A Unified Framework for Indexing and Matching Hierarchical Shape Structures

Ali Shokoufandeh[1] and Sven Dickinson[2]

[1] Department of Mathematics and Computer Science
Drexel University
3141 Chestnut Street
Philadelphia, PA USA 19104-2875
[2] Department of Computer Science
University of Toronto
6 King's College Rd.
Toronto, Ontario Canada M5S 3G4

Abstract. Hierarchical image structures are abundant in computer vision, and have been used to encode part structure, scale spaces, and a variety of multiresolution features. In this paper, we describe a unified framework for both indexing and matching such structures. First, we describe an indexing mechanism that maps the topological structure of a directed acyclic graph (DAG) into a low-dimensional vector space. Based on a novel eigenvalue characterization of a DAG, this topological signature allows us to efficiently retrieve a small set of candidates from a database of models. To accommodate occlusion and local deformation, local evidence is accumulated in each of the DAG's topological subspaces. Given a small set of candidate models, we will next describe a matching algorithm that exploits this same topological signature to compute, in the presence of noise and occlusion, the largest isomorphic subgraph between the image structure and the candidate model structure which, in turn, yields a measure of similarity which can be used to rank the candidates. We demonstrate the approach with a series of indexing and matching experiments in the domains of 2-D and (view-based) 3-D generic object recognition.

1 Introduction

The indexing and matching of hierarchical (e.g., multiscale or multilevel) image features is a common problem in object recognition. Such structures are often represented as rooted trees or directed acyclic graphs (DAGs), where nodes represent image feature abstractions and arcs represent spatial relations, mappings across resolution levels, component parts, etc [40, 14]. 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 DAGs or rooted trees. Yet, at some level of abstraction, the two structures (or two of their substructures) may be quite similar.

The matching procedure is expensive and must be used sparingly. For large databases of object models, it is simply unacceptable to perform a linear search of the database. Therefore, an indexing mechanism is essential for selecting a small set of candidate models to which the matching procedure is applied. When working with hierarchical image structures, in the form of graphs, indexing is a challenging task, and can be formulated as the fast selection of a set of candidate models that share a subgraph with the query. But how do we test a given candidate without resorting to subgraph isomorphism? If there were a small number of subgraphs shared among many models, representing a vocabulary of *object parts*, one could conceive of a two-stage indexing process, in which image structures were matched to the part vocabulary, with parts "voting" for candidate models [6]. However, we're still faced with the complexity of subgraph isomorphism, albeit for a smaller database (vocabulary of parts).

In this paper, we present a unified solution to the problems of indexing and matching hierarchical structures. Drawing on techniques from the domain of eigenspaces of graphs, we present a technique that maps any rooted hierarchical structure, i.e., DAG or rooted tree, to a vector in a low-dimensional space. The mapping not only reduces the dimensionality of the representation, but does so while retaining important information about the branching structure, node distribution, and overall structure of the graph – information that is critical in distinguishing DAGs or rooted trees. Moreover, the technique accommodates both noise and occlusion, meeting the needs of an indexing structure for vision applications. Armed with a low-dimensional, robust vector representation of an input structure, indexing can be reduced to a nearest-neighbor search in a database of points, each representing the structure of a model (or submodel).

Once a candidate is retrieved by the indexing mechanism, we exploit this *same* eigen-characterization of hierarchical structure to compute a node-to-node correspondence between the input and model candidate hierarchical structures. We therefore unify our approaches to indexing and matching through a novel representation of hierarchical structure, leading to an efficient and effective framework for the recognition of hierarchical structures from large databases. In this paper, we will review our representation, described in [33,35], including a new analysis on its stability. We then describe the unifying role of our representation in the indexing and matching of hierarchical structures. Finally, we demonstrate the approach on two separate object recognition domains.

2 Related Work

Eigenspace approaches to shape description and indexing are numerous. Due to space constraints, we cite only a few examples. Turk and Pentland's eigenface approach [38] represented an image as a linear combination of a small number of basis vectors (images) computed from a large database of images. Nayar and Murase extended this work to general 3-D objects where a dense set of views was acquired for each object [17]. Other eigenspace methods have been applied to higher-level features, offering more potential for generic shape description and matching. For example, Sclaroff and Pentland compute the eigenmodes of vibration of a 2-D region [26], while Shapiro and Brady looked at how the modes of vibration of a set of 2-D points could be used to solve the point correspondence problem under translation, rotation, scale, and small skew [28].

In an attempt to index into a database of graphs, Sossa and Horaud use a small subset of the coefficients of the d_2-polynomial corresponding to the Laplacian matrix associated with a graph [36], while a spectral graph decomposition was reported by Sengupta and Boyer for the partitioning of a database of 3-D models, where nodes in a graph represent 3-D surface patches [27]. Sarkar [25] and Shi and Malik [31] have formulated the perceptual grouping and region segmentation problems, respectively, as graph partitioning problems and have used a generalized eigensystem approach to provide an efficient approximation. We note that in contrast to many of the above approaches to indexing, the representation that we present in this paper is independent of the contents of the model database and uses a uniform basis to represent all objects.

There have been many approaches to object recognition based on graph matching. An incomplete list of examples include Sanfeliu and Fu [24], Shapiro and Haralick [30,29], Wong et al. [41,22], Boyer and Kak [2] (for stereo matching), Kim and Kak [11], Messmer and Bunke [16], Christmas et al. [4], Eshera and Fu [7], Pellilo et al. [20], Gold and Rangarajan [10], Zhu and Yuille [42], and Cross and Hancock [5]. Although many of these approaches handle both noise and occlusion, none unify both indexing and matching through a single spectral mechanism.

3 Indexing Hierarchical Structures

We make the assumption that if a DAG[1] has rich structure in terms of depth and/or branching factor, its topology alone may serve as a discriminating index into a database of model structures. Although false positives (e.g., model DAGs that have the same structure, but whose node labels are different) may arise, they may be few in number and can be pruned during verification. As stated in Section 1, we seek a reduced representation for a DAG that will support efficient indexing and matching. An effective topological encoding of a DAG's structure should: **1)** map a DAG's topology to a point in some low-dimensional space; **2)** capture local topology to support matching/indexing in the presence of occlusion; **3)** be invariant to re-orderings of the DAG's branches, i.e., re-orderings which do not affect the parent-child relationships in the DAG; **4)** be

[1] Although a hierarchical structure can take the form of a DAG or rooted tree, we will henceforth limit our discussion to DAGs, since a rooted tree is a special case of a DAG.

as unique as possible, i.e., different DAGs should have different encodings; **5)** be stable, i.e., small perturbations of a DAG's topology should result in small perturbations of the index; and **6)** should be efficiently computed.

3.1 An Eigen-decomposition of Structure

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 symmetric $\{0, 1, -1\}$ adjacency matrix, with 1's (-1's) indicating a forward (backward) edge between adjacent nodes in the graph (and 0's on the diagonal). The eigenvalues of a graph's adjacency matrix encode important structural properties of the graph. Furthermore, the eigenvalues of a symmetric matrix A are invariant to any orthonormal transformation of the form P^tAP. Since a permutation matrix is orthonormal, the eigenvalues of a graph are invariant to any consistent re-ordering of the graph's branches. However, before we can exploit a graph's eigenvalues for indexing purposes, we must establish their stability under minor topological perturbation, due to noise, occlusion, or deformation.

We will begin by showing that any structural change to a graph can be modeled as a two-step transformation of its original adjacency matrix. The first step transforms the graph's original adjacency matrix to a new matrix having the same spectral properties as the original matrix. The second step adds a noise matrix to this new matrix, representing the structural changes due to noise and/or occlusion. These changes take the form of the addition/deletion of nodes/arcs to/from the original graph. We will then draw on an important result that relates the distortion of the eigenvalues of the matrix resulting from the first step to the magnitude of the noise added in the second step. Since the eigenvalues of the original matrix are the same as those of the transformed matrix (first step), the noise-dependent eigenvalue bounds therefore apply to the original matrix. The result will establish the insensitivity of a graph's spectral properties to minor topological changes.

Let's begin with some definitions. Let $A_G \in \{0, 1, -1\}^{m \times m}$ denote the adjacency matrix of the graph G on m vertices, and assume H is an n-vertex graph obtained by adding $n - m$ new vertices and a set of edges to the graph G. Let $\Psi : \{0, 1, -1\}^{m \times m} \to \{0, 1, -1\}^{n \times n}$, be a *lifting operator* which transforms a subspace of $R^{m \times m}$ to a subspace of $R^{n \times n}$, with $n \geq m$. We will call this operator *spectrum preserving* if the eigenvalues of any matrix $\mathcal{A} \in \{0, 1, -1\}^{m \times m}$ and its image with respect to the operator ($\Psi(\mathcal{A})$) are the same up to a degeneracy, i.e., the only difference between the spectra of $\mathcal{A}$ and $\Psi(\mathcal{A})$ is the number of zero eigenvalues ($\Psi(\mathcal{A})$ has $n - m$ more zero eigenvalues then $\mathcal{A}$).

As stated above, our goal is to show that any structural change in graph G can be represented in terms of a spectrum preserving operator and a noise matrix. Specifically, if A_H denotes the adjacency matrix of the graph H, then there exists a spectrum preserving operator $\Psi()$ and a noise matrix $E_H \in \{0, 1, -1\}^{n \times n}$ such that:

$$A_H = \Psi(A_G) + E_H \tag{1}$$

We will define $\Psi()$ as a lifting operator consisting of two steps. First, we will add $n-m$ zero rows and columns to the matrix A_G, and denote the resulting matrix by A'_G. Next, A'_G will be pre- and post-multiplied by a permutation matrix P and its transpose P^t, respectively, aligning the rows and columns corresponding to the same vertices in A_H and $\Psi(A_G)$. Since the only difference between the eigenvalues of A'_G and A_G is the number of zero eigenvalues, and PA'_GP^t has the same set of eigenvalues as the matrix A'_G, $\Psi()$ is a spectrum preserving operator. As a result, the noise matrix E_H can be represented as $A_H - \Psi(A_G) \in \{0, 1, -1\}^{n\times n}$.

Armed with a spectrum-preserving lifting operator and a noise matrix, we can now proceed to quantify the impact of the noise on the original graph's eigenvalues. Specifically, let λ_k for $k \in \{1, ..., n\}$ denote the k^{th} largest eigenvalue of the matrix A. A seminal result of Wilkinson [39] (see also Stewart and Sun [37]) states that:

Theorem 1. *If A and $A+E$ are $n\times n$ symmetric matrices, then:*

$$\lambda_k(A) + \lambda_k(E) \leq \lambda_k(A+E) \leq \lambda_k(A) + \lambda_1(E), \quad \text{for all } k \in \{1, ..., n\} \qquad (2)$$

For H and G, we know that $\lambda_1(E_H) = ||E_H||_2$. Therefore, using the above theorem, for all $k \in \{1, ..., n\}$:

$$\begin{aligned} |\lambda_k(A_H) - \lambda_k(\Psi(A_G))| &= |\lambda_k(\Psi(A_G) + E_H) - \lambda_k(\Psi(A_G))| \\ &\leq \max\{|\lambda_1(E_H)|, |\lambda_n(E_H)|\} \qquad (3) \\ &= ||E_H||_2. \end{aligned}$$

The above chain of inequalities gives a precise bound on the distortion of the eigenvalues of $\Psi(A_G)$ in terms of the largest eigenvalue of the noise matrix E_H. Since $\Psi()$ is a spectrum preserving operator, the eigenvalues of A_G follow the same bound in their distortions.

The above result has several important consequences for our application of a graph's eigenvalues to graph indexing. Namely, if the perturbation E_H is small in terms of its complexity, then the eigenvalues of the new graph H (e.g., the query graph) will remain close to their corresponding non-zero eigenvalues of the original graph G (e.g., the model graph), independent of where the perturbation is applied to G. The magnitude of the eigenvalue distortion is a function of the number of vertices added to the graph due to the noise or occlusion. Specifically, if the noise matrix E_H introduces k new vertices to G, then the distortion of every eigenvalue can be bounded by $\sqrt{k-1}$ (Neumaier [18]). This bound can be further tightened if the noise matrix has simple structure. For example, if E_H represents a simple path on k vertices, then its norm can be bounded by $(2\cos\pi/(k+1))$ (Lovász and Pelikán [15]). In short, large distortions are due to the introduction/deletion of large, complex subgraphs to/from G, while small structural changes will have little impact on the higher order eigenvalues G. The eigenvalues of a graph are therefore stable under minor perturbations in graph structure.

3.2 Formulating an Index

Having established the stability of a DAG's eigenvalues under minor perturbation of the graph, we can now proceed to define an index based on the eigenvalues. We could, for example, define a vector to be the sorted eigenvalues of a DAG, with the resulting index used to retrieve nearest neighbors in a model DAG database having similar topology. However, for large DAGs, the dimensionality of the index (and model DAG database) would be prohibitively large. Our solution to this problem 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 $T_1, T_2, \ldots, T_S$. For each subgraph, T_i, whose root degree is $\delta(T_i)$, we compute the eigenvalues of T_i's submatrix, sort the eigenvalues in decreasing order by absolute value, and let S_i be the sum of the $\delta(T_i) - 1$ largest absolute values. 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 each nonterminal node in the DAG, computed over the subgraph rooted at that node. The reasons for computing a description for each node, rather than just the root, will become clear in the next section.

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, and 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 submatrix 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 [19], 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 [19]). *For the sum of the first k eigenvalues of a symmetric matrix A, the following semidefinite programming* *characterization holds:*

$$\begin{aligned} \lambda_1(A) + \ldots + \lambda_k(A) = \max\ & A \bullet U \\ \text{s.t.}\ \ & \text{trace}(U) = k \\ & 0 \preceq U \preceq I, \end{aligned} \tag{4}$$

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.

3.3 Properties of the Index

Our topological index satisfies the six criteria outlined in Section 1. The eigen-decomposition yields a low-dimensional (criterion 1) vector assigned to each node in the DAG, which captures the local topology of the subgraph rooted at that node (criterion 2 – this will be more fully explained in Section 3.4). Furthermore, a node's vector is invariant to any consistent re-ordering of the node's subgraphs (criterion 3). The components of a node's vector are based on summing the largest eigenvalues of its subgraph's adjacency submatrix. Although our dimensionality-reducing summing operation has cost us some uniqueness, our partial sums still have very low ambiguity (criterion 4). From the sensitivity analysis in Section 3.1, we have shown our index to be stable to minor perturbations of the DAG's topology (criterion 5). As shown in Theorem 2, these sums can be computed even more efficiently (criterion 6) than the eigenvalues themselves. The vector labeling of all DAGs isomorphic to T not only has the same vector labeling, but spans the same subspace in $R^{\Delta(T)-1}$. Moreover, this extends to any DAG which has a subgraph isomorphic to a subgraph of T.

3.4 Candidate Selection

Given a query DAG corresponding to an image, our task is to search the model DAG database for one or more model DAGs which are similar to the image DAG. If the number of model DAGs is large, a linear search of the database is intractable. Therefore, the goal of our indexing mechanism is to quickly select a small number of model candidates for verification. Those candidates will share coarse topological structure with the image DAG (or one of its subgraphs, if it is occluded or poorly segmented). Hence, we begin by mapping the topology of the image DAG to a set of indices that capture its structure, discounting any information associated with its nodes. We then describe the structure of our model database, along with our mechanism for indexing into it to yield a small set of model candidates. Finally, we present a local evidence accumulation procedure that will allow us to index in the presence of occlusion.

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, which we will call 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, as 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 large occlusion, both of which result in the addition or removal of significant structure. In either case, altering the structure of the DAG will affect the TSV's computed at its nodes. The signatures corresponding to the roots of those subgraphs (DAGs) that survive the occlusion will not change. However, the signature of a root of a subgraph that has undergone any perturbation will change which, in turn, will affect the signatures of any of its ancestor nodes, including the root of the entire DAG. We therefore cannot rely on indexing solely with the root's signature. Instead, we will exploit 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., [13,8]. 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. Nearby candidates can be retrieved using a nearest neighbor retrieval method. In our implementation, model points were stored in a Voronoi database, whose off-line construction (decomposition) is $O((kn)^{\lfloor(\delta+1)/2\rfloor+1}) + O((kn)^{\lfloor(\delta+1)/2\rfloor}\log(kn))$ ([21]), and whose run-time search is $O(\log^{\delta}(kn))$ for fixed δ [21]; details are given in [33].

Accumulating Local Evidence Each node in the image DAG will generate a set of (model object, model node) votes. To collect these votes, we set up an accumulator with one bin per model object. Furthermore, we can weight the votes that we add to the accumulator. For example, if the label of the model node is not compatible with the label of its corresponding image node, then the vote is discarded, i.e., it receives a zero weight. If the nodes are label-compatible, then we can weight the vote according to the distance between their respective TSV's – the closer the signatures, the more weight the vote gets.

We can also weight the vote according to the complexity of its corresponding subgraph, allowing larger and more complex subgraphs (or "parts") to have higher weight. This can be easily accommodated within our eigenvalue framework, for the richer the structure, the larger its maximum eigenvalue:

Theorem 3 (Lovász and Pelikán [15]). *Among the graphs with n vertices, the star graph ($K_{1,n-1}$), has the largest eigenvalue ($\sqrt{n-1}$), while the path on n nodes (P_n) has the smallest eigenvalue ($2\cos\pi/(n+1)$).*

Since the size of the eigenvalues, and hence their sum, is proportional to both the branching factor as well as the number of nodes, the magnitude of the signature is also used to weight the vote. If we let u be the TSV of an image DAG node and v the TSV of a model DAG node that is sufficiently close, the weight of the resulting vote, i.e., the local evidence for the model, is computed as (we use $p = 2$):

$$W = \frac{||u||_p}{1 + ||v - u||_p} \tag{5}$$

Once the evidence accumulation is complete, those models whose support is sufficiently high are selected as candidates for verification. The bins can, in effect, be organized in a heap, requiring a maximum of $O(\log k)$ operations to maintain the heap when evidence is added, where k is the number of non-zero object accumulators. Once the top-scoring models have been selected, they must be individually verified according to some matching algorithm.

4 Matching Hierarchical Structures

Each of the top-ranking candidates emerging from the indexing process must be verified to determine which is most similar to the query. If there were no clutter, occlusion, or noise, our problem could be formulated as a graph isomorphism problem. If we allowed clutter and limited occlusion, we would search for the largest isomorphic subgraphs between query and model. Unfortunately, with the presence of noise, in the form of the addition of spurious graph structure and/or the deletion of salient graph structure, large isomorphic subgraphs may simply not exist. It is here that we call on our eigen-characterization of graph structure to help us overcome this problem.

Each node in our graph (query or model) is assigned a TSV, which reflects the underlying structure in the subgraph rooted at that node. If we simply discarded all the edges in our two graphs, we would be faced with the problem of finding the best correspondence between the nodes in the query and the nodes in the model; two nodes could be said to be in close correspondence if the distance between their TSVs (and the distance between their domain-dependent node labels) was small. In fact, such a formulation amounts to finding the maximum cardinality, minimum weight matching in a bipartite graph spanning the two sets of nodes. At first glance, such a formulation might seem like a bad idea (by throwing away all that important graph structure!) until one recalls that the graph structure is really encoded in the node's TSV. Is it then possible to reformulate a noisy, largest isomorphic subgraph problem as a simple bipartite matching problem?

Unfortunately, in discarding all the graph structure, we have also discarded the underlying hierarchical structure. There is nothing in the bipartite graph matching formulation that ensures that hierarchical constraints among corresponding nodes are obeyed, i.e., that parent/child nodes in one graph don't match child/parent nodes in the other. This reformulation, although softening the overly strict constraints imposed by the largest isomorphic subgraph formulation, is perhaps too weak. We could try to enforce the hierarchical constraints in our bipartite matching formulation, but no polynomial-time solution is known to exist for the resulting formulation. Clearly, we seek an efficient approximation method that will find corresponding nodes between two noisy, occluded DAGs, subject to hierarchical constraints.

Our algorithm, a modification to Reyner's algorithm [23], combines the above bipartite matching formulation with a greedy, best-first search in a recursive procedure to compute the corresponding nodes in two rooted DAGs. As in the above bipartite matching formulation, we compute the maximum cardinality, minimum weight matching in the bipartite graph spanning the two sets of nodes. Edge weight will encode a function of both topological similarity as well as domain-dependent node similarity. The result will be a selection of edges yielding a mapping between query and model nodes. As mentioned above, the computed mapping may not obey hierarchical constraints. We therefore greedily choose only the best edge (the two most similar nodes in the two graphs, representing in some sense the two most similar subgraphs), add it to the solution set, and recursively apply the procedure to the subgraphs defined by these two nodes. 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. In this way, the search for corresponding nodes is focused in corresponding subgraphs (rooted DAGs) in a top-down manner, thereby ensuring that hierarchical constraints are obeyed.

Before formalizing our algorithm, some definitions are in order. Let $G = (V_1, E_1)$ and $H = (V_2, E_2)$ be the two DAGs 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 topological signature vector (TSV), introduced in Section 3.2.[2] Furthermore, for any pair of vertices u and v, let $C(u, v)$ denote the domain dependent node label distance between vertices u and v. 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.

[2] 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.

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 node labels, and has the value ∞ otherwise. Next, we form a bipartite edge weighted graph $\mathcal{G}(V_1, V_2, E_{\mathcal{G}})$ with edge weights from the matrix $\Pi(G,H)$.[3] Using the scaling algorithm of Goemans, Gabow, and Williamson [9], 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 rooted subgraphs 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 bottom of the DAGs, forming a list of ordered correspondence lists (or matchings) $\{\mathcal{M}_1, \ldots, \mathcal{M}_k\}$. In backtracking step i, we remove any subgraph 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 recompute $\mathcal{M}_i$ on the subgraphs 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 a traditional depth-first search, we are dynamically recomputing the branches at each node in the search tree. Processing at a particular node will terminate when either rooted subgraph loses all of its nodes to the solution set. The precise algorithm is given in Figure 1; additional details and examples are given in [35,32].

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 $n = \max(n_1, n_2)$ [9]. 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 above algorithm therefore provides, in polynomial time better than $O(n^3)$ an approximate optimal solution to the largest isomorphic subgraph problem in the presence of noise.

[3] $G(A,B,E)$ 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 isomorphism($G$,$H$)
    $\Phi(G, H) \leftarrow \emptyset$
    $d \leftarrow \max(\delta(G), \delta(H))$
    for $u \in V_G$ compute $\chi(u) \in R^{d-1}$ (see Section 3.2)
    for $v \in V_H$ compute $\chi(v) \in R^{d-1}$ (see Section 3.2)
    call match(root($G$),root($H$))
    return(cost($\Phi(G, H)$))
end

procedure match($u$,$v$)
    do
        {
        let $G_u \leftarrow$ rooted subgraph of $G$ at $u$
        let $H_v \leftarrow$ rooted subgraph 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 [9])
        $(u', v') \leftarrow$ minimum weight pair in $\mathcal{M}$
        $\Phi(G, H) \leftarrow \Phi(G, H) \cup \{(u', v')\}$
        call match($u'$,$v'$)
        $G_u \leftarrow G_u - \{x | x \in V_{G_u}$ and $(x, w) \in \Phi(G, H)\}$
        $H_v \leftarrow H_v - \{y | y \in V_{H_v}$ and $(w, y) \in \Phi(G, H)\}$
        }
    while ($G_u \neq \emptyset$ and $H_v \neq \emptyset$)
```

Fig. 1. Algorithm for Matching Two Hierarchical Structures

5 Demonstration

In this section, we briefly illustrate our unified approach to indexing and matching on two different object recognition domains.

5.1 2-D Generic Object Recognition

To demonstrate our approach to indexing, we turn to the domain of 2-D object recognition [33,35]. We adopt a representation for 2-D shape that is based on a coloring of the shocks (singularities) of a curve evolution process acting on simple closed curves in the plane [12]. Any given 2-D shape gives rise to a rooted *shock tree*, in which nodes represent parts (whose labels are drawn from four qualitatively-defined classes) and arcs represent relative time of formation (or relative size). Figure 2 illustrates a 2-D shape, along with its corresponding shock tree.

We demonstrate our indexing algorithm on a database of 60 object silhouettes. In Figure 3, query shapes are shown in the left column, followed by the top ten database candidates (based on accumulator scores), ordered left to right. The

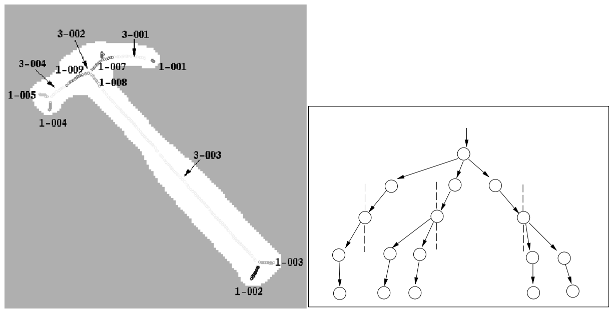

Fig. 2. An illustrative example taken from [35]. The labels on the shocks of the hammer (left) correspond to vertices in the derived shock graph (right).

candidate in the box is the closest candidate, found by computing the distance (using the matcher) between the query and each database shape (linear search). For unoccluded shapes, the results are very encouraging, with the correct candidate ranking very highly. With increasing occlusion, high indexing ambiguity (smaller unoccluded subtrees are less distinctive and "vote" for many objects that contain them) leads to slightly decreased performance, although the target object is still ranked highly. We are investigating the incorporation of more node information, as well as the encoding of subtree relations (currently, each subtree votes independently, with no constraints among subtrees enforced) to improve indexing performance.

5.2 View-Based 3-D Object Recognition

For our next demonstration, we turn to the domain of view-based object recognition, in which salient blobs are detected in a multiscale wavelet decomposition of an image [3]. Figure 4 shows three images of an origami figure, along with their computed multiscale blob analyses (shown inverted for improved visibility). Each image yields a DAG, in which nodes correspond to blobs, and arcs are directed from blobs at coarser scales to blobs at finer scales if the distance between their centers does not exceed the sum of their radii. Node similarity is a function of the difference in saliency between two blobs.

In Figure 5, we show the results of matching the first and second, and first and third images, respectively, in Figure 4. The first and third images are taken from similar viewpoints, so the match yields many good correspondences. However, for the first and third images, taken from different viewpoints, fewer corresponding features were found. Note that since only the DAG structure is matched (and not DAG geometry), incorrect correspondences may arise when nodes have similar saliency and size but different relative positions. A stronger match can be attained by enforcing geometric consistency (see [34]).

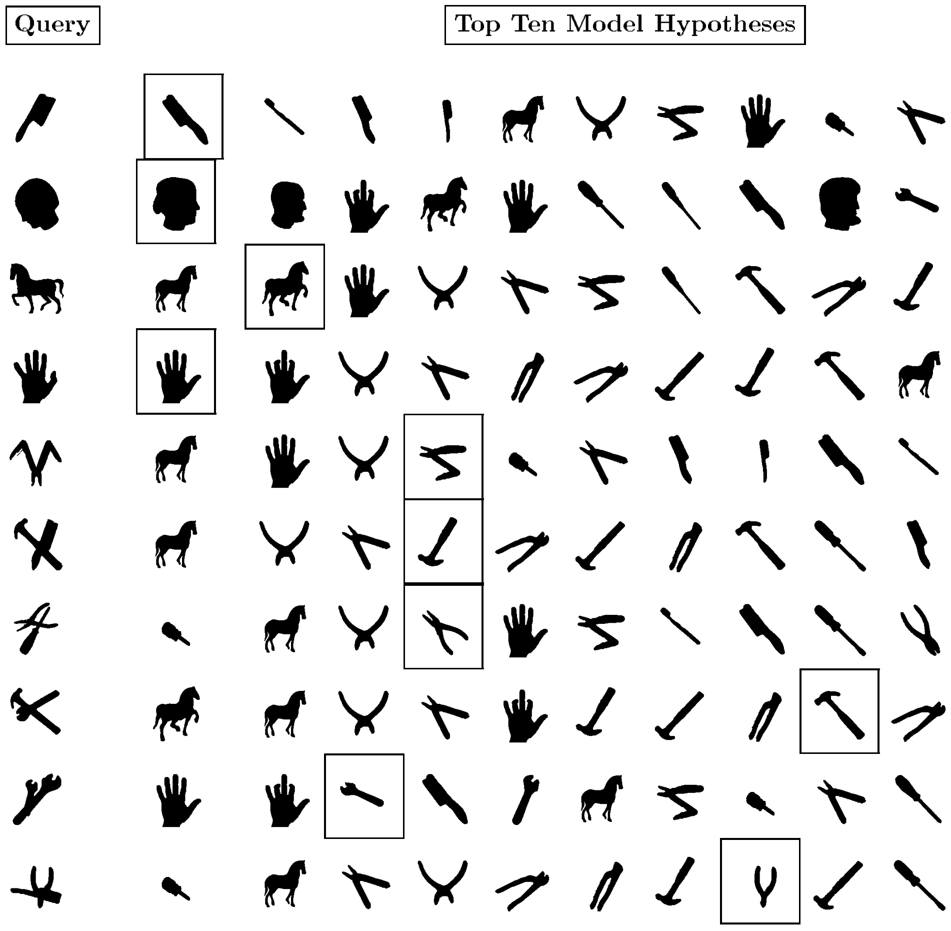

Fig. 3. Indexing Demonstration. Shape on left is query shape, followed by top ten candidates in decreasing score from left to right. Boxed candidate is closest to query (using a linear search based on matcher).

6 Conclusions

We have presented a unified solution to the tightly-coupled problems of indexing and matching hierarchical structures. The structural properties of a DAG are captured by the eigenvalues of its corresponding adjacency matrix. These eigenvalues, in turn, can be combined to yield a low-dimensional vector representation of DAG structure. The resulting vectors can be used to retrieve, in the presence of noise and occlusion, structurally similar candidates from a database using efficient nearest-neighbor searching methods. Moreover, these same vectors contribute to the edge weights in a recursive bipartite matching formulation that

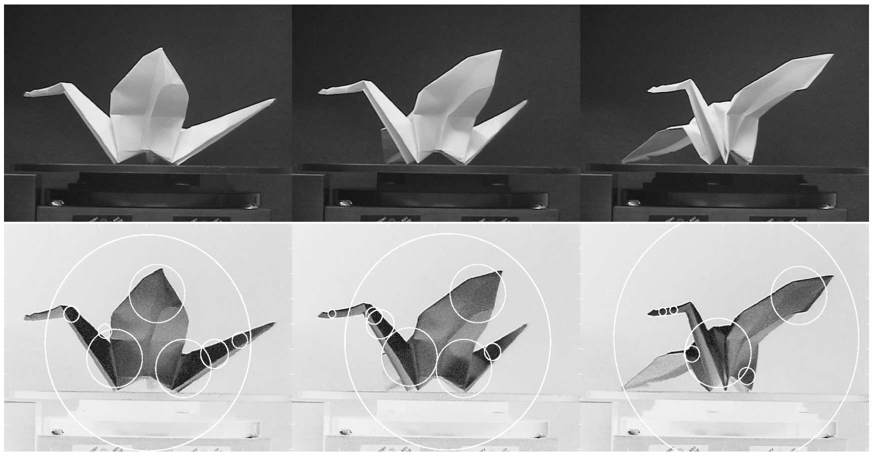

Fig. 4. Top row contains original images, while bottom row (shown inverted for improved visibility) contains corresponding multiscale blob analyses (see text).

computes an approximation to the largest isomorphic subgraph of two graphs (query and model) in the presence of noise and occlusion. Our formulation is general and is applicable to the indexing and matching of any rooted hierarchical structure, whether DAG or rooted tree. The only domain-dependent component is the node label distance function, which is used in conjunction with the topological distance function to compute a bipartite edge weight. We have tested the approach extensively on the indexing and matching of shock graphs, and have only begun to test the approach on other domains, including the preliminary results reported in this paper in the domain of multiscale blob matching.

7 Acknowledgements

The authors gratefully acknowledge the support of NSERC, NSF, and ARO for their support of this work. The authors would also like to thank our collaborators, Kaleem Siddiqi, Steven Zucker, and Ee-Chien Chang, for their kind permission to use the figures shown in Section 5.

References

1. F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM J. Optim.*, 5(1):13–51, 1995.
2. K.L. Boyer and A.C. Kak. Structural stereopsis for 3-D vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(2):144–166, March 1988.
3. E. Chang, Stephane Mallat, and Chee Yap. Wavelet foveation. *Journal of Applied and Computational Harmonic Analysis*, 9(3):312–335, October 2000.

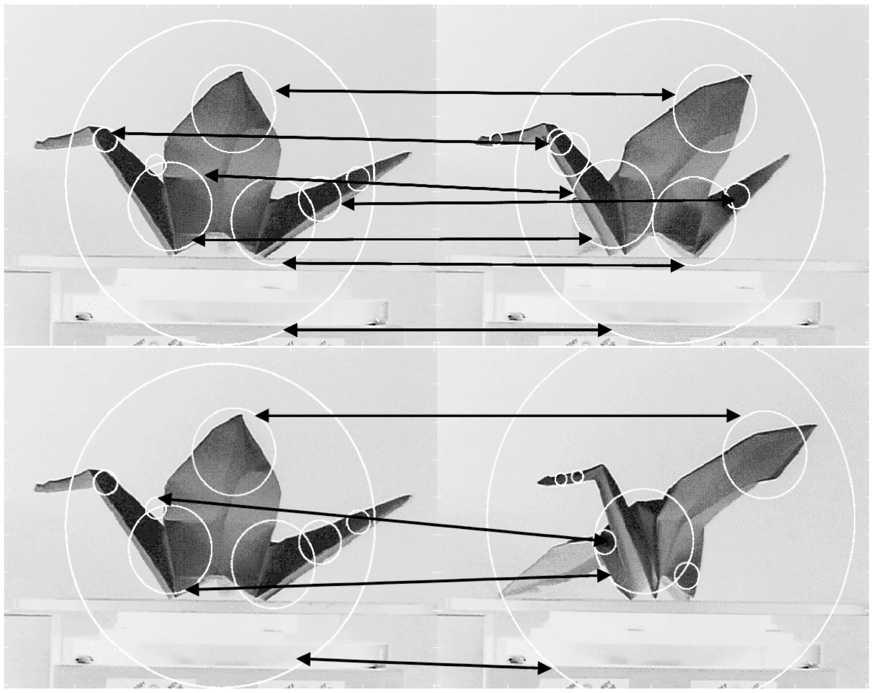

Fig. 5. Results of matching the first two, and first and third images, respectively, in Figure 4. The first two images are taken from similar viewpoints, and hence many correspondences were found. However, for the first and third images, taken from different viewpoints, fewer corresponding features were found. Note that since only the DAG structure is matched, Top row contains original images, while bottom row (shown inverted for improved visibility) contains corresponding multiscale blob analyses (see text).

4. W. J. Christmas, J. Kittler, and M. Petrou. Structural matching in computer vision using probabilistic relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:749–764, August 1995.
5. A.D. Cross and E.R. Hancock. Graph matching with a dual-step em algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1236–1253, November 1998.
6. S. Dickinson, A. Pentland, and A. Rosenfeld. From volumes to views: An approach to 3-D object recognition. *CVGIP: Image Understanding*, 55(2):130–154, 1992.
7. M. A. Eshera and K. S. Fu. A graph distance measure for image analysis. *IEEE Trans. SMC*, 14:398–408, May 1984.
8. P. Flynn and A. Jain. 3D object recognition using invariant feature indexing of interpretation tables. *CVGIP:Image Understanding*, 55(2):119–129, March 1992.
9. H. Gabow, M. Goemans, and D. Williamson. An efficient approximate algorithm for survivable network design problems. *Proc. of the Third MPS Conference on Integer Programming and Combinatorial Optimization*, pages 57–74, 1993.
10. Steven Gold and Anand Rangarajan. A graduated assignment algorithm for graph matching. *IEEE PAMI*, 18(4):377–388, 1996.

11. W. Kim and A. C. Kak. 3d object recognition using bipartite matching embedded in discrete relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):224–251, 1991.
12. B. B. Kimia, A. Tannenbaum, and S. W. Zucker. Shape, shocks, and deformations I: The components of two-dimensional shape and the reaction-diffusion space. *International Journal of Computer Vision*, 15:189–224, 1995.
13. Y. Lamdan, J. Schwartz, and H. Wolfson. Affine invariant model-based object recognition. *IEEE Transactions on Robotics and Automation*, 6(5):578–589, October 1990.
14. Tony Lindeberg. Detecting Salient Blob–Like Image Structures and Their Scales With a Scale–Space Primal Sketch—A Method for Focus–of–Attention. *International Journal of Computer Vision*, 11(3):283–318, December 1993.
15. L. Lovász and J. Pelicán. On the eigenvalues of a tree. *Periodica Math. Hung.*, 3:1082–1096, 1970.
16. B. T. Messmer and H. Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:493–504, May 1998.
17. H. Murase and S. Nayar. Visual learning and recognition of 3-D objects from appearance. *International Journal of Computer Vision*, 14:5–24, 1995.
18. A. Neumaier. Second largest eigenvalue of a tree. *Linear Algebra and its Applications*, 46:9–25, 1982.
19. M. L. Overton and R. S. Womersley. Optimality conditions and duality theory for minimizing sums of the largest eigenvalues of symmetric matrices. *Math. Programming*, 62(2):321–357, 1993.
20. M. Pelillo, K. Siddiqi, and S. Zucker. Matching hierarchical structures using association graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(11):1105–1120, November 1999.
21. F. Preparata and M. Shamos. *Computational Geometry.* Springer-Verlag, New York, NY, 1985.
22. Recognition and shape synthesis of 3D objects based on attributed hypergraphs. A. wong and s. lu and m. rioux. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11:279–290, 1989.
23. S. W. Reyner. An analysis of a good algorithm for the subtree problem. *SIAM J. Comput.*, 6:730–732, 1977.
24. 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, May 1983.
25. S. Sarkar. Learning to form large groups of salient image features. In *IEEE CVPR*, Santa Barbara, CA, June 1998.
26. S. Sclaroff and A. Pentland. Modal matching for correspondence and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(6):545–561, June 1995.
27. K. Sengupta and K. Boyer. Using spectral features for modelbase partitioning. In *Proceedings, International Conference on Pattern Recognition*, Vienna, Austria, August 1996.
28. L. Shapiro and M. Brady. Feature-based correspondence: an eigenvector approach. *Image and Vision Computing*, 10(5):283–288, June 1992.
29. L. G. Shapiro and R. M. Haralick. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3:504–519, 1981.

30. L. G. Shapiro and R. M. Haralick. A metric for comparing relational descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7:90–94, January 1985.
31. J. Shi and J. Malik. Normalized cuts and image segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, San Juan, Puerto Rico, June 1997.
32. A. Shokoufandeh and S. Dickinson. Applications of bipartite matching to problems in object recognition. In *Proceedings, ICCV Workshop on Graph Algorithms and Computer Vision (web proceedings: http://www.cs.cornell.edu/iccv-graph-workshop/papers.htm)*, September 1999.
33. A. Shokoufandeh, S. Dickinson, K. Siddiqi, and S. Zucker. Indexing using a spectral encoding of topological structure. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 491–497, Fort Collins, CO, June 1999.
34. A. Shokoufandeh, I. Marsic, and S. Dickinson. View-based object recognition using saliency maps. *Image and Vision Computing*, 17:445–460, 1999.
35. K. Siddiqi, A. Shokoufandeh, S. Dickinson, and S. Zucker. Shock graphs and shape matching. *International Journal of Computer Vision*, 30:1–24, 1999.
36. H. Sossa and R. Horaud. Model indexing: The graph-hashing approach. In *Proceedings, IEEE CVPR*, pages 811–814, 1992.
37. G.W. Stewart and J.-G. Sun. *Matrix Perturbation Theory.* Academic Press, San Diego, 1990.
38. M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.
39. J. Wilkinson. *The Algebraic Eigenvalue Problem.* Clarendon Press, Oxford, England, 1965.
40. A. Witkin. Scale space filtering. In Alex Pentland, editor, *From Pixels to Predicates.* Ablex, Norwood, NJ, 1986.
41. A. K. C. Wong and M. You. Entropy and distance of random graphs with application to structural pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7:599–609, September 1985.
42. S. Zhu and A. L. Yuille. Forms: a flexible object recognition and modelling system. *International Journal of Computer Vision*, 20(3):187–212, 1996.