

Graph-Theoretical Methods in Computer Vision

Ali Shokoufandeh¹ and Sven Dickinson²

¹ Department of Mathematics and Computer Science
Drexel University
Philadelphia, PA
USA

ashokoufandeh@mcs.drexel.edu

² Department of Computer Science and Center for Cognitive Science
Rutgers University
New Brunswick, NJ
USA
sven@cs.rutgers.edu

Abstract. The management of large databases of hierarchical (e.g., multi-scale or multilevel) image features is a common problem in object recognition. Such structures are often represented as trees or directed acyclic graphs (DAGs), where nodes represent image feature abstractions and arcs represent spatial relations, mappings across resolution levels, component parts, etc. Object recognition consists of two processes: *indexing* and *verification*. In the indexing process, a collection of one or more extracted image features belonging to an object is used to select, from a large database of object models, a small set of candidates likely to contain the object. Given this relatively small set of candidates, a verification, or matching procedure is used to select the most promising candidate. Such matching problems can be formulated as *largest isomorphic subgraph* or *largest isomorphic subtree* problems, for which a wealth of literature exists in the graph algorithms community. However, the nature of the vision instantiation of this problem often precludes the direct application of these methods. Due to occlusion and noise, no significant isomorphisms may exist between two graphs or trees. In this paper, we review our application of spectral encoding of a graph for indexing to large database of image features represented as DAGs. We will also review a more general class of matching methods, called *bipartite matching*, to two problems in object recognition.

1 Introduction

The management of large databases of hierarchical (e.g., multi-scale or multi-level) image features is a common problem in object recognition. Such structures are often represented as trees or DAGs, where nodes represent image feature abstractions and arcs represent spatial relations, mappings across resolution levels, component parts, etc. Object recognition consists of two processes: indexing and verification. In the indexing process, a collection of one or more extracted image features belonging to an object is used to select, from a large database of

object models, a small set of candidates likely to contain the object. Given this relatively small set of candidates, a verification, or matching procedure is used to select the most promising candidate. The requirements of matching include computing a correspondence between nodes in an image structure and nodes in a model structure, as well as computing an overall measure of distance (or, alternatively, similarity) between the two structures. Such matching problems can be formulated as *largest isomorphic subgraph* or *largest isomorphic subtree* problems, for which a wealth of literature exists in the graph algorithms community. However, the nature of the vision instantiation of this problem often precludes the direct application of these methods. Due to occlusion and noise, no significant isomorphisms may exist between two graphs or trees. Yet, at some level of abstraction, the two structures (or two of their substructures) may be quite similar.

In this paper, we review our application of spectral encoding of a graph for indexing to large database of image features represented as DAG. Our indexing mechanism maps the topological structure of a DAG into a low-dimensional vector space, based on eigenvalue characterization of its adjacency matrix. Invariant to any re-ordering of the DAG's branches, the vector provides an invariant signature of the shape's coarse topological structure. Furthermore, we can efficiently index into a database of topological signatures to retrieve model objects having similar topology. In a set of experiments, we show that the indexing mechanism is very effective in selecting a small set of model candidates that contain the correct object.

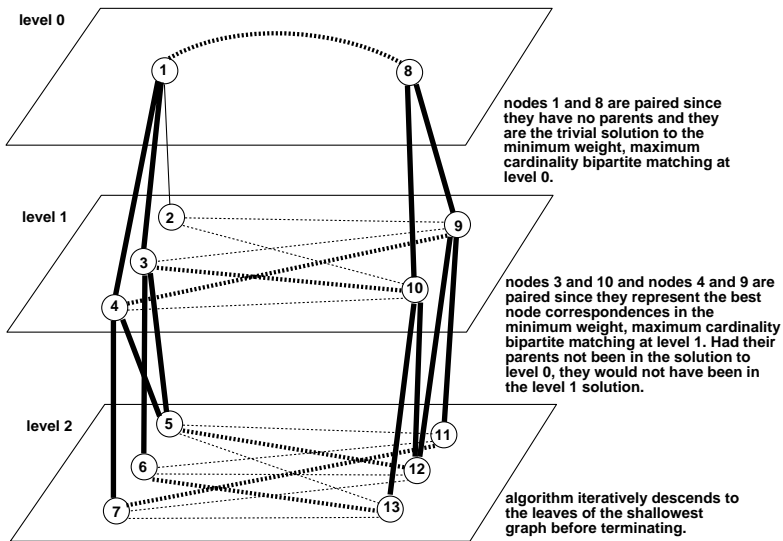


Fig. 1. Bipartite Matching

We will also review our application of a more general class of matching methods, called *bipartite matching*, to problems in object recognition [25,26,28,30,31]. As shown in Fig. 1, given two graphs (or trees) G_1 and G_2 , $H(G_1, G_2, E)$ is a weighted bipartite graph with weight matrix $W = [w_{u,v}]$ of size $|G_1| \times |G_2|$, for all edges of the form $(u, v) \in E$, $u \in G_1$, $v \in G_2$, and (u, v) has an associated weight $= w_{u,v}$. Solving the maximum cardinality minimum weight matching in H solves an optimization problem which tries to minimize total edge weight on one hand while trying to maximize the total number of edges in the solution set on the other hand. The time complexity for finding such a matching in a weighted bipartite graph with n vertices is $O(n^2\sqrt{n \log \log n})$ time, using the scaling algorithm of Gabow, Gomans and Williamson [11].

We will apply this framework for solving two object recognition problems, one involving DAGs and one involving rooted trees. Each algorithm will, as an integral step, compute the maximum cardinality, minimum weight matching in a bipartite graph. Furthermore, each algorithm, in turn, takes a different approach to preserving hierarchical order in the solution. We describe each algorithm in detail and evaluate its performance on sets of real images.

2 Two Object Recognition Domains

2.1 The Saliency Map Graph

Our first image representation is a multi-scale view-based description of 3-D objects that, on one hand, avoids the need for complex feature extraction, such as lines, curves, or regions, while on the other hand, provides the locality of representation necessary to support occluded object recognition as well as invariance to minor changes in both illumination and shape. In computing a representation for a 2-D image, a multi-scale wavelet transform is applied to the image, resulting in a hierarchical map that captures salient regions at their appropriate scales of resolution. Each such region maps to a node in a DAG, in which an arc is directed from a coarser scale region to a finer scale region if the center of the finer scale's region falls within the interior of the coarser scale's region. The resulting hierarchical graph structure, called the *saliency map graph* (SMG), encodes both the topological and geometrical information found in the saliency map. An example of an image and its corresponding saliency map graph are shown in Figs 2(a) and (b), respectively. Details of the representation, including its computation and invariance properties, can be found in [25,26,28].

2.2 Shock Trees

Our second image representation describes the generic shape of a 2-D object, and is based on a coloring of the shocks (singularities) of a curve evolution process acting on simple closed curves in the plane [15]. Intuitively, the taxonomy of shocks consists of four distinct types: the radius function along the medial axis varies monotonically at a 1, achieves a strict local minimum at a 2, is constant at

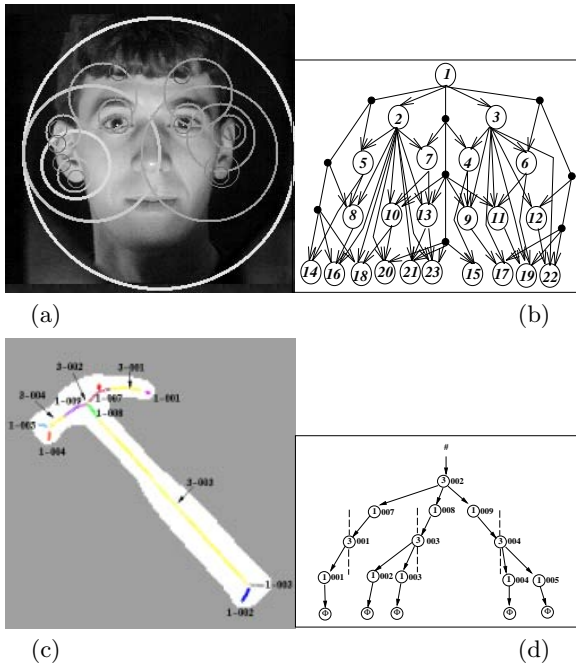


Fig. 2. Two Object Recognition Domains: (a) example image; (b) saliency map graph corresponding to image in (a); (c) example silhouette with computed shocks; (d) shock tree corresponding to silhouette in (c).

a 3 and achieves a strict local maximum at a 4. We have recently abstracted this system of shocks into a *shock graph* where vertices are labelled by their shock types, and the shock formation times direct the edges. The space of such shock graphs is completely characterized by a small number of rules, which in turn permits the reduction of each graph to a *unique rooted tree* [30,31]. Figure 2(c) and (d) show the the 2-D silhouette of a hammer and its corresponding shock tree, respectively.

3 Indexing Mechanism for Directed Acyclic Graphs

3.1 An Eigenvalue Characterization of a DAG

To describe the topology of a DAG, we turn to the domain of eigenspaces of graphs, first noting that any graph can be represented as a $\{-1, 0, 1\}$ adjacency matrix, with 1's (and -1's) indicating directed edges in the graph (and 0's on the diagonal). The eigenvalues of a graph's (or DAG's) adjacency matrix encode important structural properties of the graph (or DAG). Furthermore, the eigenvalues of a symmetric matrix A are invariant to any orthonormal transformation of the form $P^t A P$. Since a permutation matrix is orthonormal, the eigenvalues

of a DAG are invariant to any consistent re-ordering of the DAG's branches. However, before we can exploit a DAG's eigenvalues for matching purposes, we must establish their stability under minor topological perturbation, due to noise, occlusion, or deformation.

We begin with the case in which the image DAG is formed by either adding a new root to the model DAG, adding one or more subgraphs at leaf nodes of the model DAG, or deleting one or more entire model subgraphs. In this case, the model DAG is a subgraph of the query DAG, or vice versa. The following theorem relates the eigenvalues of two such DAGs:

Theorem 1 (see Cvetković et al. [6]). *Let A be a symmetric¹ matrix with eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$ and let B be one of its principal² sub-matrices. If the eigenvalues of B are $\nu_1 \geq \nu_2 \geq \dots \geq \nu_m$, then $\lambda_{n-m+i} \leq \nu_i \leq \lambda_i$ ($i = 1, \dots, m$).*

This important theorem, called the *Interlacing Theorem*, implies that as A and B become less similar (in the sense that one is a smaller subgraph of the other), their eigenvalues become proportionately less similar (in the sense that the intervals that contain them increase in size, allowing corresponding eigenvalues to drift apart).

The other case we need to consider consists of a query DAG formed by adding to or removing from the model DAG, a small subset of internal (i.e., non-leaf) nodes. The upper bounds on the two largest eigenvalues ($\lambda_1(T)$ and $\lambda_2(T)$) of any DAG, T , with n nodes and maximum degree $\Delta(T)$ are $\lambda_1(T) \leq \sqrt{n-1}$ and $\lambda_2(T) \leq \sqrt{(n-3)/2}$, respectively (Neumaier, 1982 [18]). The lower bounds on these two eigenvalues are $\lambda_1(T) \geq \sqrt{\Delta(T)}$ (Nosal, 1970 [19]) and $\lambda_1(T)\lambda_2(T) \geq \frac{2n-2}{n-2}$ (Cvetković, 1971 [5]). Therefore, the addition or removal of a small subset of internal nodes will result in a small change in the upper and lower bounds on these two eigenvalues. As we shall next, our topological description exploits the largest eigenvalues of a DAG's adjacency matrix. Since these largest eigenvalues are stable under minor perturbation of the DAG's internal node structure, so too is our topological description.

We now seek a compact representation of the DAG's topology based on the eigenvalues of its adjacency matrix. We could, for example, define a vector to be the sorted eigenvalues of a DAG. The resulting index could be used to retrieve nearest neighbors in a model DAG database having similar topology. There is a problem with this approach. For large DAGs, the dimensionality of the signature would be prohibitively large. To solve this problem, this description will be based on eigenvalue sums rather than on the eigenvalues themselves.

Specifically, let T be a DAG whose maximum branching factor is $\Delta(T)$, and let the subgraphs of its root be T_1, T_2, \dots, T_S . For each subgraph, T_i , whose root degree is $\delta(T_i)$, compute the eigenvalues of T_i 's sub-matrix, sort the eigenvalues

¹ The original theorem is stated for Hermitian matrices, of which symmetric matrices are a subclass.

² A principal sub-matrix of a graph's adjacency matrix is formed by selecting the rows and columns that correspond to a subset of the graph's nodes.

in decreasing order by absolute value, and let S_i be the sum of the $\delta(T_i) - 1$ largest absolute values. As shown in Fig. 3, the sorted S_i 's become the components of a $\Delta(T)$ -dimensional vector assigned to the DAG's root. If the number of S_i 's is less than $\Delta(T)$, then the vector is padded with zeroes. We can recursively repeat this procedure, assigning a vector to the root of each subgraph in the DAG for reasons that will become clear in the next section.

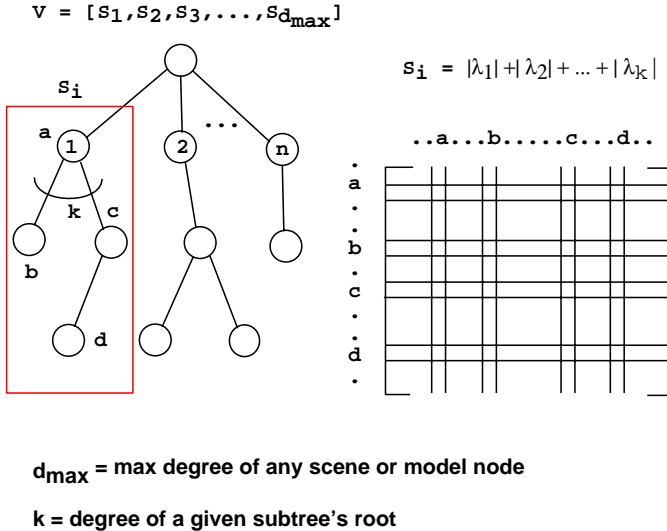


Fig. 3. Computing a Topological Signature of a DAG

Although the eigenvalue sums are invariant to any consistent re-ordering of the DAG's branches, we have given up some uniqueness (due to the summing operation) in order to reduce dimensionality. We could have elevated only the largest eigenvalue from each subgraph (non-unique but less ambiguous), but this would be less representative of the subgraph's structure. We choose the $\delta(T_i) - 1$ -largest eigenvalues for two reasons: 1) the largest eigenvalues are more informative of subgraph structure, 2) by summing $\delta(T_i) - 1$ elements, we effectively normalize the sum according to the local complexity of the subgraph root.

To efficiently compute the sub-matrix eigenvalue sums, we turn to the domain of semidefinite programming. A symmetric $n \times n$ matrix A with real entries is said to be positive semidefinite, denoted as $A \succeq 0$, if for all vectors $x \in R^n$, $x^t A x \geq 0$, or equivalently, all its eigenvalues are non-negative. We say that $U \succeq V$ if the matrix $U - V$ is positive semidefinite. For any two matrices U and V having the same dimensions, we define $U \bullet V$ as their inner product, i.e.,

$U \bullet V = \sum_i \sum_j U_{i,j} V_{i,j}$. For any square matrix U , we define $\text{trace}(U) = \sum_i U_{i,i}$.

Let I denote the identity matrix having suitable dimensions. The following result, due to Overton and Womersley [20], characterizes the sum of the first k largest eigenvalues of a symmetric matrix in the form of a semidefinite convex programming problem:

Theorem 2 (Overton and Womersley [20]). *For the sum of the first k eigenvalues of a symmetric matrix A , the following semidefinite programming characterization holds:*

$$\begin{aligned} \lambda_1(A) + \dots + \lambda_k(A) = \max A \bullet U \\ \text{s.t. } \text{trace}(U) = k \\ 0 \preceq U \preceq I, \end{aligned} \quad (1)$$

The elegance of Theorem (2) lies in the fact that the equivalent semidefinite programming problem can be solved, for any desired accuracy ϵ , in time polynomial in $O(n\sqrt{n}L)$ and $\log \frac{1}{\epsilon}$, where L is an upper bound on the size of the optimal solution, using a variant of the Interior Point method proposed by Alizadeh [1]. In effect, the complexity of directly computing the eigenvalue sums is a significant improvement over the $O(n^3)$ time required to compute the individual eigenvalues, sort them, and sum them.

3.2 A Database for Model DAGs

Our eigenvalue characterization of a DAG’s topology suggests that a model DAG’s topological structure can be represented as a vector in δ -dimensional space, where δ is an upper bound on the degree of any vertex of any image or model DAG. If we could assume that an image DAG represents a properly segmented, unoccluded object, then the vector of eigenvalue sums, call it the *topological signature vector* (or TSV), computed at the image DAG’s root could be compared with those topological signature vectors representing the roots of the model DAGs. The vector distance between the image DAG’s root TSV and a model DAG’s root TSV would be inversely proportional to the topological similarity of their respective DAGs: recall that finding two subgraphs with “close” eigenvalue sums represents an approximation to finding the largest isomorphic subgraph.

Unfortunately, this simple framework cannot support either cluttered scenes or segmentation errors, both of which result in the addition or removal of DAG structure. In either case, altering the structure of the DAG will affect the TSV computed at its nodes. The signatures corresponding to those subgraphs that survive the occlusion will not change. However, the signature of a node having one or more subgraphs which have undergone any perturbation will change which, in turn, will affect the signatures of any of its ancestor nodes, including the root. We therefore cannot rely on indexing solely with the root’s signature. Instead, we will take advantage of the local subgraphs that survive the occlusion.

We can accommodate such perturbations through a local indexing framework analogous to that used in a number of geometric hashing methods, e.g., [17,10]. Rather than storing a model DAG’s root signature, we will store the signatures of *each* node in the model DAG, along with a pointer to the object model containing that node as well as a pointer to the corresponding node in the model DAG (allowing access to node label information). Since a given model subgraph can be shared by other model DAGs, a given signature (or location in δ -dimensional space) will point to a list of (model object, model node) ordered pairs. At runtime, the signature at each node in the image DAG becomes a separate index, with each nearby candidate in the database “voting” for one or more (model object, model node) pairs. To quickly retrieve these nearby candidates, we will pre-compute the sorted pairwise distances between every signature in the database and every other signature in the database.

3.3 An Efficient Indexing Mechanism

We achieve efficient indexing through a δ -dimensional Voronoi decomposition $P(B)$ of the model space $V(B)$. For a given image TSV, the Voronoi decomposition will allow us to find the nearest model TSV in expected $O(\log^\delta(kn))$ time for fixed δ [22]. From the ranked list of neighbors L computed for each model TSV, either the ℓ nearest neighbors or all neighbors within a radius r can be computed in constant time (assuming fixed ℓ or r). To construct the Voronoi database, we partition R^δ into regions, so that for each vector $v \in V(B)$, the region $P(v)$ will denote the set of points in R^δ which are closer to v than any other vector in $V(B)$ with respect to a metric norm, such as $d(.,.)$. Such a partition is well-defined. The complexity of the Voronoi decomposition is $O((kn)^{\lfloor(\delta+1)/2\rfloor+1}) + O((kn)^{\lfloor(\delta+1)/2\rfloor} \log(kn))$ ([22]), although this is a cost incurred at preprocessing time.

The process of selecting candidates at runtime is shown in Fig. 4. Let H be an image DAG, and let F_H and $V(F_H)$ be defined as before, and let $d(u, v) = \|v - u\|_2$. For each TSV $h \in V(F_H)$, we will find the region $P(v)$ (and corresponding vector v) in $P(B)$, in which h resides. Using the list $L(v)$, we will find the set of model TSV’s $\{u_1, \dots, u_\ell\}$ such that $d(h, v) + d(v, u_i) \leq r$. Clearly, since the metric norm $d(.,.)$ satisfies the triangle inequality, the set $\{v\} \cup \{u_1, \dots, u_\ell\}$ is a subset of the TSV’s whose distance from h is less than r . Each node in the image DAG therefore leads to a number of (model object, model node) candidate votes. In the next section, we discuss the weighting of these votes, along with the combination of the evidence over all nodes in the image DAG.

4 Matching Two Saliency Map Graphs

Given the SMG computed for an input image to be recognized and a SMG computed for a given model object image (view), we propose two methods for computing their similarity. In the first method, we compare only the topological or structural similarity of the graphs, a weaker distance measure designed to

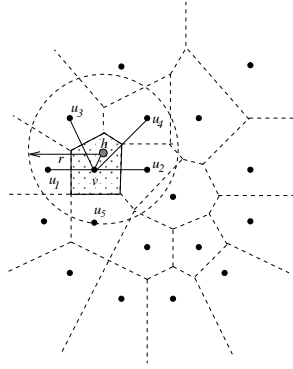


Fig. 4. Selecting the Candidate Model Objects

support limited object deformation invariance. In the second method, we take advantage of the geometrical information encoded in an SMG and strengthen the similarity measure to ensure geometric consistency, a stronger distance measure designed to support subclass or instance matching. Each method is based on formulating the problem as a maximum cardinality minimum weight matching in a bipartite graph.

4.1 Problem Formulation

Two graphs $G = (V, E)$ and $G' = (V', E')$ are said to be isomorphic if there exists a bijective mapping $f : V \rightarrow V'$ satisfying, for all $x, y \in V$ $(x, y) \in E \Leftrightarrow (f(x), f(y)) \in E'$. To compute the similarity of two SMG's, we consider a generalization of the graph isomorphism problem, which we will call the *SMG similarity problem*: Given two SMG's $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ and a partial mapping from $f : V_1 \rightarrow V_2$, let \mathcal{E} be a real-valued error function defined on the set of all partial mappings. Our error function, \mathcal{E} , incorporates two components with respect to any partial mapping: 1) we would like to reward corresponding nodes which are similar in terms of their topology, geometry, and salience; and 2) we would like to penalize a set of correspondences the more they exclude nodes from the model. Specifically,

$$\mathcal{E}(f) = \varepsilon \sum_{u \in V_1, v \in V_2} M_{u,v} \omega(u, v) |s(u) - s(v)| + (1 - \varepsilon) \sum_{u \in V_1, f(u) = \emptyset} s(u) \quad (2)$$

where $\varepsilon = |\mathbf{1}^t M(f) \mathbf{1}| / (|V_1| + |V_2|)$ represents the fraction of matched vertices ($\mathbf{1}$ denotes the identity vector), $f(\cdot) = \emptyset$ for unmatched vertices, and $s(\cdot)$ represents region saliency. For the SMG topological similarity, Sect. 4.2, $\omega(\cdot, \cdot)$ is always one, while for the SMG geometrical similarity, Sect. 4.3, it denotes the Euclidean

distance between the regions.³ A more detailed discussion of the error function is provided in [28]. We say that a partial mapping f is feasible if $f(x) = y$ implies that there are parents p_x of x and p_y of y , such that $f(p_x) = p_y$. Our goal is therefore to find a feasible mapping f which minimizes $\mathcal{E}(f)$.

4.2 A Matching Algorithm Based on Topological Similarity

In this section, we describe an algorithm which finds an approximate solution to the SMG similarity problem. The focus of the algorithm is to find a minimum weight matching between vertices of G_1 and G_2 which lie in the same level. Our algorithm starts with the vertices at level 1. Let A_1 and B_1 be the set of vertices at level 1 in G_1 and G_2 , respectively. We construct a complete weighted bipartite graph $G(A_1, B_1, E)$ with a weight function defined for edge (u, v) ($u \in A_1$ and $v \in B_1$) as $w(u, v) = |s(v) - s(u)|$.⁴ Next, we find a maximum cardinality, minimum weight matching M_1 in G using [8]. All the matched vertices are mapped to each other; that is, we define $f(x) = y$ if (x, y) is a matching edge in M_1 .

The remainder of the algorithm proceeds in phases as follows, as shown in Fig. 5. In phase i , the algorithm considers the vertices of level i . Let A_i and B_i be the set of vertices of level i in G_1 and G_2 , respectively. Construct a weighted bipartite graph $G(A_i, B_i, E)$ as follows: (v, u) is an edge of G if either of the following is true: (1) Both u and v do not have any parent in G_1 and G_2 , respectively, or (2) They have at least one matched parent of depth less than i ; that is, there is a parent p_u of u and p_v of v such that $(p_u, p_v) \in M_j$ for some $j < i$. We define the weight of the edge (u, v) to be $|s(u) - s(v)|$. The algorithm finds a maximum cardinality, minimum weight matching in G and proceeds to the next phase.

The above algorithm terminates after ℓ phases, where ℓ is the minimum number of scales in the saliency maps (or SMG's) of two graphs. The partial mapping M of SMG's can be simply computed as the union of all M_i values for $i = 1, \dots, \ell$. Finally, using the error measure defined in [28], we compute the error of the partial mapping M . Each phase of the algorithm requires simple operations with the time to complete each phase being dominated by the time to compute a minimum weight matching in a bipartite graph. As mentioned in Sect. 1, the time complexity for finding such a matching in a weighted bipartite graph with n vertices is $O(n^2\sqrt{n \log \log n})$ time, using the scaling algorithm of Gabow, Gomans and Williamson [11]. The entire procedure, as currently formulated, requires $O(\ell n^2\sqrt{n \log \log n})$ steps.

³ For perfect similarity $\mathcal{E}(f) = 0$, while $\mathcal{E}(f)$ will be $\sum_{u \in V_1} s(u)$ if there is no match.

⁴ $G(A, B, E)$ is a weighted bipartite graph with weight matrix $W = [w_{ij}]$ of size $|A| \times |B|$ if, for all edges of the form $(i, j) \in E$, $i \in A$, $j \in B$, and (i, j) has an associated weight $= w_{i,j}$.

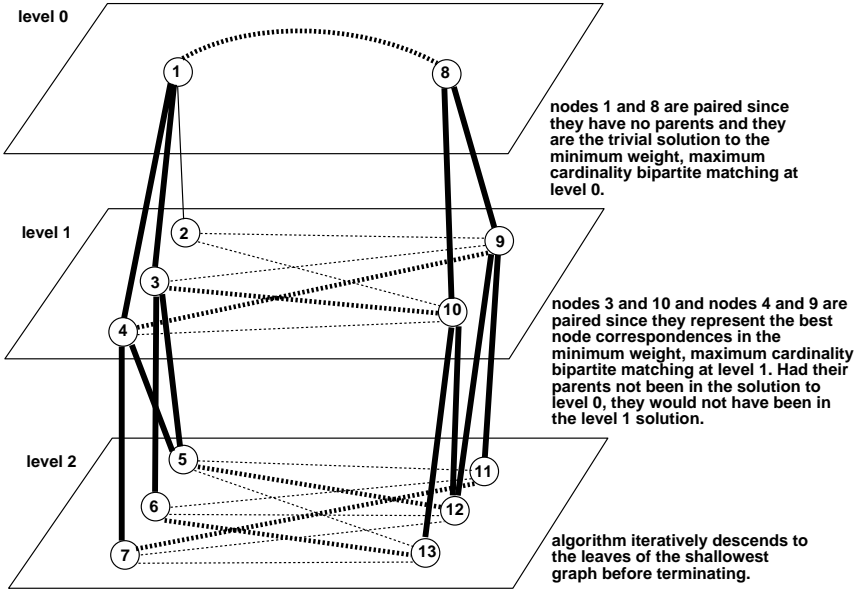


Fig. 5. Illustration of the SMGBM Algorithm (see text for explanation).

4.3 A Matching Algorithm Based on Geometric Similarity

The SMGBM similarity measure captured the structural similarity between two SMG’s in terms of branching factor and node saliency similarity; no geometric information encoded in the SMG was exploited. In this section, we describe a second similarity measure, called SMG Similarity using an Affine Transformation (SMGAT), that includes the geometric properties (e.g., relative position and orientation) of the saliency regions.

Given $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, we first assume, without loss of generality, that $|V_1| \leq |V_2|$. First, as shown in Fig. 6, the algorithm will hypothesize a correspondence between three regions of G_1 , say (r_1, r_2, r_3) , and three regions (r'_1, r'_2, r'_3) of G_2 . The mapping $\{(r_1 \rightarrow r'_1), (r_2 \rightarrow r'_2), (r_3 \rightarrow r'_3)\}$ will be considered as a basis for alignment if the following conditions are satisfied:

- r_i and r'_i have the same level in the SMG’s, for all $i \in \{1, \dots, \ell\}$.
- $(r_i, r_j) \in E_1$ if and only if $(r'_i, r'_j) \in E_2$, for all $i, j \in \{1, \dots, \ell\}$, which implies that selected regions should have the same adjacency structure in their respective SMG’s.

Once regions (r_1, r_2, r_3) and (r'_1, r'_2, r'_3) have been selected, we solve for the affine transformation (A, b) , that aligns the corresponding region triples by solving the following system of linear equalities:

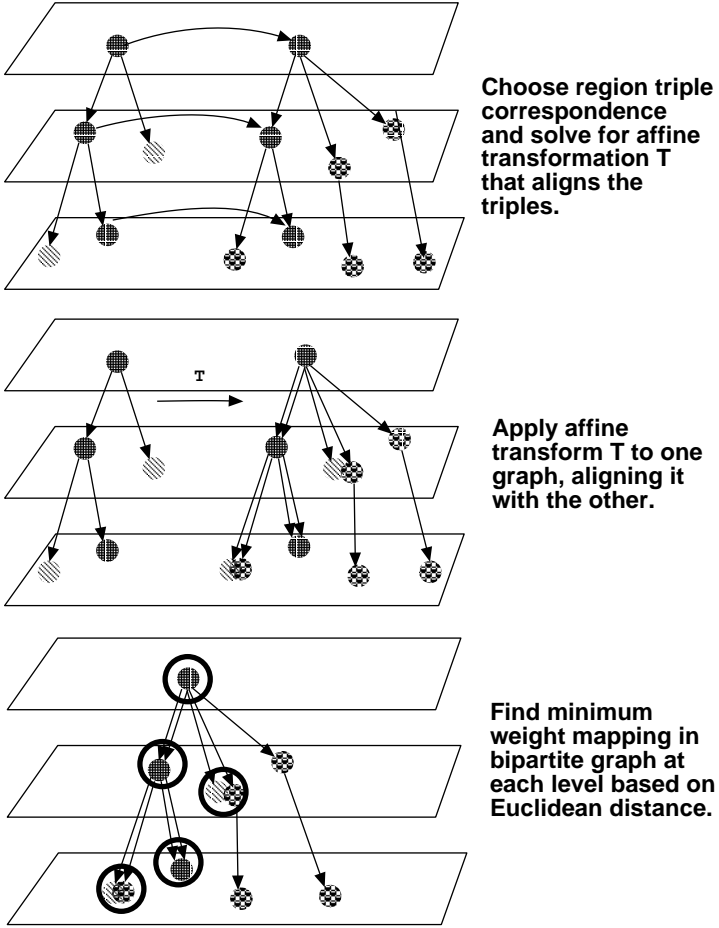


Fig. 6. Illustration of the SMGAT Algorithm (see text for explanation)

$$\begin{bmatrix} x_{r_1} & y_{r_1} & 1 & 0 & 0 & 0 \\ x_{r_2} & y_{r_2} & 1 & 0 & 0 & 0 \\ x_{r_3} & y_{r_3} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_{r_1} & y_{r_1} & 1 \\ 0 & 0 & 0 & x_{r_2} & y_{r_2} & 1 \\ 0 & 0 & 0 & x_{r_3} & y_{r_3} & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ b_1 \\ a_{21} \\ a_{22} \\ b_2 \end{bmatrix} = \begin{bmatrix} x_{r'_1} \\ x_{r'_2} \\ x_{r'_3} \\ y_{r'_1} \\ y_{r'_2} \\ y_{r'_3} \end{bmatrix}. \quad (3)$$

The affine transformation (A, b) will be applied to all regions in G_1 to form a new graph G' . Next, a procedure similar to the minimum weight matching, used in the SMGBM is applied to the regions in graphs G' and G_2 . Instead of matching regions which have maximum similarity in terms of saliency, we match regions which have minimum Euclidean distance from each other. Given two regions u

and v , the distance between them can be defined as the L_2 norm of the distance between their centers, denoted by $d(u, v) = \sqrt{(x_u - x_v)^2 + (y_u - y_v)^2}$. In a series of steps, SMGAT constructs weighted bipartite graphs $\mathcal{G}_i = (R_i, R'_i, E_i)$ for each level i of the two SMG's, where R_i and R'_i represent the set of vertices of G' and G_2 at the i -th level, respectively. The constraints for having an edge in E_i are the same as SMGBM: (u, v) is an edge in \mathcal{G}_i if either of the followings holds:

- Both u and v do not have any parents in G' and G_2 , respectively.
- They have at least one matched parent of depth less than i .

The corresponding edge will have weight equal to $w(u, v) = d(u, v)$. A maximum cardinality, minimum weight bipartite matching M_i will be found for each level \mathcal{G}_i , and the partial mapping $f_{(A,b)}$ for the affine transformation (A, b) will be formed as the union of all M_i 's. Finally, the error of this partial mapping $\mathcal{E}(f_{(A,b)})$ will be computed as the sum over each E_i of the Euclidean distance separating E_i 's nodes weighted by the nodes' difference in saliency. Once the total error is computed, the algorithm proceeds to the next valid pair of region triples. Among all valid affine transformations, SMGAT chooses that one which minimizes the error of the partial mapping.

In terms of algorithmic complexity, solving for the affine transformation (3) takes only constant time, while applying the affine transformation to G_1 to form G' is $O(\max(|V_1|, |E_1|))$. The execution time for each hypothesized pair of region triples is dominated by the complexity of establishing the bipartite matching between G_2 and G' , which is $O(\ell n^2 \sqrt{n \log \log n})$, for SMG's with n vertices and ℓ scales. In the worst case, i.e., when both saliency map graphs have only one level, there are $O(n^6)$ pairs of triples. However, in practice, the vertices of an SMG are more uniformly distributed among the levels of the graph, greatly reducing the number of possible correspondences of base triples. For a discussion of how the complexity of the bipartite matching step can be reduced, see [27].

5 Matching Two Shock Trees

5.1 Problem Formulation

Given two shock graphs, one representing an object in the scene (V_2) and one representing a database object (V_1), we seek a method for computing their similarity. Unfortunately, due to occlusion and clutter, the shock graph representing the scene object may, in fact, be embedded in a larger shock graph representing the entire scene. Thus we have a *largest subgraph isomorphism* problem, stated as follows: Given two graphs $G = (V_1, E_1)$ and $H = (V_2, E_2)$, find the maximum integer k , such that there exists two subsets of cardinality k , $E'_1 \subseteq E_1$ and $E'_2 \subseteq E_2$, and the induced subgraphs (not necessarily connected) $G' = (V_1, E'_1)$ and $H' = (V_2, E'_2)$ are isomorphic [12]. Further, since our shock graphs are labeled graphs, consistency between node labels must be enforced in the isomorphism.

The largest subgraph isomorphism problem, can be formulated as a $\{0, 1\}$ integer optimization problem. The optimal solution is a $\{0, 1\}$ bijective mapping

matrix M , which defines the correspondence between the vertices of the two graphs G and H , and which minimizes an appropriately defined distance measure between corresponding edge and/or node labels in the two graphs.

We seek the matrix M , the global optimizer of the following [16,7]:

$$\begin{aligned}
 \min & -\frac{1}{2} \sum_{u \in V_1} \sum_{v \in V_2} M(u, v) \|u, v\| \\
 \text{s.t.} & \sum_{u' \in V_2} M(u, u') \leq 1, \quad \forall u \in V_1 \\
 & \sum_{v \in V_1} M(v, v') \leq 1, \quad \forall v' \in V_2 \\
 & M(x, y) \in \{0, 1\}, \quad \forall x \in V_1, y \in V_2
 \end{aligned} \tag{4}$$

where $\|\cdot\|$ is a measure of the similarity between the labels of corresponding nodes in the two shock graphs (see Sect. 5.2).

The above minimization problem is known to be NP-hard for general graphs [12], however, polynomial time algorithms exist for the special case of finite rooted trees with no vertex labels. Matula and Edmonds [9] describe one such technique, involving the solution of $2n_1n_2$ network flow problems, where n_1 and n_2 represent the number of vertices in the two graphs. The complexity was further reduced by Reyner [24] to $O(n_1^{1.5}n_2)$ (assuming $n_1 \geq n_2$), through a reduction to the bipartite matching algorithm of Hopcraft and Karp [14]. Since we can transform any shock graph into a unique rooted shock tree [30,31], we can pursue a polynomial time solution to our problem. However, as mentioned in Sect. 1, the introduction of noise (spurious addition/deletion of nodes) and/or occlusion may prevent the existence of large isomorphic subtrees. We therefore need a matching algorithm that can find isomorphic subtrees under these conditions. To accomplish this, we have developed a topological representation for trees that is invariant to minor perturbations in structure.

5.2 The Distance between Two Vertices

The eigenvalue characterization introduced in the previous section applies to the problem of determining the topological similarity between two shock trees. This, roughly speaking, defines an equivalence class of objects having the same structure but whose parts may have different qualitative or quantitative shape. For example, a broad range of 4-legged animals will have topologically similar shock trees.

This geometry is encoded by information contained in each vertex of the shock tree. Recall from Sect. 2.2 that $\hat{1}$'s and $\hat{3}$'s represent curve segments of shocks. We choose not to explicitly assign label types 2 and 4, because each may be viewed as a limit case when the number of shocks in a $\hat{3}$, in the appropriate context, approaches 1 (see Sect. 2.2). Each shock in a segment is further labeled by its position, its time of formation (radius of the skeleton), and its direction of flow (or orientation in the case of $\hat{3}$'s), all obtained from the shock detection algorithm [29]. In order to measure the similarity between two vertices u

and v , we interpolate a low dimensional curve through their respective shock trajectories, and assign a cost $C(u, v)$ to an affine transformation that aligns one interpolated curve with the other. Intuitively, a low cost is assigned if the underlying structures are scaled or rotated versions of one another (details can be found in [30,31]).

5.3 Algorithm for Matching Two Shock Trees

As stated in Sect. 1, large isomorphic subtrees may not exist between an image shock tree and a model shock tree, due to noise and/or occlusion. A weaker formulation of the problem would be to find the maximum cardinality, minimum weight matching in a bipartite graph spanning the nodes between two shock trees, with edge weights some function of topological distance and geometrical distance. Although the resulting optimization formulation is more general, allowing nodes in one tree to match any nodes in another tree (thereby allowing nodes to match over “noise” nodes), the formulation is weaker since it doesn’t enforce hierarchical ordering among nodes. Preserving such ordering is essential, for it makes little sense for a node ordering in one tree to match a reverse ordering in another tree. Unfortunately, we are not aware of a polynomial-time algorithm for solving the bipartite matching problem subject to hierarchical constraints. To achieve a polynomial time approximation, we will embed a bipartite matching procedure into a recursive greedy algorithm that will look for maximally similar subtrees.

Our recursive algorithm for matching the rooted subtrees G and H corresponding to two shock graphs is inspired by the algorithm proposed by Reyner [24]. The algorithm recursively finds matches between vertices, starting at the root of the shock tree, and proceeds down through the subtrees in a depth-first fashion. The notion of a match between vertices incorporates two key terms: the first is a measure of the topological similarity of the subtrees rooted at the vertices (see Sect. 3.1), while the second is a measure of the similarity between the shock geometry encoded at each node (see Sect. 5.2). Unlike a traditional depth-first search which backtracks to the next statically-determined branch, our algorithm effectively recomputes the branches at each node, always choosing the next branch to descend in a best-first manner. One very powerful feature of the algorithm is its ability to match two trees in the presence of noise (random insertions and deletions of nodes in the subtrees).

Before stating our algorithm, some definitions are in order. Let $G = (V_1, E_1)$ and $H = (V_2, E_2)$ be the two shock graphs to be matched, with $|V_1| = n_1$ and $|V_2| = n_2$. Define d to be the maximum degree of any vertex in G and H , i.e., $d = \max(\delta(G), \delta(H))$. For each vertex v , we define $\chi(v) \in R^{d-1}$ as the unique eigen-decomposition vector introduced in Sect. 3.1.⁵ Furthermore, for any pair

⁵ Note that if the maximum degree of a node is d , then excluding the edge from the node’s parent, the maximum number of children is $d - 1$. Also note that if $\delta(v) < d$, then then the last $d - \delta(v)$ entries of χ are set to zero to ensure that all χ vectors have the same dimension.

of vertices u and v , let $C(u, v)$ denote the shock distance between u and v , as defined in Sect. 5.2. Finally, let $\Phi(G, H)$ (initially empty) be the set of final node correspondences between G and H representing the solution to our matching problem.

The algorithm begins by forming a $n_1 \times n_2$ matrix $\Pi(G, H)$ whose (u, v) -th entry has the value $C(u, v) \|\chi(u) - \chi(v)\|_2$, assuming that u and v are compatible in terms of their shock order, and has the value ∞ otherwise.⁶ Next, we form a bipartite edge weighted graph $\mathcal{G}(V_1, V_2, E_{\mathcal{G}})$ with edge weights from the matrix $\Pi(G, H)$.⁷ Using the scaling algorithm of Goemans, Gabow, and Williamson [11], we then find the maximum cardinality, minimum weight matching in \mathcal{G} . This results in a list of node correspondences between G and H , called \mathcal{M}_1 , that can be ranked in decreasing order of similarity.

From \mathcal{M}_1 , we choose (u_1, v_1) as the pair that has the minimum weight among all the pairs in \mathcal{M}_1 , i.e., the first pair in \mathcal{M}_1 . (u_1, v_1) is removed from the list and added to the solution set $\Phi(G, H)$, and the remainder of the list is *discarded*. For the subtrees G_{u_1} and H_{v_1} of G and H , rooted at nodes u_1 and v_1 , respectively, we form the matrix $\Pi(G_{u_1}, H_{v_1})$ using the same procedure described above. Once the matrix is formed, we find the matching \mathcal{M}_2 in the bipartite graph defined by weight matrix $\Pi(G_{u_1}, H_{v_1})$, yielding another ordered list of node correspondences. The procedure is recursively applied to (u_2, v_2) , the edge with minimum weight in \mathcal{M}_2 , with the remainder of the list discarded.

This recursive process eventually reaches the leaves of the subtrees, forming a list of ordered correspondence lists (or matchings) $\{\mathcal{M}_1, \dots, \mathcal{M}_k\}$. In backtracking step i , we remove any subtrees from the graphs G_i and H_i whose roots participate in a matching pair in $\Phi(G, H)$ (we enforce a one-to-one correspondence of nodes in the solution set). Then, in a depth-first manner, we first recompute \mathcal{M}_i on the subtrees rooted at u_i and v_i (with solution set nodes removed). As before, we choose the minimum weight matching pair, and recursively descend. Unlike in a traditional depth-first search, we dynamically recompute the branches at each node in the search tree. Processing at a particular node will terminate when either subtree loses all of its nodes to the solution set. We can now state the algorithm more precisely:

```

procedure isomorphism( $G, H$ )
   $\Phi(G, H) \leftarrow \emptyset$ 
   $d \leftarrow \max(\delta(G), \delta(H))$ 
  for  $u \in V_G$  compute  $\chi(u) \in R^{d-1}$  (Section 3.1)
  for  $v \in V_H$  compute  $\chi(v) \in R^{d-1}$  (Section 3.1)
  call match(root( $G$ ), root( $H$ ))
  return(cost( $\Phi(G, H)$ ))
end

```

⁶ If either $C(u, v)$ or $\|\chi(u) - \chi(v)\|_2$ is zero, the (u, v) -th entry is the other term.

⁷ $G(A, B, E)$ is a weighted bipartite graph with weight matrix $W = [w_{ij}]$ of size $|A| \times |B|$ if, for all edges of the form $(i, j) \in E$, $i \in A$, $j \in B$, and (i, j) has an associated weight $= w_{i,j}$.


```

procedure match( $u, v$ )
  do
    {
      let  $G_u \leftarrow$  rooted subtree of  $G$  at  $u$ 
      let  $H_v \leftarrow$  rooted subtree of  $H$  at  $v$ 
      compute  $|V_{G_u}| \times |V_{H_v}|$ 
      weight matrix  $\Pi(G_u, H_v)$ 
       $\mathcal{M} \leftarrow$  max cardinality, minimum weight
        bipartite matching in  $\mathcal{G}(V_{G_u}, V_{H_v})$ 
        with weights from  $\Pi(G_u, H_v)$  (see [11])
       $(u', v') \leftarrow$  minimum weight pair in  $\mathcal{M}$ 
       $\Phi(G, H) \leftarrow \Phi(G, H) \cup \{(u', v')\}$ 
      call match( $u', v'$ )
       $G_u \leftarrow G_u - \{x | x \in V_{G_u} \text{ and } (x, w) \in \Phi(G, H)\}$ 
       $H_v \leftarrow H_v - \{y | y \in V_{H_v} \text{ and } (w, y) \in \Phi(G, H)\}$ 
    }
  while ( $G_u \neq \emptyset$  and  $H_v \neq \emptyset$ )

```

In terms of algorithmic complexity, observe that during the depth-first construction of the matching chains, each vertex in G or H will be matched at most once in the forward procedure. Once a vertex is mapped, it will never participate in another mapping again. The total time complexity of constructing the matching chains is therefore bounded by $O(n^2\sqrt{n\log\log n})$, for $n = \max(n_1, n_2)$ [11]. Moreover, the construction of the $\chi(v)$ vectors will take $O(n\sqrt{n}L)$ time, implying that the overall complexity of the algorithm is $\max(O(n^2\sqrt{n\log\log n}), O(n^2\sqrt{n}L))$.

The approximation has to do with the use of a scaling parameter to find the maximum cardinality, minimum weight matching [11]; this parameter determines a tradeoff between accuracy and the number of iterations until convergence. The matching matrix M in (4) can be constructed using the mapping set $\Phi(G, H)$. The algorithm is particularly well-suited to the task of matching two shock trees since it can find the best correspondence in the presence of occlusion and/or noise in the tree.

6 Experiments

6.1 Indexing Experiments

We test our indexing algorithm on a database of 60 object silhouettes, some representative examples of which are shown in Fig. 7. In the first experiment, we select 20 shapes from the database, compute their shock trees, compute the topological signature vectors for each of their nodes, and populate the resulting vectors in a model database. Each element, in turn, will be removed from the database and used as a query tree for the remaining database of 19 model trees. For each of the 20 trials, the 19 object candidates will be ranked in decreasing order of accumulator contents. To evaluate the quality of the indexing results,

we will compute the distance between the query tree and each of the candidates, using the matcher developed in [31], and note which model tree is the closest match. If indexing is to be effective, the closest matching model should be among the best (highest-weight) candidates returned by the indexing strategy.

In the second and third experiments, we apply the same procedure to databases of size 40 and 60 model trees, respectively, in order to evaluate the scaling properties of our indexing algorithm. Thus, in the second experiment, we have 40 indexing trials, while in the third experiment, we have 60 indexing trials. Finding the position of the closest model shape among the sorted candidates for any given query shape requires that we first compute the 60×60 distance matrix.

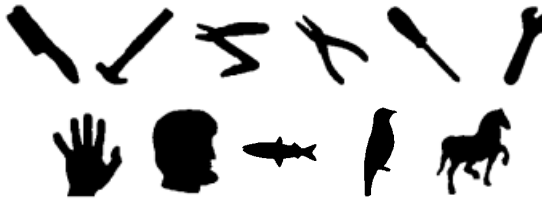


Fig. 7. Samples from a Database of 60 Object Silhouettes

The results of the first experiment are shown in Fig. 8(a), where the horizontal axis indicates the rank of the target object (or closest matching object) in the sorted candidates, and the vertical axis represents the number of times that rank is achieved. For this experiment, the average rank is 1.6, which implies that on average, 8.4% of the sorted candidates need to be verified before the closest matching model is found. The results of the second and third experiments are shown in Fig. 8(b) and (c), respectively. The results are very encouraging and show that as database size increases, the indexing algorithm continues to prune over 90% of the database (avg. rank of 7.9% in expt. 2, 8.8% in expt. 3).

In a final experiment, we generate some occluded scenes from shapes in our database. In Table 1, we show three examples of occluded query images (left column) and the top ten (sorted left to right) model candidates from a database of 40 model shapes. Twice, the rank of the target is 4th, while once it is 3rd, indicating that for these three examples, at most 10% of the model indexing candidates need to be verified. We are currently conducting a more comprehensive set of occlusion experiments.

It should be noted that the indexing mechanism reflects primarily the topological structure of the query. Thus, in row 1 of Table 1, for example, the topological structure of the query (brush occluding the hammer) is more similar to the pliers-like objects (two of the top three candidates) than to the hammer itself. Topological similarity of shock trees is a necessary but not sufficient condition

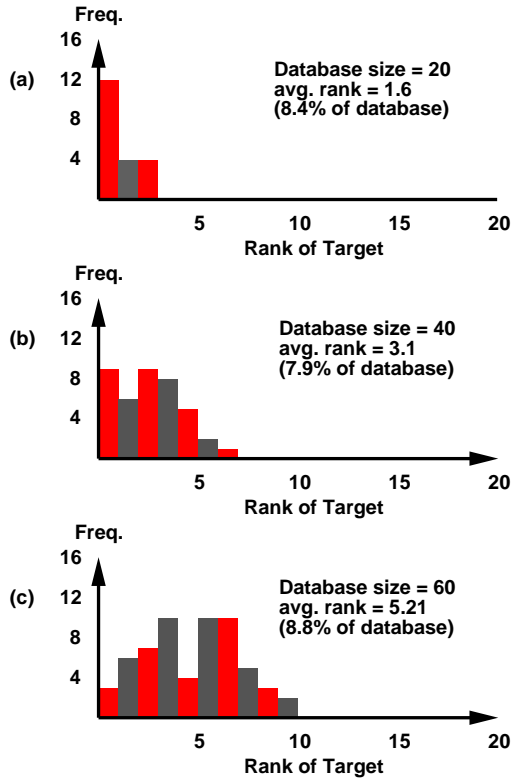


Fig. 8. Indexing Results for a Database of 60 Images. In each case, the horizontal axis indicates the rank of the target object (or closest matching object) in the sorted candidates, and the vertical axis represents the number of times that rank is achieved. (See text for discussion.)

for shape similarity, as it ignores the geometries of the object's parts (nodes in its shock tree). Therefore, the fact that objects with different shape can rank high in the candidate list is not surprising.

6.2 Matching of Saliency Maps

To evaluate our representation and matching framework, we apply it to a database of model object views generated by Murase and Nayar at Columbia University. Views of each of the 20 objects are taken from a fixed elevation every 5 degrees (72 views per object) for a total of 1440 model views. The top row of images in Fig. 9 shows three adjacent model views for one of the objects (piggy bank) plus one model view for each of two other objects (bulb socket and cup). The second row shows the computed saliency maps for each of the five images, while the third row shows the corresponding saliency map graphs. The time to compute the saliency map averaged 156 seconds/image for the five images on

Table 1. Indexing using an occluded query shape. For each row, the query is shown to the left, while the top ten candidate models (from a database of 40 models) are shown on the right, in decreasing order of weight.

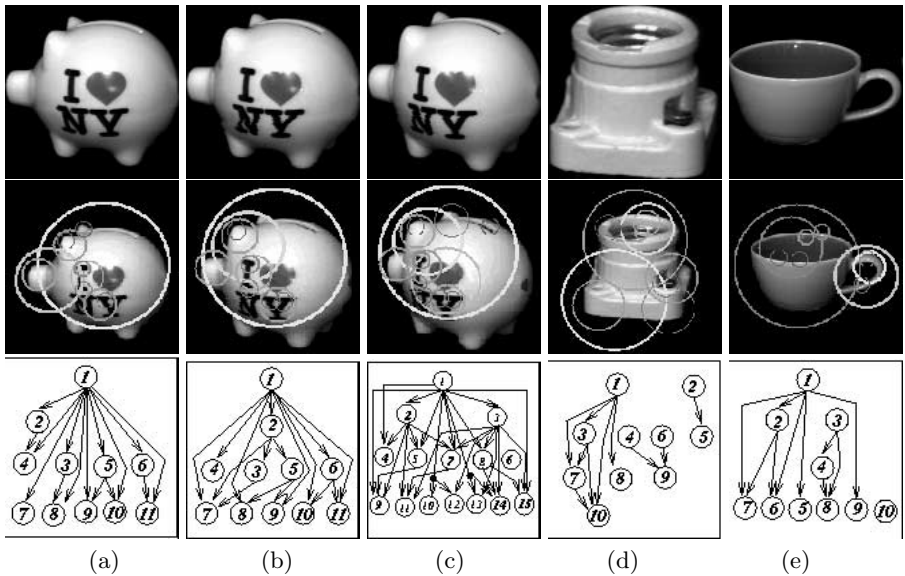
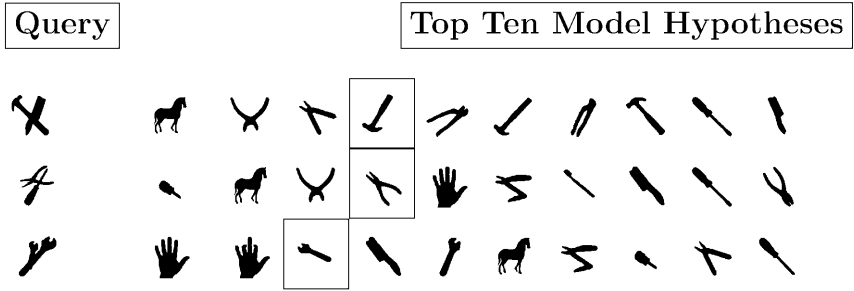


Fig. 9. A sample of views from the database: top row represents original images, second row represents saliency maps, while third row represents saliency map graphs.

a Sun Sparc 20, but can be reduced to real-time on a system with hardware support for convolution, e.g., a Datacube MV200. The average time to compute the distance between two SMG's is 50 ms using SMGBM, and 1.1 second using SMGAT (an average of 15 nodes per SMG).

To illustrate the matching of an unoccluded image to the database, we compare the middle piggy bank image (Fig. 9(b)) to the remaining images in the database. Table 2 shows the distance of the test image to the other images in Fig. 9; the two other piggy bank images (Figs 9 (a) and (c)) were the closest

matching views in the entire database. Table 2 also illustrates the difference between the two matching algorithms. SMGBM is a weaker matching algorithm, searching for a topological match between two SMG's. SMGAT, on the other hand, is more restrictive, searching for a geometrical match between the two SMG's. For similar views, the two algorithms are comparable; however, as two views diverge in appearance, their similarity as computed by SMGAT diverges more rapidly than their SMGBM similarity. In a third experiment, we compare

Table 2. Distance of Fig. 9(b) to other images in Fig. 9

Algorithm	9(a)	9(c)	9(d)	9(e)
SMGBM	9.57	10.06	14.58	23.25
SMGAT	8.91	12.27	46.30	43.83

every image to every other image in the database, resulting in over 1 million trials. There are three possible outcomes: 1) the image removed from the database is closest to one of its neighboring views of the correct object; 2) the image removed from the database is closest to a view belonging to the correct object but not a neighboring view; and 3) the image removed from the database is closest to a view belonging to a different object. The results are shown in Table 3. As we would expect, the SMGAT algorithm, due to its stronger matching criterion, outperforms the SMGBM algorithm. If we include as a correct match any image belonging to the same object, both algorithms (SMGBM and SMGAT) perform extremely well, yielding success rates of 97.4% and 99.5%, respectively. To illus-

Table 3. An exhaustive test of the two matching algorithms. For each image in the database, the image is removed from the database and compared, using both algorithms, to every remaining image in the database. The closest matching image can be either one of its two neighboring views, a different view belonging to the correct object, or a view belonging to a different object.

Algorithm	% Hit	% Miss right object	% Miss wrong object
SMGBM	89.0	8.4	2.6
SMGAT	96.6	2.9	0.5

trate the matching of an occluded image to the database, we compare an image containing the piggy bank occluded by the bulb socket, as shown in Fig. 10. Table 4 shows the distance of the test image to the other images in Fig. 9. The closest matching view is the middle view of the piggy bank which is, in fact, the view embedded in the occluded scene. In a labeling task, the subgraph matching the closest model view would be removed from the graph and the procedure applied to the remaining subgraph. After removing the matching subgraph, we

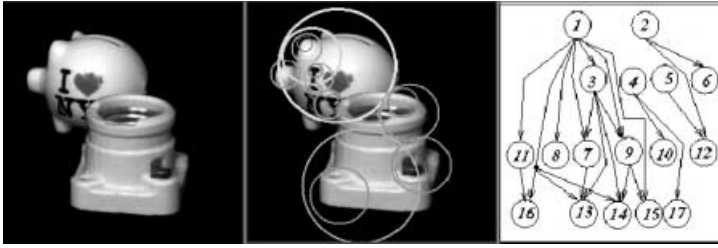


Fig. 10. Occluded Object Matching: (a) original image; (b) saliency map; and (c) saliency map graph

match the remaining scene subgraph to the entire database, as shown in Table 5. In this case, the closest view is the correct view (Figure 9(d)) of the socket.

Table 4. Distance of Fig. 10(a) to other images in Fig. 9. The correct piggy bank view (Fig. 9(b)) is the closest matching view.

Algorithm	9(a)	9(b)	9(c)	9(d)	9(e)
SMGBM	9.56	3.47	8.39	12.26	14.72
SMGAT	24.77	9.29	21.19	30.17	33.61

Table 5. Distance of Fig. 10(a) (after removing from its SMG the subgraph corresponding to the matched piggy back image) to other images in Fig. 9.

























Algorithm	9(a)	9(b)	9(c)	9(d)	9(e)
SMGBM	12.42	14.71	14.24	4.53	9.83
SMGAT	18.91	20.85	17.08	7.19	15.44

6.3 Matching of Shock Trees

To evaluate our matcher’s ability to compare objects based on their prototypical or coarse shape, we begin with a database of 24 objects belonging to 9 classes. To select a given class prototype, we select that object whose total distance to the other members of its class is minimum.⁸ We then compute the similarity between each remaining object in the database and each of the class prototypes, with the results shown in Table 6. For each row in the table, a box has been placed around the most similar shape. We note that for the 15 test shapes drawn

⁸ For each of the three classes having only two members, the class prototype was chosen at random.

Table 6. Similarity between database shapes and class prototypes. In each row, a box is drawn around the most similar shape (see the text for a discussion).

Instance	Distance to Class Prototype								
									
	0.02	2.17	4.48	3.55	2.96	0.21	4.58	14.33	10.01
	2.39	0.10	5.97	15.90	3.98	0.14	26.12	17.28	28.94
	10.89	4.72	2.08	12.24	3.12	2.15	19.73	10.11	12.64
	7.15	6.42	1.19	1.35	5.10	3.38	10.58	11.11	11.11
	4.08	7.72	2.98	1.49	4.26	4.14	26.60	13.54	14.21
	14.77	6.72	5.69	0.36	2.30	5.90	10.58	16.25	19.10
	7.86	8.90	5.94	0.74	1.59	1.10	10.81	10.39	16.08
	2.66	4.23	3.23	6.47	0.62	1.48	11.73	15.38	15.15
	3.18	5.31	1.25	4.64	0.60	1.30	14.18	17.22	9.08
	4.55	0.76	1.32	2.86	1.49	0.11	21.38	15.35	13.04
	6.77	19.46	22.11	13.27	8.21	29.50	0.15	5.12	5.03
	8.73	23.14	31.45	24.41	10.16	31.08	0.18	8.45	7.05
	12.46	19.0	27.40	14.58	24.26	17.10	8.85	7.49	16.93
	13.86	23.07	12.81	11.24	17.48	23.23	6.02	6.92	3.06
	15.73	21.28	14.10	12.46	19.56	19.21	9.53	7.12	5.06

from 9 classes, all but one are most similar to their class prototype, with the class prototype coming in a close second in that case.

Three very powerful features of our system are worth highlighting. First, the method is truly generic: the matching scores impose a partial ordering in each row, which reflects the qualitative similarity between structurally similar shapes. An increase in structural complexity is reflected in a higher cost for the best match, e.g., in the bottom two rows of Fig. 6. Second, the procedure is designed to handle noise or occlusion, manifest as missing or additional vertices in the shock graph. Third, the depth-first search through subtrees is extremely efficient.

7 Selected Related Work

Multi-scale image descriptions have been used by other researchers to locate a particular target object in the image. For example, Rao et al. use correlation

to compare a multi-scale saliency map of the target object with a multi-scale saliency map of the image in order to fixate on the object [23]. Although these approaches are effective in finding a target in the image, they, like any template-based approach, do not scale to large object databases. Their bottom-up descriptions of the image are not only global, offering little means for segmenting an image into objects or parts, but offer little invariance to occlusion, object deformation, and other transformations.

Wiskott *et al.* [33] use Gabor wavelet jets to extract salient image features. Wavelet jets represent an image patch (containing a feature of interest) with a set of wavelets across the frequency spectrum. Each collection of wavelet responses represents a node in a grid-like planar graph covering overlapping regions of the image. Image matching reduces to a form of elastic graph matching, in which the similarity between the corresponding Gabor jets of nodes is maximized. Correspondence is proximity-based, with nodes in one graph searching for (spatially) nearby nodes in another graph. Effective matching therefore requires that the graphs be coarsely aligned in scale and image rotation.

Another related approach is due to Crowley *et al.* [3,2,4]. From a Laplacian pyramid computed on an image, peaks and ridges at each scale are detected as local maxima. The peaks are then linked together to form a tree structure, from which a set of *peaks paths* are extracted, corresponding to the branches of the tree. During matching, correspondence between low-resolution peak paths in the model and the image are used to solve for the pose of the model with respect to the image. Given this initial pose, a greedy matching algorithm descends down the tree, pairing higher-resolution peak paths from the image and the model. Using a log likelihood similarity measure on peak paths, the best corresponding paths through the two trees is found. The similarity of the image and model trees is based on a very weak approximation of the trees' topology and geometry, restricted, in fact, to a single path through the tree.

Graph matching is a very popular topic in the computer vision community. Although space prohibits us from providing a comprehensive review, we will mention some particularly relevant related work. A graduated assignment algorithm has been proposed for subgraph isomorphism, weighted graph matching, and attributed relational graph matching [13]. The method was applied to matching non-hierarchical point features and performs well in the presence of noise and occlusion. Cross and Hancock propose a two step matching algorithm for locating point correspondences and estimating geometric transformation parameters between 2-D images. Point correspondence is achieved via maximum a posteriori graph-matching, while expectation maximization (EM) is used to recover the maximum likelihood transformation parameters. The novel idea of using graph-based models to provide structural constraints on parameter estimation is an important contribution their work. This, combined with the EM algorithm, allows their system to impose an explicit deformational model on the feature points.

The matching of shock trees has been addressed by a number of other groups. In recent work, Pelillo *et al.* [21] introduced a matching algorithm which extends

the detection of maximum cliques in association graphs to hierarchically organized tree structures. They use the concept of connectivity to derive an association graph, and prove that attributed tree matching is equivalent to finding a maximum clique in the association graph. They applied their algorithm to articulated and deformed shapes represented as shock trees. In a related paper, Tirthapura et al. [32] present an alternative use of shock graphs for shape matching. Their approach relies on graph transformations based on the edit distance between two graphs, defined as the “least action” path consisting of a sequence of elementary edit transformations taking one graph to another. The first approach can handle occlusion, but does not accommodate spurious noise in the graphs; the second approach handles spurious noise, but cannot effectively deal with occlusion. Both approaches focus solely on graph (tree) structure, and would have to be modified to include the concept of node similarity.

8 Conclusions

In this paper, we have reviewed three different algorithms for object recognition, each based on solving a bipartite matching formulation of a particular problem. The formulation is both very general and very powerful. We have shown edge weights that encode difference in region saliency, Euclidean distance in the image, and a function of topological and geometric distance. We have also seen different ways in which hierarchical ordering of nodes in a graph/tree can be enforced. In the case of saliency map graph matching, parent/child relationships are used to bias edge weights at lower levels of the matching, while in the case of shock tree matching, a depth-first procedure is used to ensure hierarchical consistency. It should be noted that the method by which we enforce hierarchical ordering in the matching of saliency map graphs is not applicable to the matching of shock graphs (DAGs or trees), since the method assumes that corresponding nodes in the hierarchy are at comparable scales. In a shock graph, a leaf child of the root may be as small in scale as a leaf further down the tree. However, we are exploring the application of our shock tree matching and indexing methods to multi-scale DAG representations.

Finally, we have shown how matching complexity can be managed in a coarse-to-fine framework. In the case of saliency map graph matching, solutions to the bipartite matching problem at a coarser level are used to constrain solutions at a finer level, while in the case of shock tree matching, large corresponding subtree roots (found through a solution to the bipartite matching problem) are used to establish correspondence between their descendents. Furthermore, in the case of shock tree matching, our eigen-characterization of a tree’s topological structure allows us to efficiently compare subtree structures in the presence of noise and occlusion.

References

- [1] Alizadeh, F.: Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM J. Optim.* **5(1)** (1995) 13–51.
- [2] Crowley, J., Parker, A.: A representation for shape based on peaks and ridges in the difference of low-pass transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6(2)** (1984) 156–169.
- [3] Crowley, J. L.: A Multiresolution Representation for Shape. In Rosenfeld, editor. *Multiresolution Image Processing and Analysis* (1984) 169–189. Springer Verlag, Berlin.
- [4] Crowley, J. L., Sanderson, A. C.: Multiple Resolution Representation and Probabilistic Matching of 2-D Gray-Scale Shape. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **9(1)** (1987) 113–121.
- [5] Cvetković, D.: *Graphs and their spectra*. University of Beograd (1971) 354–356.
- [6] Cvetković, D., Rowlinson, P., Simić, S.: *Eigenspaces of Graphs*. Cambridge University Press, Cambridge, United Kingdom (1997).
- [7] Mjolsness, G. G. E., Anandan., P.: Optimization in model matching and perceptual organization. *Neural Computation* **1** (1989) 218–229.
- [8] Edmonds. E.: Paths, trees, and flowers. *Canadian Journal of Mathematics* **17** (1965) 449–467.
- [9] Edmonds, J., Matula. D.: An algorithm for subtree identification. *SIAM Rev.* **10** (1968) 273–274.
- [10] Flynn, P., Jain, A.: 3D object recognition using invariant feature indexing of interpretation tables. *CVGIP: Image Understanding* **55(2)** (1992) 119–129.
- [11] Gabow, H., Goemans, M., Williamson, D.: An efficient approximate algorithm for survivable network design problems. *Proc. of the Third MPS Conference on Integer Programming and Combinatorial Optimization* (1993) 57–74.
- [12] Garey, M., Johnson, D.: *Computer and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco (1979).
- [13] Gold, S., Rangarajan, A.: A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **18(4)** (1996) 377–388.
- [14] Hopcroft, J., Karp, R.: An $n^{\frac{5}{2}}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.* **2** (1973) 225–231.
- [15] Kimia, B. B., Tannenbaum, A., Zucker, S. W.: Shape, shocks, and deformations I: The components of two-dimensional shape and the reaction-diffusion space. *International Journal of Computer Vision* **15** (1995) 189–224.
- [16] Kobler, J.: *The graph isomorphism problem: its structural complexity*. Birkhauser, Boston (1993).
- [17] Lamdan, Y., Schwartz, J., Wolfson., H.: Affine invariant model-based object recognition. *IEEE Transactions on Robotics and Automation* **6(5)** (1990) 578–589.
- [18] Neumaier, A.: Second Largest Eigenvalue of a Tree. *Linear Algebra and its Applications* **46** (1982) 9–25.
- [19] Nosal, E.: *Eigenvalues of Graphs*. University of Calgary (1970).
- [20] Overton, M. L., Womersley, R. S.: Optimality conditions and duality theory for minimizing sums of the largest eigenvalues of symmetric matrices. *Math. Programming* **62(2)** (1993) 321–357.
- [21] Pelillo, M., Siddiqi, K., Zucker, S.W.: Matching hierarchical structures using association graphs. *Fifth European Conference on Computer Vision* **2** (1998) 3–16.

- [22] Preparata, F., Shamos, M.: *Computational Geometry*. Springer-Verlag. New York, NY (1985).
- [23] Rao, R. P. N, Zelinsky, G. J., Hayhoe, M. M., Ballard, D. H.: Modeling Saccadic Targeting in Visual Search. In Touretzky, Mozer, and Hasselmo, editors. *Advances in Neural Information Processing Systems* **8** 830–836. MIT Press. Cambridge, MA (1996).
- [24] Reyner, S. W.: An analysis of a good algorithm for the subtree problem. *SIAM J. Comput.* **6** (1997) 730–732.
- [25] Shokoufandeh, A., Marsic, I., Dickinson, S.: Saliency regions as a basis for object recognition. In *Third International Workshop on Visual Form*. Capri, Italy (1997).
- [26] Shokoufandeh, A., Marsic, I., Dickinson, S.: View-based object matching. In *Proceedings, IEEE International Conference on Computer Vision*. Bombay (1998) 588–595.
- [27] Shokoufandeh, A., Marsic, I., Dickinson, S.: View-based object recognition using saliency maps. Technical Report DCS-TR-339, Department of Computer Science, Rutgers University. New Brunswick, NJ 08903 (1998).
- [28] Shokoufandeh, A., Marsic, I., Dickinson, S.: View-based object recognition using saliency maps. *Image and Vision Computing* **27** (1999) 445–460.
- [29] Siddiqi, K., Kimia, B. B.: A shock grammar for recognition. Technical Report LEMS 143. LEMS, Brown University (1995).
- [30] Siddiqi, K., Shokoufandeh, A., Dickinson, S., Zucker, S. W.: Shock graphs and shape matching. In *Proceedings, IEEE International Conference on Computer Vision*. Bombay (1998) pages 222–229.
- [31] Siddiqi, K., Shokoufandeh, A., Dickinson, S., Zucker, S.W.: Shock graphs and shape matching. *International Journal of Computer Vision* **30** (1999) 1–22.
- [32] Tirthapura, S., Sharvit, D., Klein, P., Kimia, B.B.: Indexing Based on Edit-Distance Matching of Shape Graphs. *SPIE Proceedings on Multimedia Storage and Archiving Systems III* (1998) 25–36.
- [33] Wiskott, L., Fellous, J. M., Krüger, N., von der Malsburg, C.: Face Recognition by elastic bunch graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19(7)** (1997) 775–779.