, j) that satisfy $P_{ij} = 0$, $BP - 1_{2N} = 0$ codes the conditions $\sum_j P_{ij} = 1 \; \forall i$ and $\sum_i P_{ij} = 1 \; \forall j$, ν and μ_S are dual variables. We have to solve this system for all possible sets of active constraints S on the open set of matrices P that satisfy $P_{i,j} > 0$ for $(i, j) \notin S$, in order to define the set of stationary points of the functions F_λ. Now if we let $T(P, \nu, \mu, \lambda)$ denote the left-hand part of the KKT equation system, then we have exactly (14) with $K = N^2 + 2N + \#S$. From the implicit function theorem [35], we know that for each set of constraints S,

$$W_S = \{(P, \nu, \mu_S, \lambda) : T(P, \nu, \mu_S, \lambda) = 0 \text{ and}$$
$$T'(P, \nu, \mu_S, \lambda) \text{ has the maximal possible rank}\}$$

is a smooth one-dimensional curve or the empty set and can be parameterized by λ. In terms of the objective function $F_\lambda(P)$, the condition on $T'(P, \nu, \mu_S, \lambda)$ may be interpreted as a prohibition for the projection of $F_\lambda(P)$ on any feasible direction to be a constant. Therefore, the whole set of stationary points of $F_\lambda(P)$ when λ is varying from 0 to 1 may be represented as a union $W(\lambda) = \cup_S W_S(\lambda)$, where each $W_S(\lambda)$ is homotopic to a one-dimensional segment. The set $W(\lambda)$ may have quite complicated form. Some of $W_S(\lambda)$ may intersect each other, in this case, we observe a bifurcation point, some of $W_S(\lambda)$ may connect each other, in this case, we have a transformation point of one path into another, some of $W_S(\lambda)$ may appear only for $\lambda > 0$ and/or disappear before λ reaches 1. At the beginning, the PATH algorithm starts from $W_\emptyset(0)$, then it follows $W_\emptyset(\lambda)$ until the border of $\mathcal{D}$ (or a bifurcation point). If such an event occurs before $\lambda = 1$, then PATH moves to another segment of solutions corresponding to different constraints S and keeps moving along segments and sometimes jumping between segments until $\lambda = 1$. As stated in the previous section, one of the interesting properties of PATH algorithm is the fact that if $W_S*(\lambda)$ appears only when $\lambda = \lambda_1$ and $W_S*(\lambda_1)$ is a local minimum, then the value of the objective function F_{λ_1} in $W_S*(\lambda_1)$ is greater than at the point traced by the PATH algorithm.

3.5 Some Implementation Details

In this section, we provide a few details relevant for the efficient implementation of the PATH algorithms.

3.5.1 Frank-Wolfe

Among the different optimization techniques for the optimization of $F_\lambda(P)$ starting from the current local

minimum tracked by the PATH algorithm, we use in our experiments the Frank-Wolfe algorithm, which is particularly suited to optimization over doubly stochastic matrices [36]. The idea of the this algorithm is to sequentially minimize linear approximations of $F_0(P)$. Each step includes three operations as follows:

1. estimation of the gradient $\nabla F_\lambda(P_n)$,
2. resolution of the linear program

$$P_{n*} = \arg\min_{P \in D} \langle \nabla F_\lambda(P_n), P \rangle,$$

and

3. line search: finding the minimum of $F_\lambda(P)$ on the segment $[P_n \ P_n^*]$.

An important property of this method is that the second operation can be done efficiently by the Hungarian algorithm, in $O(N^3)$.

3.5.2 Efficient Gradient Computations

Another essential point is that we do not need to store matrices of size $N^2 \times N^2$ for the computation of $\nabla F_1(P)$ because the tensor product in $\nabla F_1(P) = -\text{vec}(\Delta^T) - 2(L_H^T \otimes L_G^T)\text{vec}(P)$ can be expressed in terms of $N \times N$ matrix multiplication:

$$\nabla F_1(P) = -\text{vec}(\Delta^T) - 2(L_H^T \otimes L_G^T)\text{vec}(P)$$
$$= -\text{vec}(\Delta^T) - 2\text{vec}(L_G^T P L_H).$$

The same thing may be done for the gradient of the convex component

$$\nabla F_0(P) = \nabla[\text{vec}(P)^T Q \text{vec}(P)],$$
$$\text{where } Q = (I \otimes A_G - A_H^T \otimes I)^T (I \otimes A_G - A_H^T \otimes I),$$
$$\nabla F_0(P) = 2Q\text{vec}(P)$$
$$= 2\text{vec}(A_G^2 P - A_G^T P A_H^T - A_G P A_H + P A_H^2).$$

3.5.3 Initialization

The proposed algorithm can be accelerated by the application of Newton algorithm as the first step of **QCV** (minimization of $F_0(P)$). First, let us rewrite the **QCV** problem as follows:

$$\min_{P \in \mathcal{D}} \|A_G P - P A_H\|_F^2 \Leftrightarrow$$

$$\min_{P \in \mathcal{D}} \text{vec}(P)^T Q \text{vec}(P) \Leftrightarrow \begin{cases} \min_P \text{vec}(P)^T Q \text{vec}(P) \\ \quad \text{such that} \\ B\text{vec}(P) = 1_{2N}, \\ \text{vec}(P) \geq 0_{N^2}, \end{cases} \quad (15)$$

where B is the matrix which codes the conditions $\sum_j P_{i,j} = 1$ and $\sum_i P_{i,j} = 1$. The Lagrangian has the following form:

$$\mathcal{L}(P, \nu, \lambda) = \text{vec}(P)^T Q \text{vec}(P) + \nu^T (B\text{vec}(P) - 1_{2N})$$
$$+ \mu^T \text{vec}(P),$$

where ν and μ are Lagrange multipliers. Now we would like to use Newton method for constrained optimization [36] to solve (15). Let μ_a denote the set of variables associated to the set of active constraints $\text{vec}(P) = 0$ at the

current points, then the Newton step consists of solving the following system of equations:

$$\begin{bmatrix} 2Q & B^T & I_a \\ B & 0 & 0 \\ I_a & 0 & 0 \end{bmatrix} \begin{bmatrix} \text{vec}(P) \\ \nu \\ \mu_a \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{array}{l} N^2 \text{ elements,} \\ 2N \text{ elements,} \\ \text{\# of act. ineq. cons.} \end{array}$$
$$(16)$$

More precisely, we have to solve (16) for P. The problem is that, in general situations, this problem is computationally demanding because it involves the inversion of matrices of size $O(N^2) \times O(N^2)$. In some particular cases, however, the Newton step becomes feasible. Typically, if none of the constraints $\text{vec}(P) \geq 0$ are active, then (16) takes the following form:[2]

$$\begin{bmatrix} 2Q & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \text{vec}(P) \\ \nu \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \begin{array}{l} N^2 \text{ elements,} \\ 2N \text{ elements.} \end{array} \quad (17)$$

The solution is then obtained as follows:

$$\text{vec}(P)_{KKT} = \frac{1}{2} Q^{-1} B^T (BQ^{-1}B^T)^{-1} 1_{2N}. \quad (18)$$

Because of the particular form of matrices Q and B, (18) may be computed very simply with the help of Kronecker product properties in $O(N^3)$ instead of $O(N^6)$. More precisely, the first step is the calculation of $M = BQ^{-1}B^T$, where $Q = (I \otimes A_G - A_H^T \otimes I)^2$. Matrix Q^{-1} may be represented as follows:

$$Q^{-1} = (U_H \otimes U_G)(I \otimes \Lambda_G - \Lambda_H \otimes I)^{-2}(U_H \otimes U_G)^T. \quad (19)$$

Therefore, the (i,j)th element of M is the following product:

$$B_i Q^{-1} B_j^T = \text{vec}(U_H^T \widetilde{B_i}^T U_G)^T)(\Lambda_G - \Lambda_H)^{-2}$$
$$\times \text{vec}(U_G^T \widetilde{B_j}^T U_H), \quad (20)$$

where B_i is the ith row of B and $\widetilde{B_i}$ is B_i reshaped into an $N \times N$ matrix. The second step is an inversion of the $2N \times 2N$ matrix M and a sum over columns $M^s = M^{-1}1_{2N}$. The last step is a multiplication of Q^{-1} by $B^T M^s$, which can be done with the same tricks as the first step. The result is the value of matrix P_{KKT}. We then have two possible scenarios:

1. If $P_{KKT} \in \mathcal{D}$, then we have found the solution of (15).
2. Otherwise, we take the point of intersection of the line (P_0, P_{KKT}) and the border $\partial\mathcal{D}$ as the next point and continue with Frank-Wolfe algorithm. Unfortunately, we can do the Newton step only once, then some of $P \geq 0$ constraints become active and efficient calculations are not feasible anymore. But even in this case, the Newton step is generally very useful because it decreases a lot the value of the objective function.

3.5.4 $d\lambda$-Adaptation Strategy

In practice, we found it useful to have the parameter $d\lambda$ in the algorithm given in Fig. 2 vary between iterations. Intuitively, $d\lambda$ should depend on the form of the objective function as follows: If $F_\alpha^\lambda(P)$ is smooth and if increasing the

2. It is true if we start our algorithm, for example, from the constant matrix $P_0 = \frac{1}{N} 1_N 1_N^T$.

parameter λ does not change a lot the form of the function, then we can afford large steps, in contrast, we should do a lot of small steps in the situation, where the objective function is very sensitive to changes in the parameter λ. The adaptive scheme we propose is the following. First, we fix a constant $d\lambda_{min} = 10^{-5}$, which represents the lower limit for $d\lambda$. When the PATH algorithm starts, $d\lambda$ is set to $d\lambda_{min}$. If we see after an update $\lambda_{new} = \lambda + d\lambda$ that $|F_{\lambda_{new}}(P^*(\lambda)) - F_\lambda(P^*(\lambda))| \leq \epsilon_\lambda$, then we double $d\lambda$ and keep multiplying $d\lambda$ by 2 as long as $|F_{\lambda_{new}}(P^*(\lambda)) - F_\lambda(P^*(\lambda))| \leq \epsilon_\lambda$. On the contrary, if $d\lambda$ is too large in the sense that $|F_{\lambda_{new}}(P^*(\lambda)) - F_\lambda(P^*(\lambda))| > \epsilon_\lambda$, then we divide $d\lambda$ by 2 until the criterion $|F_{\lambda_{new}}(P^*(\lambda)) - F_\lambda(P^*(\lambda))| \leq \epsilon_\lambda$ is met or $d\lambda = d\lambda_{min}$. Once the update on $d\lambda$ is done, we run the optimization (Frank-Wolfe) for the new value $\lambda + d\lambda$. The idea behind this simple adaptation schema is to choose $d\lambda$ which keeps $|F_{\lambda_{new}}(P^*(\lambda)) - F_\lambda(P^*(\lambda))|$ just below ϵ_λ.

3.5.5 Stopping Criterion

The choice of the update criterion $|F_{\lambda_{new}}(P^*(\lambda)) - F_\lambda(P^*(\lambda))|$ is not unique. Here, we check whether the function value has been changed a lot at the given point. But in fact, it may be more interesting to trace the minimum of the objective function. To compare the new minimum with the current one, we need to check the distance between these minima and the difference between function values. It means that we use the following condition as the stopping criterion:

$$|F_{\lambda_{new}}(P^*(\lambda_{new})) - F_\lambda(P^*(\lambda))| < \epsilon_\lambda^F \text{ and }$$
$$||P^*(\lambda_{new}) - P^*(\lambda)|| < \epsilon_\lambda^P.$$

Although this approach takes a little bit more computations (we need to run Frank-Wolfe on each update of $d\lambda$), it is quite efficient if we use the adaptation schema for $d\lambda$.

To fix the values ϵ_λ^F and ϵ_λ^P, we use a parameter M which defines a ratio between these parameters and the parameters of the stopping criterion used in the Frank-Wolfe algorithm, ϵ_{FW}^F (limit value of function decrement) and ϵ_{FW}^P (limit value of argument changing): $\epsilon_\lambda^F = M\epsilon_{FW}^F$ and $\epsilon_\lambda^P = M\epsilon_{FW}^P$. The parameter M represents an authorized level of stopping criterion relaxation when we increment λ. In practice, it means that when we start to increment λ, we may move away from the local minima and the extent of this move is defined by the parameter M. The larger the value of M, the further we move away and the larger $d\lambda$ may be used. In other words, the parameter M controls the width of the tube around the path of optimal solutions.

3.6 Algorithm Complexity

Here, we present the complexity of the algorithms discussed in the paper as follows.

- Umeyama's algorithm has three components: matrix multiplication, calculation of eigenvectors, and application of the Hungarian algorithm for (4). Complexity of each component is equal to $O(N^3)$. Thus, Umeyama's algorithm has complexity $O(N^3)$.
- LP approach (5) has complexity $O(N^7)$ (worst case) because it may be rewritten as an linear optimization problem with $3N^2$ variables [23].

In the PATH algorithm, there are three principal parameters that have a big impact on the algorithm

complexity. These parameters are ϵ_{FW}^F, ϵ_{FW}^P, M, and N. The first parameter ϵ_{FW} defines the precision of the Frank-Wolfe algorithm, in some cases, its speed may be sublinear [36]; however it should work much better when the optimization polytope has a "smooth" border [36].

The influence of the ratio parameter M is more complicated. In practice, in order to ensure that the objective function takes values between 0 and 1, we usually use the normalized version of the objective function:

$$F_{norm}(P) = \frac{||A_G P - P A_H||_F^2}{||A_G||_F^2 + ||A_H||_F^2}.$$

In this case, if we use the simple stopping criterion based on the value of the objective function, then the number of iteration over λ (number of Frank-Wolfe algorithm runs) is at least equal to $\frac{C}{M\epsilon_{FW}^F}$, where $C = \min_\mathcal{P} F_{norm} - \min_\mathcal{D} F_{norm}$.

The most important thing is how the algorithm complexity depends on the graph size N. In general, the number of iterations of the Frank-Wolfe algorithm scales as $O(\frac{\kappa}{\epsilon_{FW}^F})$, where κ is the conditional number of the Hessian matrix describing the objective function near a local minima [36]. It means that in terms of numbers of iterations, the parameter N is not crucial. N defines the dimensionality of the minimization problem, while κ may be close to zero or one depending on the graph structures, not explicitly on their size. On the other hand, N has a big influence on the cost of one iteration. Indeed, in each iteration step, we need to calculate the gradient and minimize a linear function over the polytope of doubly stochastic matrices. The gradient estimation and the minimization may be done in $O(N^3)$. In Section 4.2, we present the empirical results on how algorithm complexity and optimization precision depend on M (Fig. 7b) and N (Fig. 8).

3.7 Vertex Pairwise Similarities

If we match two labeled graphs, then we may increase the performance of our method by using information on pairwise similarities between their nodes. In fact, one method of image matching uses only this type of information, namely shape context matching [19]. To integrate the information on vertex similarities, we use the approach proposed in (3), but in our case, we use $F_\lambda(P)$ instead of $F_0(P)$

$$\min_P F_\lambda^\alpha(P) = \min_P (1-\alpha)F_\lambda(P) + \alpha\text{tr}(C^T P). \quad (21)$$

The advantage of the last formulation is that $F_\lambda^\alpha(P)$ is just $F_\lambda(P)$ with an additional linear term. Therefore, we can use the same algorithm for the minimization of $F_\lambda^\alpha(P)$ as the one we presented for the minimization of $F_\lambda(P)$.

3.8 Matching Graphs of Different Sizes

Often, in practice, we have to match graphs of different sizes N_G and N_H (suppose, for example, that $N_G > N_H$). In this case, we have to match all vertices of graph H to a subset of vertices of graph G. In the usual case when $N_G = N_H$, the error (1) corresponds to the number of mismatched edges

(edges which exist in one graph and do not exist in the other one). When we match graphs of different sizes, the situation is a bit more complicated. Let $V_G^+ \subset V_G$ denote the set of vertices of graph G that are selected for matching to vertices of graph H, let $V_G^- = V_G \setminus V_G^+$ denote all the rest. Therefore, all edges of the graph G are divided into four parts $E_G = E_G^{++} \cup E_G^{+-} \cup E_G^{-+} \cup E_G^{--}$, where E_G^{++} are edges between vertices from V_G^+, E_G^{--} are edges between vertices from V_G^-, and E_G^{+-} and E_G^{-+} are edges from V_G^+ to V_G^- and from V_G^- to V_G^+, respectively. For undirected graphs, the sets E_G^{+-} and E_G^{-+} are the same (but, for directed graphs, we do not consider, they would be different). The edges from E_G^{--}, E_G^{+-}, and E_G^{-+} are always mismatched and a question is whether we have to take them into account in the objective function or not. According to the answer, we have three types of matching error (four for directed graphs) with interesting interpretations:

1. We count only the number of mismatched edges between H and the chosen subgraph $G^+ \subset G$. It corresponds to the case when the matrix P from (1) is a matrix of size $N_G \times N_H$ and $N_G - N_H$ rows of P contain only zeros.
2. We count the number of mismatched edges between H and the chosen subgraph $G^+ \subset G$. And we also count all edges from E_G^{--}, E_G^{+-}, and E_G^{-+}. In this case, P from (1) is a matrix of size $N_G \times N_G$. And we transform matrix A_H into a matrix of size $N_G \times N_G$ by adding $N_G - N_H$ zero rows and zero columns. It means that we add dummy isolated vertices to the smallest graph and then match graphs of the same size.
3. We count the number of mismatched edges between H and chosen subgraph $G^+ \subset G$. And we also count all edges from E_G^{+-} (or E_G^{-+}). It means that we count matching error between H and G^+ and we also count the number of edges, which connect G^+ and G^-. In other words, we are looking for subgraph G^+, which is similar to H and maximally isolated in the graph G.

Each type of error may be useful according to the context and interpretation, but a priori, it seems that the best choice is the second one where we add dummy nodes to the smallest graph. The main reason is the following. Suppose that graph H is quite sparse and graph G has two candidate subgraphs G_s^+ (also quite sparse) and G_d^+ (dense). The upper bound for the matching error between H and G_s^+ is $\#V_H + \#V_{G_s^+}$, the lower bound for the matching error between H and G_d^+ is $\#V_{G_d^+} - \#V_H$. So, if $\#V_H + \#V_{G_s^+} < \#V_{G_d^+} - \#V_H$, then we will always choose the graph G_s^+ with the first strategy, even if it is not similar at all to the graph H. The main explanation of this effect lies in the fact that the algorithm tries to minimize the number of mismatched edges, but not to maximize the number of well-matched edges. In contrast, when we use dummy nodes, we do not have this problem because if we take a very sparse subgraph G^+, it increases the number of edges in G^- (the common number of edges in G^+ and G^- is constant and equal to the number of edges in G), and finally, it decreases the quality of matching.

4 SIMULATIONS

4.1 Synthetic Examples

In this section, we compare the proposed algorithm with some classical methods on artificially generated graphs. Our choice of random graph types is based on [37], where the authors discuss different types of random graphs which are the most frequently observed in various real-world applications (World Wide Web, collaborations networks, social networks, etc.). Each type of random graphs is defined by the distribution function of node degree $\text{Prob}(\text{node degree} = k) = VD(k)$. The vector of node degrees of each graph is supposed to be an i.i.d sample from $VD(k)$. In our experiments, we have used the following types of random graphs.

Binomial law	$VD(k) = C_N^k p^k (1-p)^{1-k}$
Geometric law	$VD(k) = (1 - e^{-\mu}) e^{\mu k}$
Power law	$VD(k) = C_\tau k^{-\tau}$

The schema of graph generation is as follows:

1. generate a sample $d = (d_1, \ldots, d_N)$ from $VD(k)$;
2. if $\sum_i d_i$ is odd, then go to step 1;
3. while $\sum_i d_i > 0$

   a. choose randomly two nonzero elements from d: d_{n1} and d_{n2},
   b. add edge $(n1, n2)$ to the graph,
   c. $d_{n1} \leftarrow d_{n1} - 1$ $d_{n2} \leftarrow d_{n2} - 1$.

If we are interested in isomorphic graph matching, then we compare just the initial graph and its randomly permuted copy. To test the matching of nonisomorphic graphs, we add randomly σN_E edges to the initial graph and its permitted copy, where N_E is the number of edges in the original graph and σ is the noise level.

4.2 Results

The first series of experiments are experiments on small size graphs (N = 8), here, we are interested in comparison of the PATH algorithm (see Fig. 2), the **QCV** approach (8), Umeyama spectral algorithm (4), the linear programming approach (5), and exhaustive search which is feasible for the small size graphs. The algorithms were tested on the three types of random graphs (binomial, exponential, and power). The results are presented in Fig. 4. The same experiment was repeated for middle-sized graphs (N = 20, Fig. 5) and large graphs (N = 100, Fig. 6).

In all cases, the PATH algorithm works much better than all other approximate algorithms. There are some important things to note here. First, the choice of norm in (1) is not very important—results of QCV and LP are about the same. Second, following the solution paths is very useful compared to just minimizing the convex relaxation and projecting the solution on the set of permutation matrices (PATH algorithms works much better than QCV). Another noteworthy observation is that the performance of PATH is very close to the optimal solution when the latter can be evaluated.

We note that sometimes the matching error decreases as the noise level increases (e.g., in Figs. 6c and 5c), which can be explained as follows. The matching error is upper
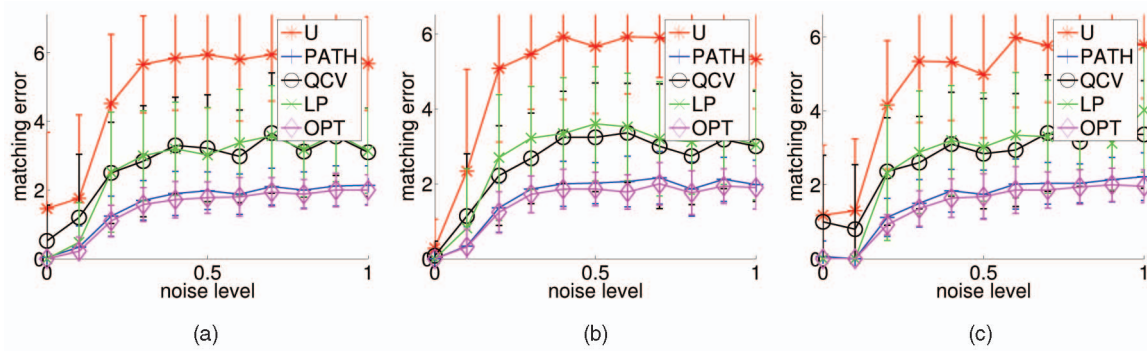
Fig. 4. Matching error (mean value over sample of size 100) as a function of noise. Graph size $N = 8$. U—Umeyama's algorithm, LP—linear programming algorithm, QCV—convex optimization, PATH—path minimization algorithm, OPT—an exhaustive search (the global minimum). The range of error bars is the standard deviation of matching errors. (a) Bin, (b) exp, and (c) pow.
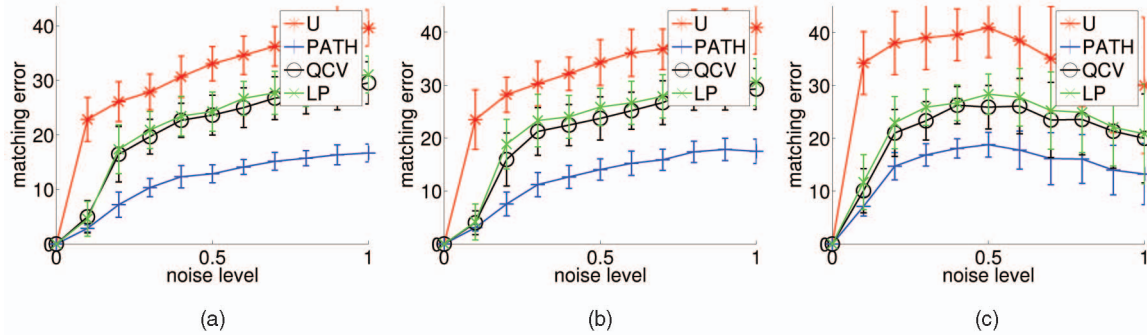

Fig. 5. Matching error (mean value over sample of size 100) as a function of noise. Graph size $N = 20$. U—Umeyama's algorithm, LP—linear programming algorithm, QCV—convex optimization, PATH—path minimization algorithm. (a) Bin, (b) exp, and (c) pow.
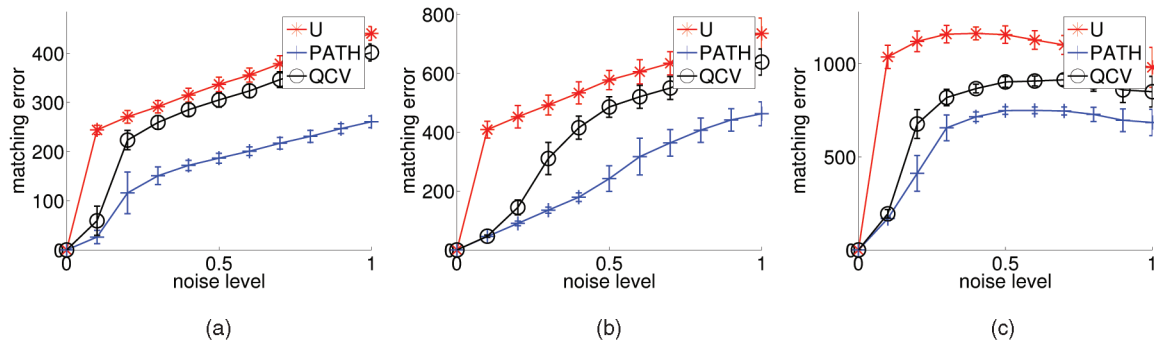

Fig. 6. Matching error (mean value over sample of size 100) as a function of noise. Graph size $N = 100$. U—Umeyama's algorithm, QCV—convex optimization, PATH—path minimization algorithm. (a) Bin, (b) exp, and (c) pow.

bounded by the minimum of the total number of zeros in the adjacency matrices A_G and A_H, so, in general, this upper bound decreases when the edge density increases. When the noise level increases, it makes graphs denser, and consequently, the upper bound of matching error decreases. The general behavior of graph matching algorithms as functions of the graph density is presented in Fig. 7a. Here again the matching error decreases when the graph density becomes very large.

The parameter M (see Section 3.5.5) defines how precisely the PATH algorithm tries to follow the path of local minima. The larger M, the faster the PATH algorithm. At the extreme, when M is close to $1/\epsilon_{FW}$, we jump directly from the convex function ($\lambda = 0$) to the concave one ($\lambda = 1$). Fig. 7b shows in more detail how algorithm speed and precision depend on M.

Another important aspect to compare the different algorithms is their runtime complexity as a function of N. Fig. 8 shows the time needed to obtain the matching between two graphs as a function of the number of vertices N (for N between 10 and 100) for the different methods. These curves are coherent with theoretical values of algorithm complexities summarized in Section 3.6. In particular, we observe that Umeyama's algorithm is the fastest method, but that QCV and PATH have the same complexity in N. The LP method is competitive with QCV and PATH for small graphs, but has a worse complexity in N.

5   QAP BENCHMARK LIBRARY

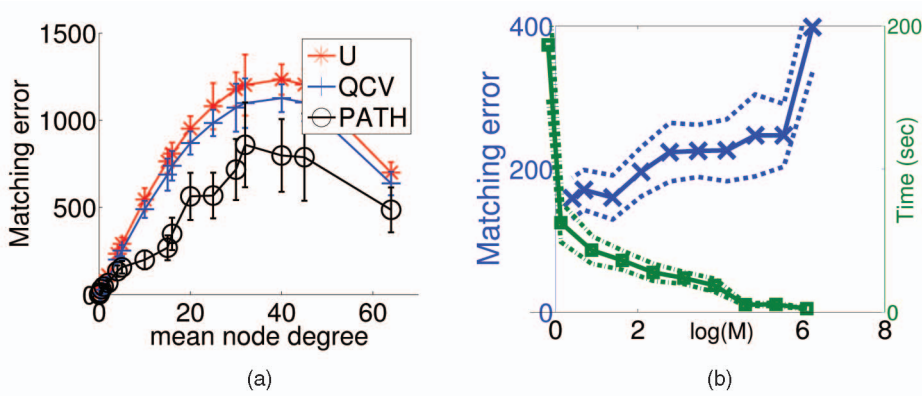The problem of graph matching may be considered as a particular case of the QAP. The minimization of the loss

Fig. 7. (a) Algorithm performance as a function of graph density. (b) Precision and speed of the PATH algorithm as a function of M, the relaxation constant used in the PATH algorithm (see Section 3.5.5). In both cases, graph size $N = 100$, noise level $\sigma = 0.3$, and sample size is equal to 30. Error bars represent standard deviation of the matching error (not averaged).
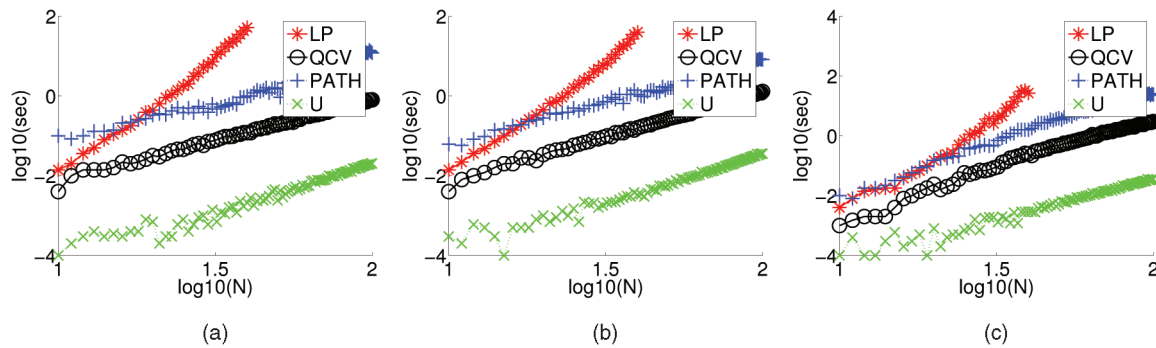

Fig. 8. Timing of U, LP, QCV, and PATH algorithms as a function of graph size, for the different random graph models. LP slope ≈ 6.7, U, QCV, and PATH slope ≈ 3.4. (a) Bin, (b) exp, and (c) pow.

function (1) is equivalent to the maximization of the following function:

$$\max_{P} \ \text{tr}\left(P^T A_G^T P A_H\right).$$

Therefore, it is interesting to compare our method with other approximate methods proposed for QAP. Cremers et al. [18] proposed the QPB algorithm for that purpose and tested it on matrices from the QAP benchmark library [38], QPB results were compared to the results of graduated assignment algorithm GRAD [17] and Umeyama's algorithm. Results of PATH application to the same matrices are presented in Table 1, scores for QPB and graduated assignment algorithm are taken directly from the publication [18]. We observe that on 14 out of 16 benchmarks, PATH is the best optimization method among the methods tested.

6 IMAGE PROCESSING

In this section, we present two applications in image processing. The first one (Section 6.1) illustrates how taking into account information on graph structure may increase image alignment quality. The second one (Section 6.2) shows that the structure of contour graphs may be very important in classification tasks. In both examples, we compare the performance of our method with the shape context approach [19], a state-of-the-art method for image matching.

6.1 Alignment of Vessel Images

The first example is dedicated to the problem of image alignment. We consider two photos of vessels in human eyes. The original photos and images of extracted vessel contours (obtained from the method of [39]) are presented in Fig. 9. To align the vessel images, the shape context algorithm uses the context radial histograms of contour points (see [19]). In other words, according to the shape context algorithm, one aligns points that have similar

TABLE 1
Experiment Results for QAPLIB Benchmark Data Sets

| QAP	MIN	PATH	QPB	GRAD	U
chr12c	11156	**18048**	20306	19014	40370
chr15a	9896	**19086**	26132	30370	60986
chr15c	9504	**16206**	29862	23686	76318
chr20b	2298	**5560**	6674	6290	10022
chr22b	6194	**8500**	9942	9658	13118
esc16b	292	300	**296**	298	306
rou12	235528	**256320**	278834	273438	295752
rou15	354210	391270	**381016**	457908	480352
rou20	725522	**778284**	804676	840120	905246
tai10a	135028	**152534**	165364	168096	189852
tai15a	388214	**419224**	455778	451164	483596
tai17a	491812	**530978**	550852	589814	620964
tai20a	703482	**753712**	799790	871480	915144
tai30a	1818146	**1903872**	1996442	2077958	2213846
tai35a	2422002	**2555110**	2720986	2803456	2925390
tai40a	3139370	**3281830**	3529402	3668044	3727478
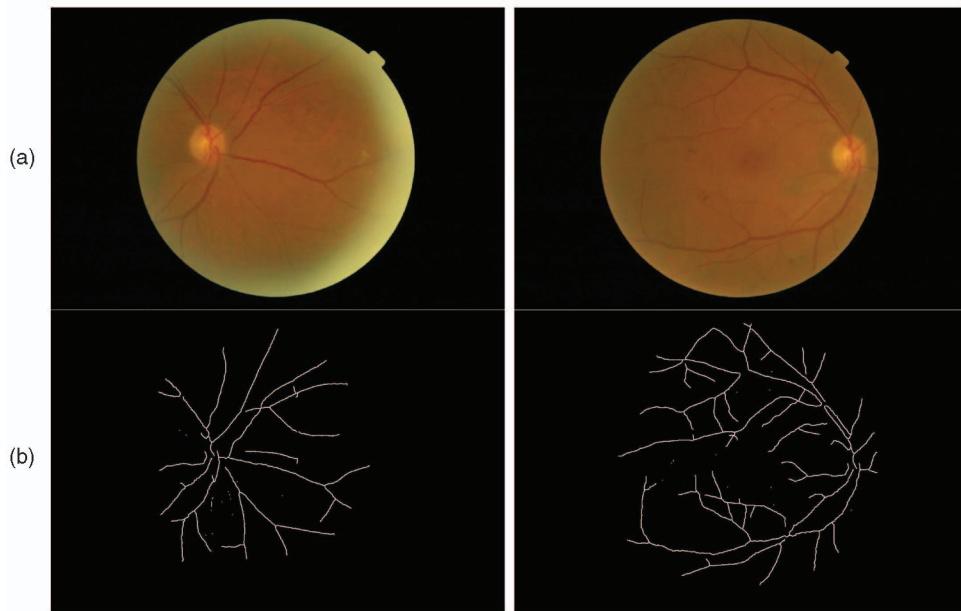
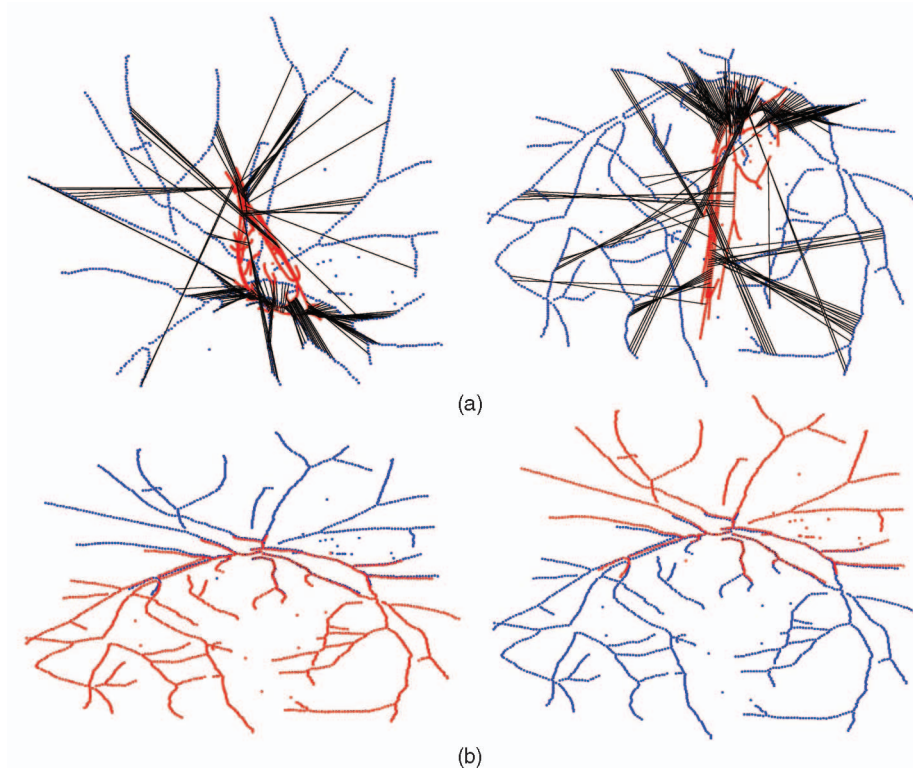Fig. 9. (a) Eye photos and (b) vessel contour extraction.

Fig. 10. Comparison of (a) alignment based on the shape context and (b) PATH optimization algorithm. For each algorithm, we present two alignments: image "1" on image "2" and the inverse. Each alignment is a spline-based transformation (see text).

context histograms. The PATH algorithm uses also information about the graph structure. When we use the PATH algorithm, we have to tune the parameter α (21), we tested several possible values and took the one which produced the best result. To construct graph, we use all points of vessel contours as graph nodes and connect all nodes within a circle of radius r (in our case, we use $r = 50$). Finally, to each edge (i, j), we associate the weight $w_{i,j} = \exp(-|x_i - y_j|)$.

A graph matching algorithm produces an alignment of image contours, then to align two images, we have to expand this alignment to the rest of the image. For this purpose, we use a smooth spline-based transformation [40]. In other words, we estimate parameters of the spline transformation from the known alignment of contour points and then apply this transformation to the whole image. Results of image matching based on shape context algorithm and PATH algorithm are presented in Fig. 10, where black lines

TABLE 2
Alignment of Vessel Images, Algorithm Performance

Method	Shape context	Umeyama	QCV	PATH
matching error	870.61	764.83	654.42	625.75

designate connections between associated points. We observe that the context shape method creates many unwanted matching, while PATH produces a matching that visually corresponds to a correct alignment of the structure of vessels. The main reason why graph matching works better than shape context matching is the fact that shape context does not take into account the relational positions of matched points and may lead to totally incoherent graph structures. In contrast, graph matching tries to match pairs of the nearest points in one image with pairs of the nearest points in another one.

Among graph matching methods, different results are obtained with different optimization algorithms. Table 2 shows the matching errors produced by different algorithms on this vessel alignment problem. The PATH algorithm has the smallest matching error, with the alignment shown in Fig. 10. QCV comes next, with an alignment that is also visually correct. On the other hand, the Umeyama algorithm has a much larger matching error and visually fails to find a correct alignment, similar to the shape context method.

6.2 Recognition of Handwritten Chinese Characters

Another example that we consider in this paper is the problem of Chinese character recognition from the ETL9B data set [41]. The main idea is to use a score of optimal matching as a similarity measure between two images of characters. This similarity measure can be used then in machine learning algorithms, K-nearest neighbors (KNNs), for instance, for character classification. Here, we compare the performance of four methods: linear support vector machine (SVM), SVM with gaussian kernel, KNN based on score of shape context matching, and KNN based on scores from graph matching which combines structural and shape context information. As a score, we use just the value of the objective function (21) at the (locally) optimal point. We

have selected three Chinese characters known to be difficult to distinguish by automatic methods. Examples of these characters as well as extracted graphs (obtained by thinning and uniformly subsampling the images) are presented in Fig. 11. For SVM-based algorithms, we use directly the values of image pixels (so each image is represented by a binary vector), in graph matching algorithm, we use binary adjacency matrices of extracted graphs and shape context matrices (see [19]).

Our data set consists of 50 examples (images) of each class. Each image is represented by 63×64 binary matrix. To compare different methods, we use the cross validation error (fivefold). The dependency of classification error from two algorithm parameters (α—coefficient of linear combination (21) and k—number of the nearest neighbors used in KNN) is shown in Fig. 12.

Two extreme choices $\alpha = 1$ and $\alpha = 0$ correspond, respectively, to pure shape context matching, i.e., when only node labels information is used, and pure unlabeled graph matching. It is worth observing here that KNN based just on the score of unlabeled graph matching does not work very well, the classification error being about 60 percent. An explanation of this phenomenon is the fact that learning patterns have very unstable graph structure within one class. The pure shape context method has a classification error of about 39 percent. The combination of shape context and graph structure information allows to decrease the classification error down to 25 percent. Beside the PATH algorithm, we tested also the QCV and Umeyama algorithms, the Umeyama algorithm almost does not decrease the classification error. The QCV algorithm works better than the Umeyama algorithm, but still worse than the PATH algorithm. Complete results can be found in Table 3.

7 CONCLUSION

We have presented the PATH algorithm, a new technique for graph matching based on convex-concave relaxations of the initial integer programming problem. PATH allows to integrate the alignment of graph structural elements with the matching of vertices with similar labels. Its results are
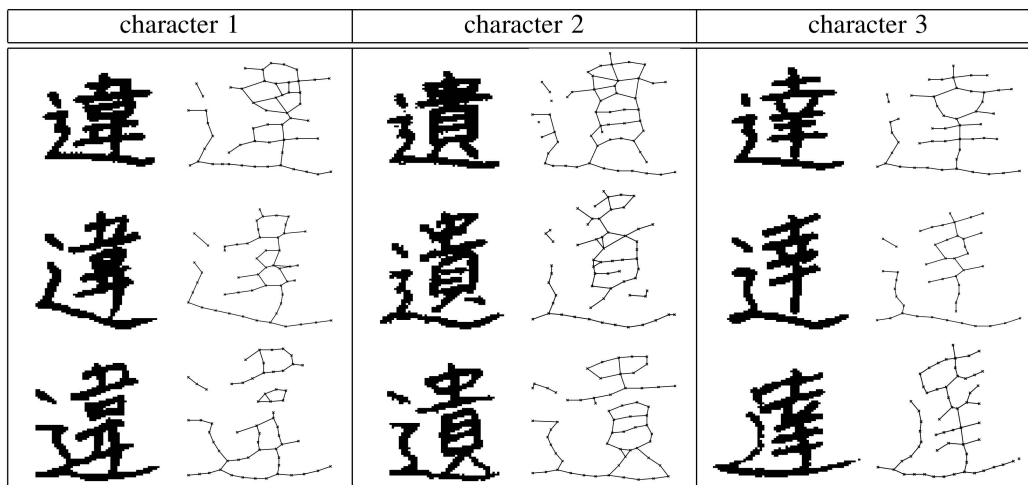

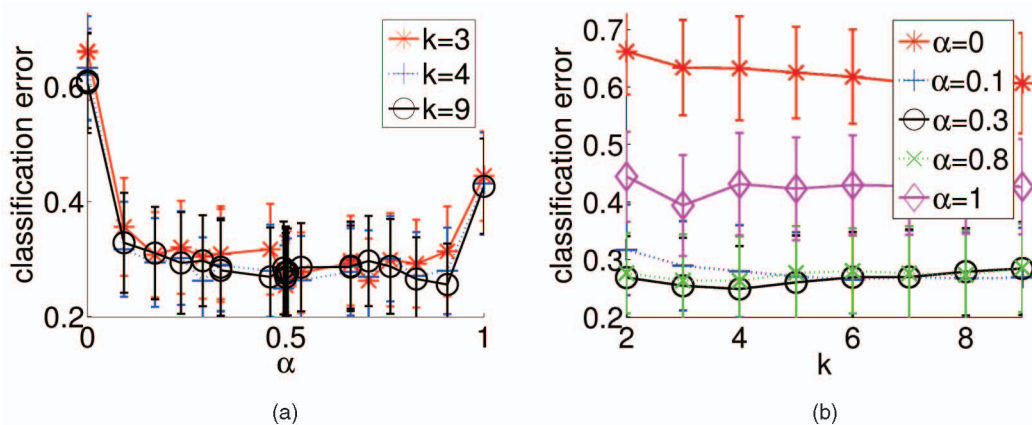Fig. 11. Chinese characters from the ETL9B data set.

Fig. 12. (a) Classification error as a function of α. (b) Classification error as a function of k. Classification error is estimated as cross-validation error (fivefold, 50 repetitions), the range of the error bars is the standard deviation of test error over onefold (not averaged over folds and repetition).

competitive with state-of-the-art methods in several graph matching and QAP benchmark experiments. Moreover, PATH has a theoretical and empirical complexity competitive with the fastest available graph matching algorithms.

Two points can be mentioned as interesting directions for further research. First, the quality of the convex-concave approximation is defined by the choice of convex and concave relaxation functions. Better performances may be achieved by more appropriate choices of these functions. Second, another interesting point concerns the construction of a good concave relaxation for the problem of directed graph matching, i.e., for asymmetric adjacency matrix. Such generalizations would be interesting also as possible polynomial-time approximate solutions for the general QAP problem.

APPENDIX A

A TOY EXAMPLE

The PATH algorithm does not generally find the global optimum of the NP-complete optimization problem. In this section, we illustrate with two examples how the set of local optima tracked by PATH may or may not lead to the global optimum.

More precisely, we consider two simple graphs with the following adjacency matrices:

$$\mathbf{G} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \text{ and } \mathbf{H} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

TABLE 3
Classification of Chinese Characters

Method	CV	STD
Linear SVM	0.377	$\pm\ 0.090$
SVM with gaussian kernel	0.359	$\pm\ 0.076$
KNN (PATH) (α=1): shape context	0.399	$\pm\ 0.081$
KNN (PATH) (α=0.4)	0.248	$\pm\ 0.075$
KNN (PATH) (α=0): pure graph matching	0.607	$\pm\ 0.072$
KNN (U) (α=0.9): α best choice	0.382	$\pm\ 0.077$
KNN (QCV) (α=0.3): α best choice	0.295	$\pm\ 0.061$

(CV, STD)—Mean and standard deviation of test error over cross-validation runs (fivefold, 50 repetitions).

Let C denote the cost matrix of vertex association

$$\mathbf{C} = \begin{bmatrix} 0.1691 & 0.0364 & 1.0509 \\ 0.6288 & 0.5879 & 0.8231 \\ 0.8826 & 0.5483 & 0.6100 \end{bmatrix}.$$

Let us assume that we have fixed the trade-off $\alpha = 0.5$ and our objective is then to find the global minimum of the following function:

$$F_0(P) = 0.5\|GP - PH\|_F^2 + 0.5\text{tr}(C'P), \quad P \in \mathcal{P}. \quad (22)$$

As explained before, the main idea underlying the PATH algorithm is to try to follow the path of global minima of $F_\lambda^\alpha(P)$ (21). This may be possible if all global minima P_λ^* form a continuous path, which is not true in general. In the case of small graphs, we can find the exact global minimum of $F_\lambda^\alpha(P)$ for all λ. The trace of global minima as functions of λ is presented in Fig. 13a (i.e., we plot the values of the nine parameters of the doubly stochastic matrix, which are, as expected, all equal to zero or one when $\lambda = 1$). When λ is near 0.2, there is a jump of global minimum from one face to another. However, if we change the linear term C to

$$\mathbf{C}' = \begin{bmatrix} 0.4376 & 0.3827 & 0.1798 \\ 0.3979 & 0.3520 & 0.2500 \\ 0.1645 & 0.2653 & 0.5702 \end{bmatrix},$$

then the trace becomes smooth (see Fig. 13b) and the PATH algorithm then finds the globally optimum point. Characterizing cases where the path is indeed smooth is the subject of ongoing research.

APPENDIX B

KRONECKER PRODUCT

The Kronecker product of two matrices $A \otimes B$ is defined as follows:

$$A \otimes B = \begin{bmatrix} Ba_{11} & \cdots & Ba_{1n} \\ \vdots & \ddots & \vdots \\ Ba_{m1} & \cdots & Ba_{mn} \end{bmatrix}.$$

Two important properties of Kronecker product that we use in this paper are:
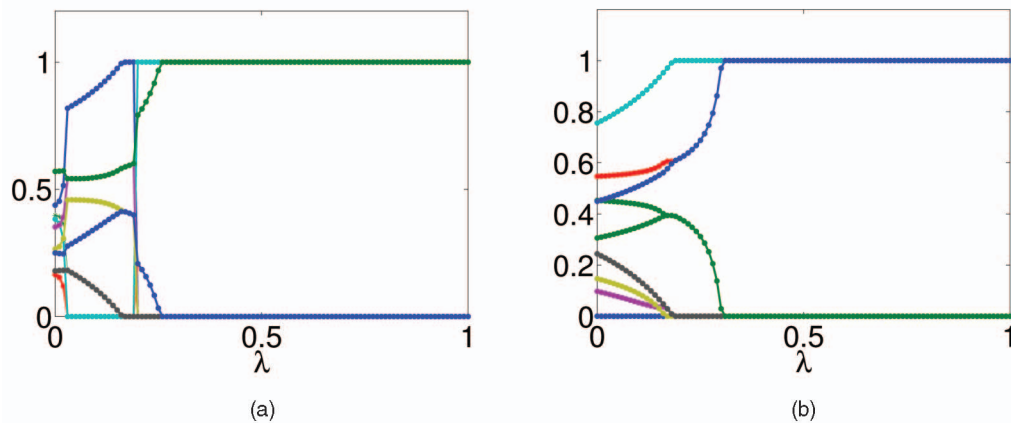
Fig. 13. Nine coordinates of global minimum of F_λ^α as a function of λ.

$$(A^T \otimes B)\mathrm{vec}(X) = \mathrm{vec}(BXA)$$
$$\text{and } \mathrm{tr}(X^T AXB^T) = \mathrm{vec}(X)^T(B \otimes A)\mathrm{vec}(X).$$

REFERENCES

[1] R.S.T. Lee and J.N.K. Liu, "An Oscillatory Elastic Graph Matching Model for Recognition of Offline Handwritten Chinese Characters," *Proc. Third Int'l Conf. Knowledge-Based Intelligent Information Eng. Systems,* pp. 284-287, 1999.

[2] A. Filatov, A. Gitis, and I. Kil, "Graph-Based Handwritten Digit String Recognition," *Proc. Third Int'l Conf. Document Analysis and Recognition,* pp. 845-848, 1995.

[3] H.F. Wang and E.R. Hancock, "Correspondence Matching Using Kernel Principal Components Analysis and Label Consistency Constraints," *Pattern Recognition,* vol. 39, no. 6, pp. 1012-1025, June 2006.

[4] B. Luo and E.R. Hancock, "Alignment and Correspondence Using Singular Value Decomposition," *Lecture Notes in Computer Science,* vol. 1876, pp. 226-235, Springer, 2000.

[5] M. Carcassoni and E.R. Hancock, "Spectral Correspondence for Point Pattern Matching," *Pattern Recognition,* vol. 36, pp. 193-204, 2002.

[6] C. Schellewald and C. Schnor, "Probabilistic Subgraph Matching Based on Convex Relaxation," *Lecture Notes in Computer Science,* vol. 3757, pp. 171-186, Springer, 2005.

[7] R. Singh, J. Xu, and B. Berger, "Pairwise Global Alignment of Protein Interaction Networks by Matching Neighborhood Topology," *Proc. 11th Int'l Conf. Research in Computational Molecular Biology,* 2007.

[8] Y. Wang, F. Makedon, J. Ford, and H. Huang, "A Bipartite Graph Matching Framework for Finding Correspondences between Structural Elements in Two Proteins," *Proc. IEEE 26th Ann. Int'l Conf. Eng. in Medicine and Biology Society,* pp. 2972-2975, 2004.

[9] W.R. Taylor, "Protein Structure Comparison Using Bipartite Graph Matching and Its Application to Protein Structure Classification," *Molecular and Cellular Proteomics,* vol. 1, no. 4, pp. 334-339, 2002.

[10] D.C. Schmidt and L.E. Druffel, "A Fast Backtracking Algorithm for Test Directed Graphs for Isomorphism," *J. ACM,* vol. 23, no. 3, pp. 433-445, 1976.

[11] J.R. Ullmann, "An Algorithm for Subgraph Isomorphism," *J. ACM,* vol. 23, no. 1, pp. 31-42, 1976.

[12] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento, "Performance Evaluation of the vf Graph Matching Algorithm," *Proc. 10th Int'l Conf. Image Analysis and Processing,* vol. 2, pp. 1038-1041, 1991.

[13] S. Umeyama, "An Eigendecomposition Approach to Weighted Graph Matching Problems," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 10, no. 5, pp. 695-703, Sept. 1988.

[14] L.S. Shapiro and J.M. Brady, "Feature-Based Correspondence," *Image and Vision Computing,* vol. 10, pp. 283-288, 1992.

[15] T. Caelli and S. Kosinov, "An Eigenspace Projection Clustering Method for Inexact Graph Matching," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 26, no. 4, pp. 515-519, Apr. 2004.

[16] H. Almohamad and S. Duffuaa, "A Linear Programming Approach for the Weighted Graph Matching Problem," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 15, no. 5, pp. 522-525, May 1993.

[17] S. Gold and A. Rangarajan, "A Graduated Assignment Algorithm for Graph Matching," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 18, no. 4, pp. 377-388, Apr. 1996.

[18] D. Cremers, T. Kohlberger, and C. Schnor, "Evaluation of Convex Optimization Techniques for the Weighted Graph-Matching Problem in Computer Vision," *Proc. 23rd DAGM-Symp. Pattern Recognition,* vol. 2191, 2001.

[19] S. Belongie, J. Malik, and J. Puzicha, "Shape Matching and Object Recognition Using Shape Contexts," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 24, no. 4, pp. 509-522, Apr. 2002.

[20] K. Brein, M. Remm, and E. Sonnhammer, "Inparanoid: A Comprehensive Database of Eukaryotic Orthologs," *Nucleic Acids Research,* vol. 33, 2005.

[21] M.R. Garey and D.S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

[22] J.M. Borwein and A.S. Lewis, *Convex Analysis and Nonlinear Optimization.* Springer-Verlag, 2000.

[23] S. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge Univ. Press, 2003.

[24] L.F. McGinnis, "Implementation and Testing of a Primal-Dual Algorithm for the Assignment Problem," *Operations Research,* vol. 31, no. 2, pp. 277-291, 1983.

[25] D. Conte, P. Foggia, C. Sansone, and M. Vento, "Thirty Years of Graph Matching in Pattern Recognition," *Int'l J. Pattern Recognition and Artificial Intelligence,* vol. 18, pp. 265-298, 2004.

[26] G.H. Golub and C.F.V. Loan, *Matrix Computations,* third ed. Johns Hopkins Univ. Press, 1996.

[27] L.F. McGinnis, "Implementation and Testing of a Primal-Dual Algorithm for the Assignment Problem," *Operations Research,* vol. 31, pp. 277-291, 1983.

[28] H. Kuhn, "The Hungarian Method for the Assignment Problem," *Naval Research,* vol. 2, pp. 83-97, 1955.

[29] M. Frank and P. Wolfe, "An Algorithm for Quadratic Programming," *Naval Research Logistics Quarterly,* vol. 3, pp. 95-110, 1956.

[30] F.R.K. Chung, *Spectral Graph Theory.* Am. Math. Soc., 1997.

[31] R. Rockafeller, *Convex Analysis.* Princeton Univ. Press, 1970.

[32] K.M. Anstreicher and N.W. Brixius, "A New Bound for the Quadratic Assignment Problem Based on Convex Quadratic Programming," *Math. Programming,* vol. 89, no. 3, pp. 341-357, 2001.

[33] A. Blake and A. Zisserman, *Visual Reconstruction.* MIT Press, 1987.

[34] E. Allgower and K. Georg, *Numerical Continuation Methods.* Springer, 1990.

[35] J. Milnor, *Topology from the Differentiable Viewpoint.* Univ. Press of Virginia, 1969.

[36] D. Bertsekas, *Nonlinear Programming.* Athena Scientific, 1999.

[37] M.E.J. Newman, S.H. Strogatz, and D.J. Watts, "Random Graphs with Arbitrary Degree Distributions and Their Applications," *Physical Rev.,* vol. 64, 2001.

[38] E. Cela, "Quadratic Assignment Problem Library," www.opt. math.tu-graz.ac.at/qaplib/, 2007.

[39] T. Walter, J.-C. Klein, P. Massin, and A. Erignay, "Detection of the Median Axis of Vessels in Retinal Images," *European J. Ophthalmology,* vol. 13, no. 2, 2003.

[40] F. Bookstein, "Principal Warps: Thin-Plate Splines and the Decomposition of Deformations," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 11, no. 6, pp. 567-585, June 1989.

[41] T. Saito, H. Yamada, and K. Yamamoto, "On the Data Base etl9b of Handprinted Characters in jis Chinese Characters and Its Analysis," *IEICE Trans.,* vol. 68, no. 4, pp. 757-764, 1985.

Mikhail Zaslavskiy received the graduate degree from the Saint-Petersburg State University, Russia, in 2003, and the Ecole Polytechnique, Palaiseau, France, in 2006. He is currently working toward the PhD degree at the Centre for Computational Biology and the Centre for Mathematical Morphology at Mines ParisTech, France, where he is working with Jean-Philippe Vert and Francis Bach. He is also affiliated with the Institut Curie and INSERM U900, Paris, France. His research interests include machine learning, graph-based methods in pattern recognition, optimization algorithms, and their applications to computational biology and computer vision.

Francis Bach received the graduate degree from the Ecole Polytechnique, Palaiseau, France, in 1997, and the PhD degree from the Computer Science Division at the University of California, Berkeley, in 2005. He is a researcher in the Willow INRIA Project Team at the Computer Science Department of the Ecole Normale Supérieure, Paris, France. His research interests include machine learning, optimization, graphical models, kernel methods, sparse methods, and statistical signal processing.

Jean-Philippe Vert received the graduate degrees from the Ecole Polytechnique, Palaiseau, France, in 1995, and the Ecole des Mines de Paris, France, in 1998, and the MSc and PhD degrees in mathematics from Paris 6 University, in 1997 and 2001, respectively. He is currently a researcher and the director of the Centre for Computational Biology at Mines ParisTech, France. He is also affiliated with the Institut Curie and INSERM U900, Paris, France. His research interests include statistics, machine learning, and their applications to computational biology.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.