Edit Distance From Graph Spectra

A. Robles-Kelly
Department of Computer Science
The University of York
Heslington, York YO105DD, UK
arobkell@cs.york.ac.uk

E. R. Hancock
Department of Computer Science
The University of York
Heslington, York YO105DD, UK
erh@cs.york.ac.uk

Abstract

This paper is concerned with computing graph edit distance. One of the criticisms that can be leveled at existing methods for computing graph edit distance is that it lacks the formality and rigour of the computation of string edit distance. Hence, our aim is to convert graphs to string sequences so that standard string edit distance techniques can be used. To do this we use graph spectral seriation method to convert the adjacency matrix into a string or sequence order. We pose the problem of graph-matching as maximum a posteriori probability alignment of the seriation sequences for pairs of graphs. This treatment leads to an expression for the edit costs. We compute the edit distance by finding the sequence of string edit operations which minimise the cost of the path traversing the edit lattice. The edit costs are defined in terms of the a posteriori probability of visiting a site on the lattice. We demonstrate the method with results on a data-set of Delaunay graphs.

1. Introduction

Graph-matching is a task of pivotal importance in high-level vision since it provides a means by which abstract pictorial descriptions can be matched to one-another. Unfortunately, since the process of eliciting graph structures from raw image data is a task of some fragility due to noise and the limited effectiveness of the available segmentation algorithms, graph-matching is invariably approached by inexact means [17, 14]. The search for a robust means of inexact graph-matching has been the focus of sustained activity over the last two decades. Early work drew heavily on ideas from structural pattern recognition and revolved around extending the concept of string edit distance to graphs [14, 5]. One of the criticisms that can be aimed at this early work is that it lacks the formal rigour of the corresponding work on string edit distance. However, recently, Bunke and his coworkers have returned to the problem and have shown the relationship between graph edit distance and the size of the maximum common subgraph [2].

An alternative approach to the problem is to convert graphs to string sequences and to use the existing theory of string edit distance. The problem of converting a graph to a sequence is known as seriation. Stated succinctly, it is the task of ordering the set of nodes in a graph in a sequence such that strongly correlated nodes are placed next to one another. The problem is important in a number of areas including data visualisation and bioinformatics, where it is used for DNA sequencing. The seriation problem can be approached in a number of ways. Clearly the problem of searching for a serial ordering of the nodes, which maximally preserves the edge ordering is one of exponential complexity. As a result, approximate solution methods have been employed. These involve casting the problem in an optimisation setting. Hence techniques such as simulated annealing and mean field annealing have been applied to the problem. It may also be formulated using semidefinite programming, which is a technique closely akin to spectral graph theory since it relies on eigenvector methods. However, recently Atkins, Boman and Hendrikson [1] have shown how to use an eigenvector of the Laplacian matrix to sequence relational data. There is an obvious parallel between this method and the use of eigenvector methods to locate steady state random walks on graphs.

The aim in this paper is to exploit this seriation method to develop a spectral method for computing graph edit distance. The task of posing the inexact graph matching problem in a matrix setting has proved to be an elusive one. This is disappointing since a rich set of potential tools are available from the field of mathematics referred to as spectral graph theory. This is the term given to a family of techniques that aim to characterise the global structural properties of graphs using the eigenvalues and eigenvectors of the adjacency matrix [3]. In the computer vision literature there have been a number of attempts to use spectral properties for graph-matching, object recognition and image segmentation. Umeyama has an eigendecomposition method

Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV 2003) 2-Volume Set
0-7695-1950-4/03 $17.00 © 2003 IEEE

that matches graphs of the same size [20]. Borrowing ideas from structural chemistry, Scott and Longuet-Higgins were among the first to use spectral methods for correspondence analysis [16]. They showed how to recover correspondences via singular value decomposition on the point association matrix between different images. In keeping more closely with the spirit of spectral graph theory, Shapiro and Brady [18] developed an extension of the Scott and Longuet-Higgins method, in which point sets are matched by comparing the eigenvectors of the point proximity matrix. Horaud and Sossa[6] have adopted a purely structural approach to the recognition of line-drawings. Their representation is based on the immanental polynomials for the Laplacian matrix of the line-connectivity graph. Shokoufandeh, Dickinson and Siddiqi [19] have shown how graphs can be encoded using local topological spectra for shape recognition from large data-bases. Using the apparatus of the EM algorithm, Luo and Hancock [9] have returned to the method of Umeyama and have shown how it can be rendered robust to differences in graph-size and structural errors. Making use of a purely structural representation of the graph, they cast the problem into a matrix setting. The correspondance matching problem is posed as the maximum likelihood estimation of the matrix of correspondance indicators and solved using the apparatus of the EM algorithm. Although the algorithm of Luo and Hancock [9] is robust to structural errors and differences in graph-size, it is slow to converge. Furthermore, it is sensitive to initialisation.

By using the spectral seriation method, we are able to convert the graph into a string. This opens up the possibility of performing graph matching by performing string alignment and minimising the Levenshtein or edit distance [7, 22]. We can follow Wagner and use dynamic programming to evaluate the edit distance between strings and hence recover correspondences [22]. It is worth stressing that although there been attempts to extend the string edit idea to trees and graphs [23, 12, 14, 17], there is considerable current effort aimed at putting the underlying methodology on a rigourous footing.

2. Graph Seriation

Consider the graph $G = (V, E)$ with node index-set V and edge-set $E = \{(j_i, j_k)|(j_i, j_k) \in V \times V, j_i \neq j_k\}$. Associated with the graph is an adjacency matrix A whose elements are defined as follows

$$A(j_i, j_k) = \begin{cases} 1 & \text{if } (j_i, j_k) \in E \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Our aim is to assign the nodes of the graph to a sequence order which preserves the edge ordering of the nodes. This sequence can be viewed as an edge connected path on the graph. Let the path commence at the node j_1 and proceed via the sequence of edge-connected nodes $X = \{j_1, j_2, j_3, ...\}$ where $(j_i, j_{i-1}) \in E$. With these ingredients, the problem of finding the path can be viewed as one of seriation, subject to edge connectivity constraints.

As noted by Atkins, Boman and Hendrikson [1], many applied computational problems, such as sparse matrix envelope reduction, graph partitioning and genomic sequencing, involve ordering a set according to a permutation $\pi = \{\pi(j_1), \pi(j_2), \ldots, \pi(j_{|V|})\}$ so that strongly correlated tokens are placed next to one another. The seriation problem is that of finding the permutation π that satisfies the condition $\pi(j_i) \leq \pi(j_k) \leq \pi(j_l) \Rightarrow \{A(i,k) \geq A(i,l) \wedge A(k,l) \geq A(i,l)\}$. This task has been posed as a combinatorial optmisation problem which involves minimising the penalty function

$$g(\pi) = \sum_{i=1}^{|V|} \sum_{k=1}^{|V|} A(j_i, j_k) \big(\pi(j_i) - \pi(j_k)\big)^2$$

for a a real symmetric adjacency matrix A.

Unfortunately, the penalty function $g(\pi)$, as given above, does not impose edge connectivity constraints on the ordering computed during the minimisation process. Furthermore, it implies no directionality in the transition from the node indexed j_i to the one indexed j_{i+1}. To overcome these shortcomings, we turn our attention instead to the penalty function

$$g(\pi) = \sum_{i=1}^{|V|-1} A(j_i, j_{i+1}) \big(\pi(j_i) - \pi(j_{i+1})\big)^2 \tag{2}$$

where the nodes indexed j_i and j_{i+1} are edge connected. After some algebra, it is straightforward to show that

$$\begin{aligned} g(\pi) = & \sum_{i=1}^{|V|-1} A(j_i, j_{i+1}) (\pi(j_i)^2 + \pi(j_{i+1})^2) \\ & - 2 \sum_{i=1}^{|V|-1} A(j_i, j_{i+1}) \pi(j_i) \pi(j_{i+1}) \end{aligned} \tag{3}$$

It is important to note that $g(\pi)$ does not have a unique minimiser. The reason for this is that its value remains unchanged if we add a constant amount to each of the coefficients of π. We also note that it is desirable that the minimiser of $g(\pi)$ is defined up to a constant λ whose solutions are polynomials in the elements of A. Therefore, we subject the minimisation problem to the constraints

$$\lambda \pi(j_i)^2 = \sum_{k=1}^{|V|} A(j_k, j_i) \pi(j_k)^2 \text{ and } \sum_{k=1}^{|V|} \pi(j_k)^2 \neq 0 \tag{4}$$

Combining the constraint conditions given in Equation 4 with the definition of the penalty function given in Equation 3, it is straightforward to show that the permutation π satisfies the condition

$$\sum_{k=1}^{|V|}\sum_{i=1}^{|V|-1}\left(A(j_k,j_i)+A(j_k,j_{i+1})\right)\pi(j_k)^2 = \lambda\sum_{i=1}^{|V|-1}(\pi(j_i)^2+\pi(j_{i+1})^2) \tag{5}$$

Using matrix notation, we can write the above equation in the more compact form

$$HA\phi = \lambda H\phi \tag{6}$$

where $\phi = \{\pi(j_1)^2, \pi(j_2)^2, \ldots, \pi(j_{|V|})^2\}^T$ and H is the $(N-1)\times N$ matrix

$$H = \begin{bmatrix} 1 & 1 & 0 & \ldots & 0 \\ 0 & 1 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & 1 & 1 \end{bmatrix} \tag{7}$$

Hence it is clear that locating the permuation π that minimises $g(\pi)$ can be posed as an eigenvalue problem, and that ϕ is an eigenvector of A. This follows from the fact Equation 6 can be obtained by multiplying both sides of the eigenvector equation $A\phi = \lambda\phi$ by H. Furthermore, due to the norm condition of the eigenvector, the constraint $\sum_{k=1}^{|V|}\pi(j_k)^2 \neq 0$ is always satisfied. Taking this analysis one step further, we can premultiply both sides of Equation 6 by ϕ^T to obtain the matrix equation $\phi^T HA\phi = \lambda\phi^T H\phi$. As a result, it follows that

$$\lambda = \frac{\phi^T HA\phi}{\phi^T H\phi} \tag{8}$$

We note that the elements of the permutation π are required to be real. Consequently, the co-efficients of the eigenvector ϕ are always non-negative. Since the elements of the matrices H and A are positive, it follows that the quantities $\phi^T HA\phi$ and $\phi^T H\phi$ are positive. Hence, the set of solutions reduces itself to those that are determined up to a constant $\lambda > 0$.

With these observations in mind, we focus on proving the existence of a permutation that minimises $g(\pi)$ subject to the constraints in Equation 4, and demonstrating that this permutation is unique. To this end we use the Perron-Frobenius theorem [21]. This concerns the proof of existence regarding the eigenvalue $\lambda^* = \max_{i=1,2,\ldots,|V|}\{\lambda_i\}$ of a primitive, real, non-negative, symmetric matrix A, and the uniqueness of the corresponding eigenvector ϕ^*. The Perron-Frobenius theorem states that the eigenvalue $\lambda^* > 0$ has multiplicity one. Moreover, the co-efficients of the corresponding eigenvector ϕ^* are all positive and the eigenvector is unique. As a result the remaining eigenvectors of A have at least one negative co-efficient and one positive co-efficient. If the matrix A is non-stochastic in nature (i.e. the graph $G = (V, E)$ is not k-regular), ϕ^* is also known to be linearly independent of the all-ones vector $\mathbf{e} = [1, 1\ldots, 1]^T$. Hence, the leading eigenvector of A is the minimiser of $g(\pi)$.

The elements of the leading eigenvector ϕ^* can be used to construct a serial ordering of the nodes in the graph. We commence from the node associated with the largest component of ϕ^*. We then sort the elements of the leading eigenvector such that they are both in the decreasing magnitude order of the co-efficients of the eigenvector, and satisfy edge connectivity constraints on the graph.The procedure is a recursive one that proceeds as follows. At each iteration, we maintain a list of nodes visited. At iteration k let the list of nodes be denoted by $\mathcal{L}_k$. Initially, $\mathcal{L}_0 = j_o$ where $j_0 = \arg\max_j \phi^*(j)$, i.e. j_0 is the component of ϕ^* with the largest magnitude. Next, we search through the set of first neighbours $\mathcal{N}_{j_0} = \{k|(j_0, k) \in E\}$ of j_o to find the node associated with the largest remaining component of ϕ^*. The second element in the list is $j_1 = \arg\max_{l\in\mathcal{N}_{j_0}} \phi^*(l)$. The node index j_1 is appended to the list of nodes visited and the result is $\mathcal{L}_1$. In the kth (general) step of the algorithm we are at the node indexed j_k and the list of nodes visited by the path so far is $\mathcal{L}_k$. We search through those first-neighbours of j_k that have not already been traversed by the path. The set of nodes is $C_k = \{l|l \in \mathcal{N}_{j_k} \wedge l \notin \mathcal{L}_k\}$. The next site to be appended to the path list is therefore $j_{k+1} = \arg\max_{l\in C_k} \phi^*(l)$. This process is repeated until no further moves can be made. This occurs when $C_k = \emptyset$ and we denote the index of the termination of the path by T. The serial ordering of the nodes of the graph X is given by the ordered list or string of nodes indices $\mathcal{L}_T$.

There are similarities between the use of the leading eigenvector for seriation and the use of spectral methods to find the steady state random walk on a graph. There are more detailed discussions of the problem of locating the steady state random walk on a graph in the reviews by Lovasz [8] and Mohar [10]. An important result described in these papers, is that if we visit the nodes of the graph in the order defined by the magnitudes of the co-efficients of the leading eigenvector of the transition probability matrix, then the path is the steady state Markov chain. However, this path is not guaranteed to be edge connected. Hence, it can not ne used to impose a string ordering on the nodes of a graph. The seriation approach adopted in this paper does impose edge connectivity constraints and can hence be used to convert graphs to strings in a manner which is suitable for computing edit distance.

3. Edit Distance Computation

In practice we will be interested in finding the edit distance for a pair of graphs $G_X = (V_X, E_X)$ and $G_Y = (V_Y, E_Y)$. From now onwards, we refer to the graph $G_X = (V_X, E_X)$ as the model graph and to the graph $G_D = (V_Y, E_Y)$ as the data-graph. The leading eigenvectors of the corresponding adjacency matrices A_X and A_Y are respectively $\phi_X^* = \{\phi_X^*(1), \phi_X^*(2), \ldots, \phi_X^*(|V_X|)\}$ and $\phi_Y^* = \{\phi_Y^*(1), \phi_Y^*(2), \ldots, \phi_Y^*(|V_Y|)\}$. We denote the seriation for the model graph by $X = \{x_1, x_2, \ldots, x_{|V_X|}\}$. In a similar fashion, the seriation corresponding to the data graph is denoted by $Y = \{y_1, y_2, \ldots, y_{|V_Y|}\}$.

3.1. Probabilistic Framework

Our aim in this section is to develop a probabilistic framework for computing the edit distance between the graphs $G_X = (V_X, E_X)$ and $G_Y = (V_Y, E_Y)$. Here, we use the strings X and Y to index the rows and column of an edit lattice. The rows of the lattice are indexed using the data-graph string, while the columns are indexed using the model-graph string. To allow for differences in the sizes of the graphs we introduce a null symbol ϵ which can be used to pad the strings. We pose the problem of computing the edit distance as that of finding a path $\Gamma = < \gamma_1, \gamma_2, ...\gamma_k, ..., \gamma_L >$ through the lattice. Each element $\gamma_k \in (V_X \cup \epsilon) \times (V_X \cup \epsilon)$ of the edit path is a Cartesian pair. We constrain the path to be connected on the edit lattice. In particular, the transition on the edit lattice from the state γ_k to to the state γ_{k+1} is constrained to move in a direction that is increasing and connected in the horizontal, vertical or diagonal direction on the lattice. The diagonal transition corresponds to the match of an edge of the data graph to an edge of the model graph. A horizontal transition means that the data-graph index is not incremented, and this corresponds to the case where the traversed nodes of the model graph are null-matched. Similarly when a vertical transition is made, then the traversed nodes of the data-graph are null-matched.

To commence, we require some formality. Suppose that $\gamma_k = (a, b)$ and $\gamma_{k+1} = (c, d)$ represent adjacent states in the edit path between the seriations X and Y. According to the classical approach, the cost of the edit path is given by

$$d(X, Y) = C(\Gamma) = \sum_{\gamma_k \in \Gamma} \eta(\gamma_k \rightarrow \gamma_{k+1}) \tag{9}$$

where $\eta(\gamma_k \rightarrow \gamma_{k+1})$ is the cost of the transition between the states $\gamma_k = (a, b)$ and $\gamma_{k+1} = (c, d)$. The optimal edit path is the one that minimises the edit distance between string, and satisfies the condition $\Gamma^* = \arg\min_\Gamma C(\Gamma)$ and hence the edit distance is $d(X, Y) = C(\Gamma^*)$. Classically, the optimal edit sequence may be found using Dijkstra's algorithm [4] or by using the quadratic programming method of Wagner and Fisher [22].

However, in this paper we adopt a different approach. We aim to find the edit path that has maximum probability given the available leading eigenvectors of the data-graph and model-graph adjacency matrices. Hence, the optimal path is the one that satisfies the condition

$$\Gamma^* = \arg\max_\Gamma P(\Gamma | \phi_X^*, \phi_Y^*) \tag{10}$$

To develop this decision criterion into a practical edit distance computation scheme, we need to develop the *a posteriori* probability appearing above. We commence by using the definition of conditional probability to re-write the *a posteriori* path probability in terms of the joint probability density $P(\phi_X^*, \phi_Y^*)$ for the leading eigenvectors and the joint density function $P(\phi_X^*, \phi_Y^*, \Gamma)$ for the leading eigenvectors and the edit path. The result is

$$P(\Gamma | \phi_X^*, \phi_Y^*) = \frac{P(\phi_X^*, \phi_Y^*, \Gamma)}{P(\phi_X^*, \phi_Y^*)} \tag{11}$$

We can rewrite the joint density appearing in the numerator to emphasise the role of the components of the adjacency matrix leading eigenvectors and the component edit transitions explicit

$$P(\Gamma | \phi_X^*, \phi_Y^*) = \frac{P(\phi_X^*(1), \phi_X^*(2), ..., \phi_Y^*(1), \phi_Y^*(2)..., \gamma_1, \gamma_2, ...)}{P(\phi_X^*, \phi_Y^*)} \tag{12}$$

To simplify the numerator, we make a conditional independence assumption. Specifically, we assume that the components of the leading eigenvector of the adjacency matrices, depend only on the edit transition γ_k associated with their node-indices. Hence, we can perform the factorisation

$$P(\Gamma | \phi_X^*, \phi_Y^*) = \frac{\left\{ \prod_{k=1}^{L} P(\phi_X^*(a), \phi_Y^*(b) | \gamma_k) \right\} P(\gamma_1, \gamma_2, \ldots, \gamma_L)}{P(\phi_X^*, \phi_Y^*)} \tag{13}$$

where $P(\gamma_1, \gamma_2, ..., \gamma_L)$ is the joint prior for the sequence of edit transitions and (a, b) is the coordinate pair on the edit lattice associated with the state γ_k. To simplify the joint prior, we assume that transitions between sites that are not adjacent on the edit lattice are conditionally independent. As a result

$$P(\gamma_1, \gamma_2, \ldots, \gamma_L) = P(\gamma_L) \prod_{k=1}^{L-1} P(\gamma_k | \gamma_{k+1}) \tag{14}$$

This takes the form of a factorisation of conditional probabilities for transitions between sites on the edit lattice $P(\gamma_k | \gamma_{k+1})$, except for the term $P(\gamma_L)$ which results from the final site visited on the lattice. To arrive at a more homogeneous expression, we use the definition of conditional probability to re-express the joint conditional measurement

density for the adjacency matrix leading eigenvectors in the following form

$$P(\phi_X^*(a),\phi_Y^*(b)|\gamma_k)=\frac{P(\gamma_k|\phi_X^*(a),\phi_Y^*(b))P(\phi_X^*(a),\phi_Y^*(b))}{P(\gamma_k)} \tag{15}$$

Substituting Equations (14) and (15) into Equation (13) we find

$$P(\Gamma|\phi_X^*,\phi_Y^*)=\left\{\prod_{k=1}^{L}P(\gamma_k|\phi_X^*(a),\phi_Y^*(b))\frac{P(\gamma_k,\gamma_{k+1})}{P(\gamma_k)P(\gamma_{k+1})}\right\} \times\frac{\prod_{k=1}^{L}P(\phi_X^*(a),\phi_Y^*(b))}{P(\phi_X^*,\phi_Y^*)} \tag{16}$$

Since, the joint measurement density $P(\phi_X^*,\phi_Y^*)$ does not depend on the edit path, it does not influence the decision process and we remove it from further consideration. Hence, the optimal path across the edit lattice is

$$\Gamma^*=\arg\max_{\gamma_1,\gamma_2\ldots,\gamma_L}\left\{\prod_{k=1}^{L}\left(P(\gamma_k|\phi_X^*(a),\phi_Y^*(b))\frac{P(\gamma_k,\gamma_{k+1})}{P(\gamma_k)P(\gamma_{k+1})}\right)\right\} \tag{17}$$

The information concerning the structure of the edit path on the lattice is captured by the quantity

$$R_{k,k+1}=\frac{P(\gamma_k,\gamma_{k+1})}{P(\gamma_k)P(\gamma_{k+1})} \tag{18}$$

To establish a link with the classical edit distance picture presented in Section 3, we can re-write the optmimal edit path as a minmisation problem involving the negative logarithm of the *a posteriori* path probability. The optimal path is the one that satisfies the condition

$$\Gamma^*=\arg\min_{\gamma_1,\gamma_2\ldots,\gamma_L}\left\{\sum_{k=1}^{L}\left[-P(\gamma_k|\phi_X^*(a),\phi_Y^*(b))-\ln R_{k,k+1}\right]\right\} \tag{19}$$

As a result the elementary edit cost $\eta(\gamma_k\rightarrow\gamma_{k+1})=\eta((a,b)\rightarrow(c,d))$ from the site (a,b) and the edit lattice to the site (c,d) is

$$\eta(\gamma_k\rightarrow\gamma_{k+1})=-\big(\ln P(\gamma_k|\phi_X^*(a),\phi_Y^*(b)) +\ln P(\gamma_{k+1}|\phi_X^*(c),\phi_Y^*(d))+\ln R_{k,k+1}\big) \tag{20}$$

3.1.1 Lattice Transition Probabilities

Recently, we have reported a methodology which lends itself to the modelling of the edge compatibility quantity $R_{k,k+1}$ [11]. It is based on the idea of constraint corruption through the action of a label-error process.

The model leads to an expression for the compatibility that is devoid of free parameters and which depends on the edge-set in the graphs being matched. Details of the derivation of the model are omitted from this paper for reasons of brevity. The compatibility contingency table is

$$R_{k,k+1}=\begin{cases}\rho_X\rho_Y & \text{if } \gamma_k\rightarrow\gamma_{k+1} \text{ if } (a,c)\in E_Y \text{ and } (b,d)\in E_X\\ \rho_X & \text{if } \gamma_k\rightarrow\gamma_{k+1} \text{ if } (a,c)\in E_Y \text{ and } (b=\epsilon\vee d=\epsilon)\\ \rho_Y & \text{if } \gamma_k\rightarrow\gamma_{k+1} \text{ if } (a=\epsilon\vee c=\epsilon) \text{ and } (b,d)\in E_X\\ 0 & \text{if } (a=\epsilon\vee c=\epsilon) \text{ and } (b=\epsilon\vee d=\epsilon)\end{cases} \tag{21}$$

3.1.2 A Posteriori Correspondence Probabilities

The second model ingredient is the *a posteriori* probability of visiting a site on the lattice. Here we assume that the differences in the components of the adjacency matrix leading eigenvectors are drawn from a Guassian distribution. Hence,

$$P(\gamma_k|\phi_X^*(a),\phi_Y^*(b))=\begin{cases}\frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{1}{2\sigma^2}\right. & \text{if } (a\neq\epsilon\wedge b\neq\epsilon)\\ \left.(\phi_X^*(a)-\phi_Y^*(b))^2\right\} & \\ \alpha & \text{if } a=\epsilon \text{ or } b=\epsilon\end{cases} \tag{22}$$

3.2. Minimum Cost Path

With the edit costs at hand, we proceed to find the path that yields the minimum edit distance. Our adopted algorithm makes use of the fact that the minimum cost path along the edit lattice is composed of sub-paths that are also always of minimum cost. Hence, following Levenshtein [7] we compute a $|V_Y|\times|V_X|$ transition-cost matrix τ whose elements are computed recursively using the formula

$$\tau_{i,j}=\begin{cases}\eta(\gamma_i\rightarrow\gamma_j) & \text{if } j=1 \text{ and } i=1\\ \eta(\gamma_i\rightarrow\gamma_j)+\tau_{i,j-1} & \text{if } i=1 \text{ and } j\geq 2\\ \eta(\gamma_i\rightarrow\gamma_j)+\tau_{i-1,j} & \text{if } j=1 \text{ and } i\geq 2\\ \eta(\gamma_i\rightarrow\gamma_j)+\min(\tau_{i-1,j}, & \\ \quad\tau_{i,j-1},\tau_{i-1,j-1}) & \text{if } i\geq 2 \text{ and } j\geq 2\end{cases} \tag{23}$$

The matrix τ is a representation of the accumulated minimal costs of the path along the edit lattice constrained to horizontal, vertical and diagonal transitions between adjacent coordinates. The minimum cost path can be proven to be that of the path closest to the diagonal of the matrix [7]. As a result, the edit distance is given by the bottom rightmost element of the transition-cost matrix. Hence, $d(X,Y)=\tau_{|V_Y|,|V_X|}$.

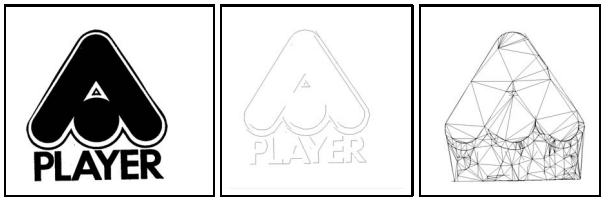
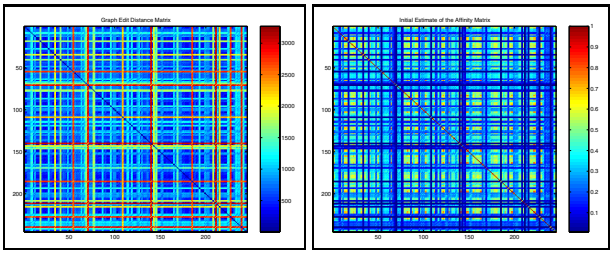


Figure 1. Example image, polygonalisation results and Delaunay triangulation



(a) (b)

Figure 2. Edit distance matrix and affinity matrix for the logotype-database.

4. Experiments

We have experimented with our new matching method on an application for performing automatic image database indexing and retrieval. We have performed experiments on a database containing 245 binary images of trademark-logotypes[1]. Here, the graphs are constructed as follows. First, we have extracted the set of straight line segments using the method in [13]. With the set of straight line segments at hand, we proceed to compute their center-points. The graphs used in our experiments are the Delaunay triangulations of these points. In the top left-hand-panel in Figure 1, we show an example of the images used in our experiments. The middle panel shows the results of the straight line extraction method. The corresponding Delaunay graph is displayed in the right-hand-panel.

4.1. Database Indexing

For indexing the database, we proceed as follows. We commence computing the complete set of distances between each of the distinct pairs of graphs. Once the set of distances is to hand, we construct the matrix D whose element $d_{i1,i2}$ is the edit distance $d_{i1,i2}$ between the graph indexed $i1$ and the graph indexed $i2$.

We commence the indexing process by applying the pairwise clustering algorithm of Robles-Kelly and Hancock [13] to the similarity data for the complete set of graphs in the database. The pairwise clustering algorithm requires distances to be represented by a matrix of pairwise affinity weights. Ideally, the smaller the distance, the stronger the weight, and hence the mutual affinity to a cluster. The affinity weights are required to be in the interval $[0, 1]$. Hence, for the pair of graphs indexed $i1$ and $i2$ the affinity weight is taken to be

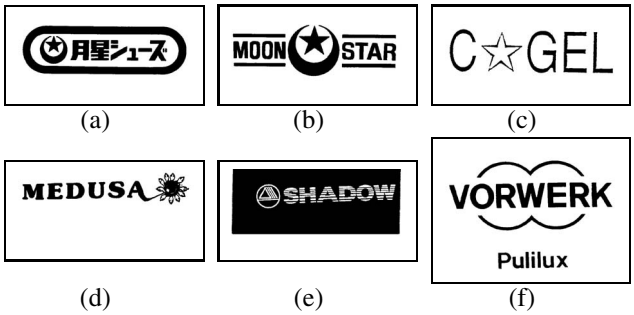
$$W_{i1,i2} = \exp\left(-\kappa \frac{d_{i1,i2}}{\max(D)}\right) \tag{24}$$

[1]All trademarks and logotypes remain the property of their respective owners. All trademarks and registered trademarks are used strictly for educational and academic purposes and without intent to infringe on the mark owners.

where κ is a constant. For all our experiments, we have set $\kappa = 3$. The clustering process is an iterative one that maintains two sets of variables. The first of these is a set of cluster membership indicators $s_{i\omega}^{(n)}$ which measures the affinity of the graph indexed i to the cluster Ω_ω indexed ω at iteration n of the algorithm. The second, is an estimate of the affinity matrix based on the current cluster-membership indicators $W^{(n)}$. These two sets of variables are estimated using interleaved update steps, which are formulated to maximise a likelihood function for the pairwise cluster configuration. The initial matrix of affinity weights computed by the clustering algorithm when indexing the database is shown in Figure 2b. The matrix D is displayed in Figure 2a.

Finally, we perform a further eigendecomposition to identify the graph corresponding to the center for each cluster in the database. To do this, we commence computing the cluster affinity matrix W^ω for the cluster indexed ω. Let G_{i1} and G_{i2} be two graphs corresponding to two images in the database indexed $i1$ and $i2$. The elements of the matrix W^ω are given by

$$W^\omega_{(i1,i2)} = \begin{cases} W_{(i1,i2)} & \text{if } (G_{i1}, G_{i2}) \in \Omega_\omega \\ 0 & \text{otherwise} \end{cases} \tag{25}$$



(a) (b) (c) (d) (e) (f)

Figure 3. Cluster-center and sample cluster members.

Recall that Sarkar and Boyer [15] have shown that the scalar quantity $\underline{x}_\omega^T W^\omega \underline{x}_\omega$, where W^ω is the affinity matrix and $\underline{x}_\omega = \{\underline{x}_\omega(1), \underline{x}_\omega(2), \ldots\}$ is a vector of cluster-membership variables, is maximised when $\underline{x}$ is the leading eigenvector of W^ω. Viewed in this way, each such matrix represents a distinct cluster and its leading eigenvector represents the individual cluster-membership affinities of the graphs. Hence, if $\underline{x}_\omega$ is the leading eigenvector of W^ω, the center of the cluster indexed ω is the graph $\hat{G}_\omega = \{G_{i1} \mid \underline{x}_\omega(i1) = \max_{k=1}^{|\Omega_\omega|}(\underline{x}_\omega(k))\}$.

In Figure 3, we show example images for the classes indexed $\omega = 1, 2$ (i.e. the first and second clusters). For the first cluster, we show the image corresponding to its center in Figure 3a. Two sample images corresponding to graphs in the first cluster are shown in Figures 3b and 3c. Figure 3d shows the center of the second cluster. Two sample images corresponding to graphs in the second cluster are displayed in Figures 3e and 3f. For our database, the indexing process extracted 34 clusters with an average of 7 images per cluster.

4.2. Search and Retrieval

Once the database has been indexed, we can perform search and retrieval making use of a query image. At input, we extract the set of straight line segments and compute their center-points. The graph G_q corresponding to the query image is obtained computing the Delaunay triangulation of these points.

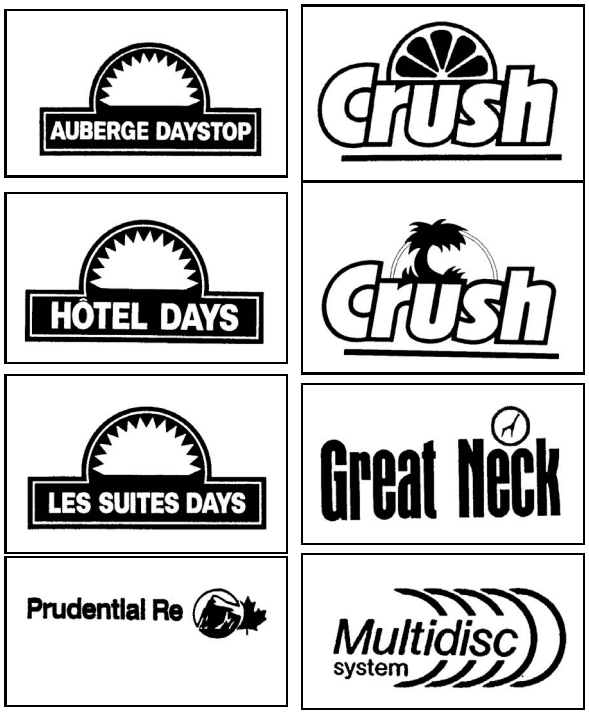


Figure 4. Top row: input query images; Bottom rows: search results.

As a consequence of the database indexing process outlined in the previous section, search and retrieval are straightforward tasks. The search process is as follows. First, we compute the set of graph edit distances $\hat{D}_q = \{\hat{d}_{q,1}, \hat{d}_{q,2}, \ldots, \hat{d}_{q,|\Omega|}\}$ between the graph G_q and the set of cluster-center graphs $\hat{\mathcal{G}} = \{\hat{G}_1, \hat{G}_2, \ldots, \hat{G}_{|\Omega|}\}$. Once the set of graph edit distances is to hand, we find the cluster index $\hat{\omega}$ such that $\hat{\omega} = \{l \mid l = \min_{k=1}^{|\Omega|}(\hat{d}_{q,k})\}$.

The query results, ordered by relevance, are given by the images corresponding to the set of graphs $\mathcal{G}^* = \{G_1^*, G_2^*, \ldots, G_{|\Omega_{\hat{\omega}}|}^*\}$ such that $G_l^* \in \Omega_{\hat{\omega}}$ subject to the condition $d_{q,1}^* < d_{q,2}^* < \ldots < d_{q,|\Omega_{\hat{\omega}}|}^*$, where $d_{q,l}^*$ is the graph edit distance between the graphs $G_l^* \in \mathcal{G}^*$ and G_q.

In Figure 4, we show the results of two query operations. In the top row, we show the two input query images. These are an image of a variant of the "Days Inn" logotype and a "Crush" logotype image. The three bottom rows show, in order of decreasing importance, from top-to-bottom, the three most relevant search results for each of the two input images.

It is worth noting that the database contains only two "Days Inn" logotype images and a single "Crush" logotype image. Hence, from our results, we can conclude the algorithm is able to cope with structural errors and differences in graph-size.

5. Conclusions

The work reported in this paper provides a synthesis of ideas from spectral graph-theory and structural pattern recognition. We use a graph spectral seriation method based on the leading eigenvector of the adjacency matrix to convert graphs into strings. We match the resulting string representations by finding the path on the edit lattice whose cost is minimum. The edit costs needed are computed using a simple probabilistic model of the edit transitions which is designed to preserve the edge order on the correspondences. The minimum cost edit sequence may be used to locate correspondences between nodes in the graphs under study.

We have demonstrated that both the edit sequence and the associated distance are of practical use. The edit sequence delivers correspondences that are robust to structural error and can be used to cluster graphs into meaningful classes for purposes of image-database indexing and retrieval.

References

[1] J. E. Atkins, E. G. Roman, and B. Hendrickson. A spectral algorithm for seriation and the consecutive ones problem. *SIAM Journal on Computing*, 28(1):297–310, 1998.

[2] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, 1997.

[3] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[4] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Math*, 1:269–271, 1959.

[5] M. A. Eshera and K. S. Fu. A graph distance measure for image analysis. *IEEE Transactions on Systems, Man and Cybernetics*, 14:398–407, 1984.

[6] R. Horaud and H. Sossa. Polyhedral object recognition by indexing. *Pattern Recognition*, 28(12):1855–1870, 1995.

[7] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Sov. Phys. Dokl.*, 6:707–710, 1966.

[8] L. Lovász. Random walks on graphs: a survey. *Bolyai Society Mathematical Studies*, 2(2):1–46, 1993.

[9] B. Luo and E. R. Hancock. Structural graph matching using the EM algorithm and singular value decomposition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 23(10):1120–1136, 2001.

[10] B. Mohar. Some applications of laplace eigenvalues of graphs. In G. Hahn and G. Sabidussi, editors, *Graph Symmetry: Algebraic Methods and Applications*, NATO ASI Series C, pages 227–275, 1997.

[11] R. Myers, R. C. Wilson, and E. R. Hancock. Bayesian graph edit distance. *PAMI*, 22(6):628–635, June 2000.

[12] B. J. Oommen and K. Zhang. The normalized string editing problem revisited. *PAMI*, 18(6):669–672, June 1996.

[13] A. Robles-Kelly and E. R. Hancock. Pairwise clustering with matrix factorisation and the EM algorithm. In *European Conference on Computer Vision*, pages 63–67, 2002.

[14] A. Sanfeliu and K. S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man and Cybernetics*, 13:353–362, 1983.

[15] S. Sarkar and K. L. Boyer. Quantitative measures of change based on feature organization: Eigenvalues and eigenvectors. *Computer Vision and Image Understanding*, 71(1):110–136, 1998.

[16] G. Scott and H. Longuet-Higgins. An algorithm for associating the features of two images. In *Proceedings of the Royal Society of London*, number 244 in B, pages 21–26, 1991.

[17] L. G. Shapiro and R. M. Haralick. Relational models for scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4:595–602, 1982.

[18] L. S. Shapiro and J. M. Brady. A modal approach to feature-based correspondence. In *British Machine Vision Conference*, pages 78–85, 1991.

[19] A. Shokoufandeh, S. J. Dickinson, K. Siddiqi, and S. W. Zucker. Indexing using a spectral encoding of topological structure. In *Proceedings of the Computer Vision and Pattern Recognition*, pages 491–497, 1998.

[20] S. Umeyama. An eigen decomposition approach to weighted graph matching problems. *PAMI*, 10(5):695–703, September 1988.

[21] R. S. Varga. *Matrix Iterative Analysis*. Springer, second edition, 2000.

[22] R. A. Wagner and M. J. Fisher. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

[23] J. T. L. Wang, B. A. Shapiro, D. Shasha, K. Zhang, and K. M. Currey. An algorithm for finding the largest approximately common substructures of two trees. *PAMI*, 20(8):889–895, August 1998.