8

Many-to-Many Feature Matching in Object Recognition

Ali Shokoufandeh[1], Yakov Keselman[2], Fatih Demirci[1], Diego Macrini[3], and Sven Dickinson[3]

[1] Computer Science Department, Drexel University, 3200 Chestnut St., Philadelphia, PA 19104
[2] School of Computer Science, DePaul University, 243 S. Wabash Ave., Chicago IL 60604
[3] Department of Computer Science, University of Toronto, 6 King's College Rd., Toronto, Ontario M5S 3G4

Abstract. One of the bottlenecks of current recognition (and graph matching) systems is their assumption of one-to-one feature (node) correspondence. This assumption breaks down in the generic object recognition task where, for example, a collection of features at one scale (in one image) may correspond to a single feature at a coarser scale (in the second image). Generic object recognition therefore requires the ability to match features many-to-many. In this paper, we will review our progress on three independent object recognition problems, each formulated as a graph matching problem and each solving the many-to-many matching problem in a different way. First, we explore the problem of learning a 2-D shape class prototype (represented as a graph) from a set of object exemplars (also represented as graphs) belonging to the class, in which there may be no one-to-one correspondence among extracted features. Next, we define a low-dimensional, spectral encoding of graph structure and use it to match entire subgraphs whose size can be different. Finally, in very recent work, we embed graphs into geometric spaces, reducing the many-to-many graph matching problem to a weighted point matching problem, for which efficient many-to-many matching algorithms exist.

8.1 Introduction

One of the fundamental obstacles to generic object recognition is the common assumption that saliency in the image implies saliency in the model. Alternatively, for every salient image feature (e.g., a long, high-contrast edge, a homogeneous region, a perceptual group of lines, a patch of pixel values, or even a neighborhood-encoded interest point), there's a corresponding salient model feature. Under this one-to-one correspondence assumption, object models are constrained to be 2-D or 3-D templates of the object, and recognition is constrained to be exemplar-based.

When image features are represented as graphs, this one-to-one feature correspondence assumption translates into a one-to-one correspondence (or isomorphism) assumption. However, there are a variety of conditions that may lead to graphs that represent visually similar image feature configurations yet do not contain a single one-to-one node correspondence. For example, due to noise or segmentation errors, a single feature (node) in one graph may map to a collection of broken features (nodes) in another graph. Or, due to scale differences, a single, coarse-grained feature in one graph may

H.I. Christensen and H.-H. Nagel (Eds.): Cognitive Vision Systems, LNCS 3948, pp. 107–125, 2006.

map to a collection of fine-grained features in another graph. In general, we seek not a one-to-one correspondence between image features (nodes), but rather a many-to-many correspondence.

In this paper, we review three of our approaches to the problem of many-to-many graph matching for object recognition. In the first approach, in material drawn from [220], we explore the problem of generic model acquisition from a set of images containing exemplars belonging to a known class. Specifically, we present to the system a set of region adjacency graphs, representing region segmentations of the exemplar images, for which a single region (node-to-node) correspondence may not exist across the input graphs. The goal is to search the space of abstractions of these input graphs for the most informative abstraction common to all the input exemplars.

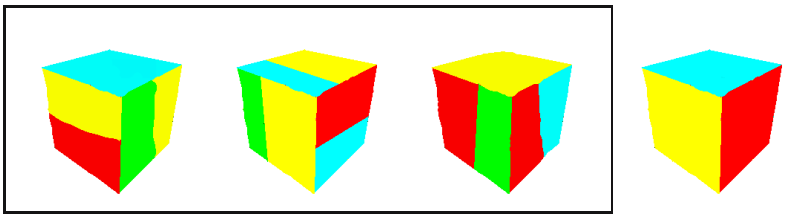
In the second approach, in material drawn from [424], we explore the problem of graph matching based on a spectral embedding of hierarchical graph structure. We map a directed acyclic graph (DAG) to a low-dimensional vector, or signature, that characterizes the topological properties of the DAG. This signature allows us to compute the distance between two graphs without insisting on isomorphism or one-to-one node correspondence. Finally, in material drawn from [221], we explore the problem of many-to-many graph matching using low-distortion graph embedding techniques. By embedding the nodes of a graph into a geometric space, we can reformulate the many-to-many graph matching problem as a many-to-many point matching problem, for which the Earth Mover's Distance (EMD) algorithm is ideal.

The ability to match features (or graphs) many-to-many is essential for generic object recognition, in which shape similarity between two objects exists not in the form of low-level features that appear explicitly in the image, but rather in the form of feature (graph) abstractions. The three approaches reviewed in this paper provide three different perspectives on the problem. The abstraction of image data is an important problem that's been largely unaddressed by the object recognition community, and the three methods reported here can be seen as generating and matching graph abstractions.

8.2 Generic Model Acquisition from Examples

Assume that we are presented with a collection of images, such that each image contains a single exemplar, all exemplars belong to a single known class, and the viewpoint with respect to the exemplar in each image is similar. Fig. 8.1(a) illustrates a simple example in which three different images, each containing a block in a similar orientation, are presented to the system. Our task is to find the common structure in these images, under the assumption that structure that is common across many exemplars of a known class must be definitive of that class. Fig. 8.1(b) illustrates the class "abstraction" that is derived from the input examples. In this case, the domain of input examples is rich enough to "intersect out" irrelevant structure (or appearance) of the block. However, had many or all the exemplars had vertical stripes, the approach would be expected to include vertical stripes in that view of the abstracted model.

Any discussion of model acquisition must be grounded in image features. In our case, each input image will be region-segmented to yield a region adjacency graph. Similarly, the output of the model acquisition process will yield a region adjacency graph containing



(a) (b)

Fig. 8.1. Illustrative example of generic model acquisition: (a) Input exemplars belonging to a single known class; (b) generic model abstracted from examples

the "meta-regions" that define a particular view of the generic model. Other views of the exemplars would similarly yield other views of the generic model. The integration of these views into an optimal partitioning of the viewing sphere, or the recovery of 3-D parts from these views, is beyond the scope of this paper. For now, the result will be a collection of 2-D views that describes a generic 3-D object. This collection would then be added to the view-based object database used at recognition time.

8.2.1 Problem Formulation

Returning to Fig. 8.1, let us now formulate our problem more concretely. As we stated, each input image is processed to form a region adjacency graph (we employ the region segmentation algorithm of Felzenzwalb and Huttenlocher [119]). Let us now consider the region adjacency graph corresponding to one input image. We will assume, for now, that our region adjacency graph represents an oversegmentation of the image. In [220], we discuss the problem of undersegmentation, and how our approach can accommodate it. The space of all possible region adjacency graphs formed by any sequence of merges of adjacent regions will form a lattice, as shown in Fig. 8.2. The lattice size is exponential in the number of regions obtained after initial oversegmentation.[4]

Each of the input images will yield its own lattice. The bottom node in each lattice will be the original region adjacency graph. In all likelihood, if the exemplars have different shapes (within-class deformations) and/or surface markings, the graphs forming the bottom of their corresponding lattices may bear little or no resemblance to each other. Clearly, similarity between the exemplars cannot be ascertained at this level, for there does not exist a one-to-one correspondence between the "salient" features (i.e., regions) in one graph and the salient features in another. On the other hand, the top of each exemplar's lattice, representing a silhouette of the object (where all regions have been merged into one region), carries little information about the salient surfaces of the object.

We can now formulate our problem more precisely, recalling that a lattice consists of a set of nodes, with each node corresponding to an entire region adjacency graph. Given

[4] Indeed, considering the simple case of a long rectangular strip subdivided into $n+1$ adjacent rectangles, the first pair of adjacent regions able to be merged can be selected in n ways, the second in $n-1$, and so on, giving a lattice size of $n!$.

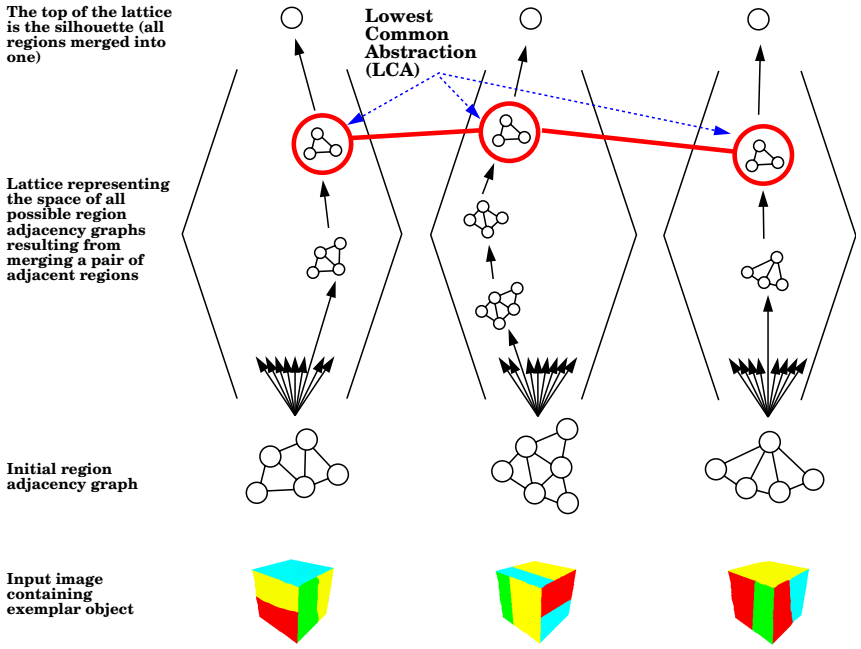


Fig. 8.2. The Lowest Common Abstraction (LCA) of a set of input exemplars (see, too, Figure 13.3)

N input image exemplars, $E_1, E_2, \ldots, E_N$, let $L_1, L_2, \ldots, L_N$ be their corresponding lattices, and for a given lattice, L_i, let $L_i n_j$ be its constituent nodes, each representing a region adjacency graph, G_{ij}. We define a *common abstraction*, or CA, as a set of nodes (one per lattice) $L_1 n_{j_1}, L_2 n_{j_2}, \ldots, L_N n_{j_N}$ such that for any two nodes $L_p n_{j_p}$ and $L_q n_{j_q}$, their corresponding graphs G_{pj_p} and G_{qj_q} are isomorphic. Thus, the root node (whose graph consists of one node representing the silhouette region) of each lattice is a common abstraction. We define the *lowest common abstraction*, or LCA, as the common abstraction whose underlying graph has maximal size (in terms of number of nodes). Given these definitions, our problem can be simply formulated as finding the LCA of N input image exemplars.

Intuitively, we are searching for a node (region segmentation) that is common to every input exemplar's lattice and that retains the maximum amount of structure common to all exemplars. Unfortunately, the presence of a single heavily undersegmented exemplar (a single-node silhouette in the extreme case) will drive the LCA toward the trivial silhouette CA. In a later section, we will relax our LCA definition to make it less sensitive to such outliers.

8.2.2 The LCA of Two Examples

For the moment, we will focus our attention on finding the LCA of two lattices, while in the next section, we will accommodate any number of lattices. Since the input lattices are exponential in the number of regions, actually computing the lattices is intractable.

Clearly, we need a means for focusing the search for the LCA that avoids significant lattice generation. Our approach will be to restrict the search for the LCA to the *intersection* of the lattices. Typically, the intersection of two lattices is much smaller than either lattice (unless the images are very similar), and leads to a tractable search space. But how do we generate this new "intersection" search space without enumerating the lattices?

Our solution is to work top-down, beginning with a node known to be in the intersection lattice – the root node, representing a single region (silhouette). If the intersection lattice contains only this one node, i.e., one or both of the region segmented images contain a single region, then the process stops and the LCA is simply the root (silhouette). However, in most cases, the root of each input lattice is derived from an input region adjacency graph containing multiple regions. So, given two silhouettes, each representing the apex of a separate, non-trivial lattice, we have the opportunity to search for a lower abstraction (than the root) common to both lattices. Our approach will be to find a decomposition of each silhouette region into two subregions, such that: 1) the shapes of the corresponding subregions are similar (for this, we employ the shape matching method outlined in Section 8.3 [424]), and 2) the relations among the corresponding regions are similar. Since there is an infinite number of possible decompositions of a region into two component regions, we will restrict our search to the space of decompositions along region boundaries in the original region adjacency graphs. Note that there may be multiple 2-region decompositions that are common to both lattices; each is a member of the intersection set.

Assuming that we have some means for ranking the matching decompositions (if more than one exist), we pick the best one (the remainder constituting a set of backtracking points), and recursively apply the process to each pair of isomorphic component subregions.[5] The process continues in this fashion, "pushing" its way down the intersection lattice, until no further decompositions are found. This lower "fringe" of the search space represents the LCA of the original two lattices. The specific algorithm for choosing the optimal pair of decompositions is given in [220], and can be summarized as follows:

1. Map each region adjacency graph to its dual *boundary segment graph*, in which boundary segments become nodes and edges capture segment adjacency.
2. Form the product graph (or association graph) of the two boundary segment graphs. Nodes and arcs in the product graph correspond to pairs of nodes and arcs, respectively, in the boundary segment graphs. Therefore, a path in the product graph corresponds to a pair of paths in the boundary segment graphs which, in turn, correspond to a pair of decompositions of the region adjacency graphs.
3. With appropriate edge weights, along with a suitable objective function, the optimal pair of corresponding decompositions corresponds to the shortest path in the product graph.
4. The optimal pair of decompositions is verified in terms of satisfying the criteria of region shape similarity and region relation consistency.

[5] Each subregion corresponds to the union of a set of regions corresponding to nodes belonging to a connected subgraph of the original region adjacency graph.

8.2.3 The LCA of Multiple Examples

So far, we've addressed only the problem of finding the LCA of two examples. How, then, can we extend our approach to find the LCA of multiple examples? Furthermore, when moving toward multiple examples, how do we prevent a "noisy" example, such as a single, heavily undersegmented silhouette, from driving the solution toward the trivial silhouette? To extend our two-exemplar LCA solution to a robust, multi-exemplar solution, we begin with two important observations. First, the LCA of two exemplars lies in the intersection of their abstraction lattices. Thus, both exemplar region adjacency graphs can be transformed into their LCA by means of sequences of region merges. Second, the total number of merges required to transform the graphs into their LCA is minimal among all elements of the intersection lattice, i.e., the LCA lies at the lower fringe of the lattice.

Our solution begins by constructing an approximation to the intersection lattice of multiple examples. Consider the closure of the set of the original region adjacency graphs under the operation of taking pairwise LCA's. In other words, starting with the initial region adjacency graphs, we find their pairwise LCA's, then find pairwise LCA's of the resulting abstraction graphs, and so on (note that duplicate graphs are removed). We take all graphs, original and LCA, to be nodes of a new *closure* graph. If graph H was obtained as the LCA of graphs G_1 and G_2, then directed arcs go from nodes corresponding to G_1, G_2 to the node corresponding to H in the closure graph.

Next, we will relax the first property above to accommodate "outlier" exemplars, such as undersegmented, input silhouettes. Specifically, we will not enforce that the LCA of multiple exemplars lies in the intersection set of all input exemplars. Rather, we will choose a node in our approximate intersection lattice that represents a "low abstraction" for many (but not necessarily all) input exemplars. More formally, we will define the LCA of a set of exemplar region adjacency graphs to be that element in the intersection of two or more abstraction lattices that minimizes the total number of edit operations (merges or splits) required to obtain the element from *all* the given exemplars. If a node in the intersection lattice lies along the lower fringe with respect to a number of input exemplars, then its sum distance to all exemplars is small. Conversely, the sum distance between the silhouette outlier (in fact, the true LCA) and the input exemplars will be large, eliminating that node from contention. Our algorithm for computing this "median" of the closure graph, along with an analysis of its complexity, is given in [220].

8.2.4 Demonstration

In Figure 8.3, we illustrate the results of our approach applied to a set of three coffee cup images, respectively. The lower row represents the original images, the next row up represents the input region segmented images (with black borders), while the LCA is shown with an orange border. The closure graph consists of only four members, with the same pairwise LCA emerging from all input pairs. The solution captures our intuitive notion of the cup's surfaces, with a handle, a body, a top, and the hole in the handle (the algorithm cannot distinguish a surface from a hole). Additional examples can be found in [220].

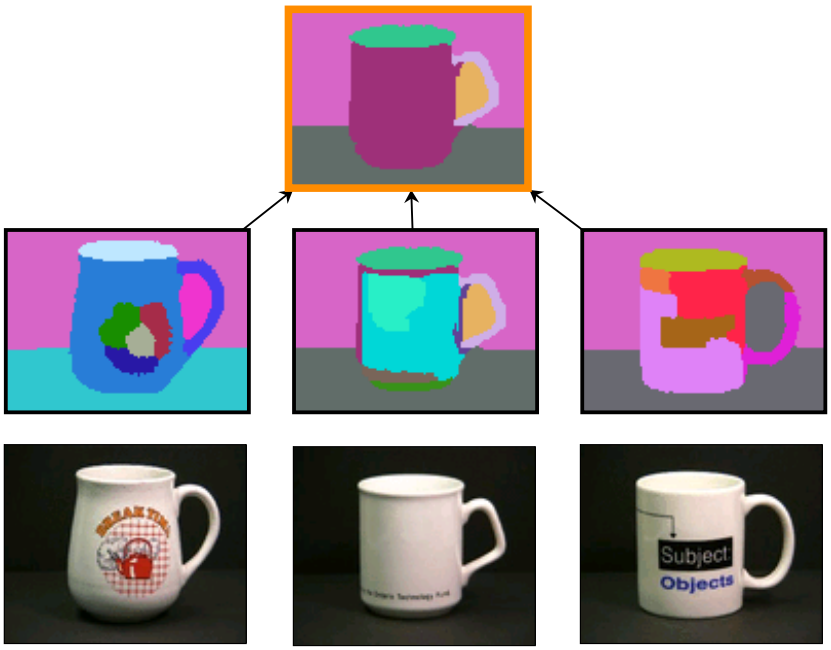


Fig. 8.3. Computed LCA (orange border) of three examples

8.3 Graph Matching Using a Spectral Encoding of Graph Structure

We now turn to the problem of inexact graph matching where, due to noise and occlusion, two graphs representing very similar objects may not be isomorphic. In a coarse-to-fine feature hierarchy, we'd like to find correspondences in a coarse-to-fine fashion, requiring that we have some way of compactly, robustly, and efficiently describing the underlying structure rooted at a node. Given such a low-dimensional encoding of DAG structure, we would compare the "shapes" of two graphs by comparing their encodings. Although correspondence may be established between two nodes in two graphs, according to the similarity of their encodings, it is important to note that these encodings describe the *entire* underlying structures rooted at these two nodes. Since these structures may have different numbers of nodes and may have slight variation in branching structure, the resulting matching can be seen as many-to-many, effectively matching two graph abstractions.

8.3.1 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 an antisymmetric $\{0, 1, -1\}$ node-adjacency matrix (which we will subsequently refer to as an 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 the adjacency matrix A are invariant to any orthonormal transformation of the form $P^t AP$. Since a permutation matrix is orthonormal, the eigenvalues of a graph are invariant to any consistent reordering of the graph's branches. In recent work [421], we established the stability of a graph's eigenvalues to minor perturbation due to noise and occlusion.

8.3.2 Formulating an Index

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 $T_1, T_2, \ldots, T_S$, as shown in Figure 8.4. For each subgraph, T_i, whose root degree is $\delta(T_i)$, we compute the eigenvalues of T_i's submatrix, sort them in decreasing order by absolute value of their magnitude, and let S_i be the sum of the $\delta(T_i) - 1$ largest magnitudes. The sorted S_i's become the components of a $\Delta(T)$-dimensional vector, called a *Topological Signature Vector, or TSV*, 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 TSV 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.

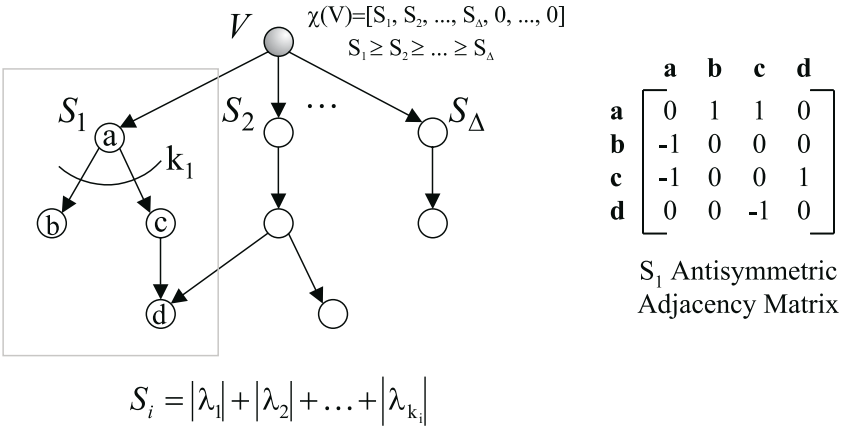


Fig. 8.4. Forming a low-dimensional vector description (TSV) of graph structure

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.

8.3.3 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 (a DAG that encodes the graph structure of an object and an occluder, i.e. portions of the DAG may not represent the target object but that of an occluder), subject to hierarchical constraints.

Our algorithm, a modification to Reyner's algorithm [371], 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 (TSV) 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.

8.3.4 Demonstration

To demonstrate our approach to matching, we turn to the domain of 2-D object recognition [422, 424]. 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 [223]. 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 8.5 illustrates a 2-D shape, its shock structure, and its resulting shock graph, while Figures 8.6 illustrates a matching example of two similar shapes, showing part correspondences. Extensive examples and performance evaluation can be found in [424, 271, 270].

8.4 Many-to-Many Graph Matching Using Graph Embedding

Several existing approaches to the problem of many-to-many graph matching suffer from computational inefficiency and/or from an inability to handle small perturbations in graph structure. We seek a solution to this problem while addressing drawbacks of existing approaches. Drawing on recently-developed techniques from the domain of low-distortion graph embedding, we have explored an efficient method for mapping a graph's structure to a set of vectors in a low-dimensional space. This mapping not only simplifies the original graph representation, but it retains important information about both local (neighborhood) as well as global graph structure. Moreover, the mapping is stable with respect to noise in the graph structure.

Armed with a low-dimensional, robust vector representation of an input graph's structure, many-to-many graph matching can now be reduced to the much simpler problem of matching weighted distributions of points in a normed vector space, using a *distribution-based* similarity measure. We consider one such similarity measure, known as the Earth Mover's Distance (EMD), and show that the many-to-many vector mapping that realizes the minimum Earth Mover's Distance corresponds to the desired many-to-many matching between nodes of the original graphs. The result is a more efficient

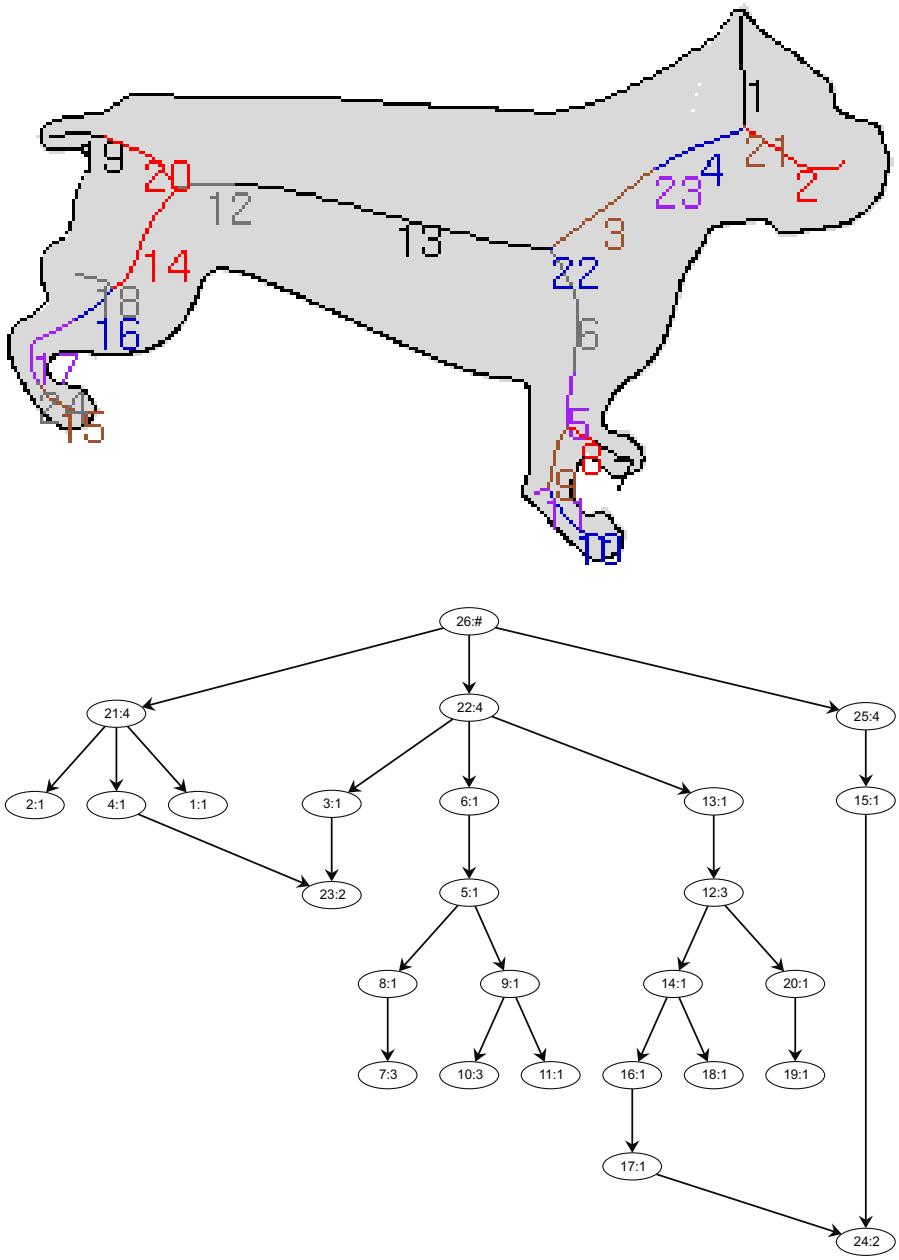


Fig. 8.5. An illustrative example of a shape and its corresponding shock graph

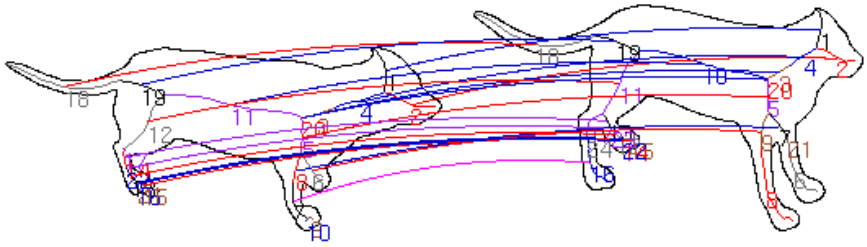


Fig. 8.6. Example part correspondences computed by the matching algorithm

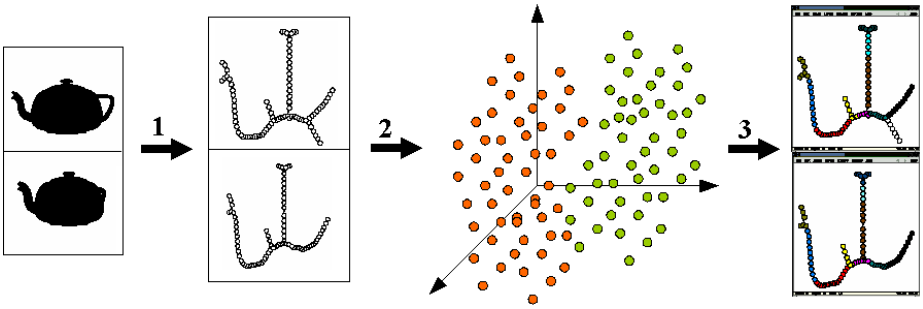


Fig. 8.7. Overview of many-to-many matching procedure. a pair of views are represented by undirected, rooted, weighted graphs (transition 1). The graphs are mapped into a low-dimensional vector space using a low-distortion graph embedding (transition 2). A many-to-many point (graph node) correspondence is computed by the earth mover's distance under transformation (transition 3).

and more stable approach to many-to-many graph matching that, in fact, includes the special case of one-to-one graph matching. The overview of the approach is presented in Figure 8.7.

8.4.1 Low-Distortion Embedding

Our interest in low-distortion embedding is motivated by its ability to transform the problem of many-to-many matching in finite graphs to geometrical problems in low-dimensional vector spaces. Specifically, let $G_1 = (\mathcal{A}_1, E_1, \mathcal{D}_1)$, $G_2 = (\mathcal{A}_2, E_2, \mathcal{D}_2)$ denote two graphs on vertex sets $\mathcal{A}_1$ and $\mathcal{A}_2$, edge sets E_1 and E_2, under distance metrics $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively ($\mathcal{D}_i$ represents the distances between all pairs of nodes in G_i). Ideally, we seek a single embedding that can map each graph to the same vector space, in which the two embeddings can be directly compared. However, in general, this is not possible without introducing unacceptable distortion.

We will therefore tackle the problem in two steps. First, we will seek low-distortion embeddings f_i that map sets $\mathcal{A}_i$ to normed spaces $(\mathcal{B}_i, ||.||_k)$, $i \in \{1, 2\}$. Next, we will align the normed spaces, so that the embeddings can be directly compared. Using

these mappings, the problem of many-to-many vertex matching between G_1 and G_2 is therefore reduced to that of computing a mapping $\mathcal{M}$ between subsets of $\mathcal{B}_1$ and $\mathcal{B}_2$.

In practice, the robustness and efficiency of mapping $\mathcal{M}$ will depend on several parameters, such as the magnitudes of distortion of the $\mathcal{D}_i$'s under the embeddings, the computational complexity of applying the embeddings, the efficiency of computing the actual correspondences (including alignment) between subsets of $\mathcal{B}_1$ and $\mathcal{B}_2$, and the quality of the computed correspondence. The latter issue will be addressed in Section 8.4.2.

The problem of low-distortion embedding has a long history for the case of planar graphs, in general, and trees, in particular. More formally, the most desired embedding is the subject of the following conjecture:

Conjecture 1. [162] Let $G = (\mathcal{A}, E)$ be a planar graph, and let $M = (\mathcal{A}, \mathcal{D})$ be the shortest-path metric for the graph G. Then there is an embedding of M into $||.||_p$ with $O(1)$ distortion.

This conjecture has only been proven for the case in which G is a tree. Although the existence of such a distortion-free embedding under $||.||_k$-norms was established in [258], no deterministic construction was provided. One such deterministic construction was given by Matoušek [287], suggesting that if we could somehow map our graphs into trees, with small distortion, we could adopt Matoušek's framework.

The Shortest Path Metric

Before we can proceed with Matoušek's embedding, we must choose a suitable metric for our graphs, i.e., we must define a distance between any two vertices. Let $G = (\mathcal{A}, E)$ denote an edge-weighted graph with real edge weights $\mathcal{W}(e)$, $e \in E$. We will say that $\mathcal{D}$ is a metric for G if, for any three vertices $u, v, w \in \mathcal{A}$, $\mathcal{D}(u, v) = \mathcal{D}(v, u) \geq 0$, $\mathcal{D}(u, u) = 0$, and $\mathcal{D}(u, v) \leq \mathcal{D}(u, w) + \mathcal{D}(w, v)$. In general, there are many ways to define metric distances on a weighted graph. The best-known metric is the shortest-path metric $\delta(.,.)$, i.e., $\mathcal{D}(u, v) = \delta(u, v)$, the shortest path distance between u and v for all $u, v \in \mathcal{A}$. In fact, if the weighted graph G is a tree, the shortest path between any two vertices is unique, and the weights of the shortest paths between vertices will define a metric $\mathcal{D}(.,.)$.

In the event that G is not a tree, $\mathcal{D}(.,.)$ can be defined through a special representation of G, known as the *centroid metric tree* $\mathfrak{T}$ [2]. The path-length between any two vertices u, v in $\mathfrak{T}$ will mimic the metric $\delta(u, v)$ in G. A metric $\mathcal{D}(.,.)$ on n objects $\{v_1, \ldots, v_n\}$ is a centroid metric if there exist labels $\ell_1, \ldots, \ell_n$ such that for all $i \neq j, \mathcal{D}(v_i, v_j) = \ell_i + \ell_j$. If G is not a tree, its centroid metric tree $\mathfrak{T}$ is a star on vertex-set $\mathcal{A} \cup \{c\}$ and weighted edge-set $\{(c, v_i) |\, \mathcal{W}(c, v_i) = \ell_i,\ v_i \in \mathcal{A}\}$. It is easy to see that the path-lengths between v_i and v_j in $\mathfrak{T}$ will correspond to $\mathcal{D}(v_i, v_j)$ in G. For details on the construction of a metric labeling ℓ_i of a metric distance $\mathcal{D}(.,.)$, see [2].

Path Partition of a Graph

The construction of the embedding depends on the notion of a path partition of a graph. In this subsection, we introduce the path partition, and then use it in the next subsection

to construct the embedding. Given a weighted graph $G = (\mathcal{A}, E)$ with metric distance $\mathcal{D}(.,.)$, let $\mathfrak{T} = (\mathcal{A}, \mathfrak{E})$ denote a tree representation of G, whose vertex distances are consistent with $\mathcal{D}(.,.)$. In the event that G is a tree, $\mathfrak{T} = G$; otherwise $\mathfrak{T}$ is the centroid metric tree of G. To construct the embedding, we will assume that $\mathfrak{T}$ is a rooted tree. It will be clear from the construction that the choice of the root does not affect the distortion of the embedding.

The dimensionality of the embedding of $\mathfrak{T}$ depends on the caterpillar dimension, denoted by $\mathrm{cdim}(\mathfrak{T})$, and is recursively defined as follows [287]. If $\mathfrak{T}$ consists of a single vertex, we set $\mathrm{cdim}(\mathfrak{T}) = 0$. For a tree $\mathfrak{T}$ with at least 2 vertices, $\mathrm{cdim}(\mathfrak{T}) \leq k + 1$ if there exist paths $P_1, \ldots, P_r$ beginning at the root and otherwise pairwise disjoint, such that each component $\mathfrak{T}_j$ of $\mathfrak{T} - \mathfrak{E}(P_1) - \mathfrak{E}(P_2) - \cdots - \mathfrak{E}(P_r)$ satisfies $\mathrm{cdim}(\mathfrak{T}_j) \leq k$. Here $\mathfrak{T} - \mathfrak{E}(P_1) - \mathfrak{E}(P_2) - \cdots - \mathfrak{E}(P_r)$ denotes the tree $\mathfrak{T}$ with the edges of the P_i's removed, and the components $\mathfrak{T}_j$ are rooted at the single vertex lying on some P_i. The caterpillar dimension can be determined in linear time for a rooted tree $\mathfrak{T}$, and it is known that $\mathrm{cdim}(\mathfrak{T}) \leq \log(|\mathcal{A}|)$ (see [287]).

The construction of vectors $f(v)$, for $v \in \mathcal{A}$, depends on the notion of a *path partition* of $\mathfrak{T}$. The path partition $\mathfrak{P}$ of $\mathfrak{T}$ is empty if $\mathfrak{P}$ is a single vertex; otherwise $\mathfrak{P}$ consists of some paths $P_1, \ldots, P_r$ as in the definition of $\mathrm{cdim}(\mathfrak{T})$, plus the union of path partitions of the components of $\mathfrak{T} - \mathfrak{E}(P_1) - \mathcal{M}(P_2) - \cdots - \mathfrak{E}(P_r)$. The paths $P_1, \ldots, P_r$ have level 1, and the paths of level $k \geq 2$ are the paths of level $k - 1$ in the corresponding path partitions of the components of $\mathfrak{T} - \mathfrak{E}(P_1) - \mathfrak{E}(P_2) - \cdots - \mathfrak{E}(P_r)$. Note that the paths in a path partition are edge-disjoint, and their union covers the edge-set of $\mathfrak{T}$.

To illustrate these concepts, consider the tree shown in Figure 8.8. The three darkened paths from the root represent three level 1 paths. Following the removal of the level 1 paths, we are left with 6 connected components that, in turn, induce seven level 2 paths, shown with lightened edges.[6] Following the removal of the seven level 2 paths, we are left with an empty graph. Hence, the caterpillar dimension ($\mathrm{cdim}(\mathfrak{T})$) is 2. It is easy to see that the path partition $\mathfrak{P}$ can be constructed using a modified depth-first search in $O(|\mathcal{A}|)$ time.

Construction of the Embedding

Given a path partition $\mathfrak{P}$ of $\mathfrak{T}$, we will use m to denote the number of levels in $\mathfrak{P}$, and let $P(v)$ represent the unique path between the root and a vertex $v \in \mathcal{A}$. The first segment of $P(v)$ of weight l_1 follows some path P^1 of level 1 in $\mathfrak{P}$, the second segment of weight l_2 follows a path P^2 of level 2, and the last segment of weight l_α follows a path P^α of level $\alpha \leq m$. The sequences $\langle P^1, \ldots, P^\alpha \rangle$ and $\langle l_1, \ldots, l_\alpha \rangle$ will be referred to as the *decomposition sequence* and the *weight sequence* of $P(v)$, respectively.

To define the embedding $f : \mathcal{A} \to \mathcal{B}$ under $||.||_2$, we let the relevant coordinates in $\mathcal{B}$ be indexed by the paths in $\mathfrak{P}$. The vector $f(v)$, $v \in \mathcal{A}$, has non-zero coordinates corresponding to the paths in the decomposition sequence of $P(v)$. Returning to Figure 8.8, the vector $f(v)$ will have 10 components (defined by three level 1 paths and seven level 2 paths). Furthermore, every vector $f(v)$ will have at most two non-zero components.

[6] Note that the third node from the root in the middle level 1 branch is the root of a tree-component consisting of five nodes that will generate two level 2 paths.

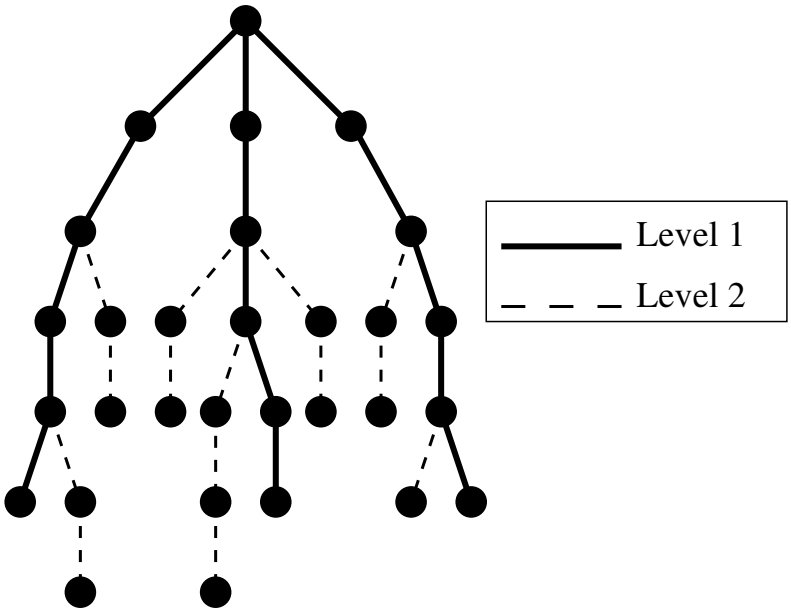


Fig. 8.8. Path partition of a tree

Consider, for example, the lowest leaf node in the middle branch. Its path to the root will traverse three level 2 edges corresponding to the fourth level 2 path, as well as three level 1 edges corresponding to the second level 1 path.

Such embedding functions have become fairly standard in the metric space representation of weighted graphs [259, 287]. In fact, Matoušek [287] has proven that setting the i-th coordinate of $f(v)$, corresponding to path P^k, $1 \leq k \leq \alpha$, in decomposition sequence $\langle P^1, \ldots, P^\alpha \rangle$, to

$$f(v)_i = \sqrt{l_k \left[l_k + \sum_{j=1}^{\alpha} \max\left(0, l_j - l_k/2m\right) \right]}$$

will result in a small distortion of at most $\sqrt{\log \log |\mathcal{A}|}$. It should be mentioned that although the choice of path decomposition $\mathfrak{P}$ is not unique, the resulting embeddings are isomorphic up to a transformation. Computationally, constructions of $\mathfrak{T}$, $\mathfrak{P}$, and $\mathcal{B}$ are all linear in terms of $|\mathcal{A}|$ and $|\mathfrak{E}|$.

The above embedding has preserved both graph structure and edge weights, but has not accounted for node information. To accommodate node information in our embedding, we will associate a weight w_v to each vector $f(v)$, for all $v \in \mathcal{A}$. These weights will be defined in terms of vertex labels which, in turn, encode image feature values. Note that nodes with multiple feature values give rise to a vector of weights assigned to every point. We will present an example of one such distribution in Section 8.4.3.

8.4.2 Distribution-Based Matching

By embedding vertex-labeled graphs into normed spaces, we have reduced the problem of many-to-many matching of graphs to that of many-to-many matching of weighted distributions of points in normed spaces. However, before we can match two point distributions, we must map them into the same normed space. This involves reducing the dimensionality of the higher-dimensional distribution and aligning the two distributions. Given a pair of weighted distributions in the same normed space, the Earth Mover's Distance (EMD) framework [384] is then applied to find an optimal match between the distributions. The EMD approach computes the minimum amount of work (defined in terms of displacements of the masses associated with points) it takes to transform one distribution into another.

Embedding Point Distributions in the Same Normed Space

Embeddings produced by the graph embedding algorithm can be of different dimensions and are defined only up to a distance-preserving transformation (a translated and rotated version of a graph embedding will also be a graph embedding). Therefore, in order to apply the EMD framework, we perform a PCA-based "registration" step, whose objective is to project the two distributions into the same normed space. Intuitively, the projection of the original vectors associated with each graph embedding onto the subspace spanned by the first K right singular vectors of the covariance matrix retains the maximum information about the original vectors among all projections onto subspaces of dimension K. Hence, if K is the minimum of the two vector dimensions, projecting the two embeddings onto the first K right singular vectors of their covariance matrices will equalize their dimensions while losing minimal information [221]. The resulting transformation is expected to minimize the initial EMD between the distributions.

The Earth Mover's Distance

The Earth Mover's Distance (EMD) [384, 74] is designed to evaluate the dissimilarity between two multi-dimensional distributions in some feature space. The EMD approach assumes that a distance measure between single features, called the *ground distance*, is given. The EMD then "lifts" this distance from individual features to full distributions. Computing the EMD is based on a solution to the well-known *transportation problem* [4], whose optimal value determines the minimum amount of "work" required to transform one distribution into the other. Moreover, if the weights of the distributions are the same, and the ground distance is a metric, EMD induces a metric distance [384].

Recall that a translated and rotated version of a graph embedding will also be a graph embedding. To accommodate pairs of distributions that are "not rigidly embedded", Cohen and Guibas [74] extended the definition of EMD, originally applicable to pairs of fixed sets of points, to allow one of the sets to undergo a transformation. They also suggested an iterative process (which they call **FT**, short for "an optimal **F**low and an optimal **T**ransformation") that achieves an infimum of the objective function for EMD. The iterative process stops when the improvement in the objective function value falls below a threshold. For our application, the set of allowable transformations consists

of only those transformations that preserve distances. Therefore, we use a weighted version of the Least Squares Estimation algorithm [464] to compute an optimal distance-preserving transformation given a flow between the distributions. The main advantage of using EMD lies in the fact that it subsumes many histogram distances and permits partial and many-to-many matches under transformation in a natural way. This important property allows the similarity measure to deal with uneven clusters and noisy datasets.

8.4.3 Demonstration

To demonstrate our approach to many-to-many matching, we turn to the domain of view-based object recognition using silhouettes. For a given view, an object's silhouette is first represented by an undirected, rooted, weighted graph, in which nodes represent *shocks* [424] (or, equivalently, skeleton points) and edges connect adjacent shock points.[7] We will assume that each point p on the discrete skeleton is labeled by a 4-dimensional vector $v(p) = (x, y, r, \alpha)$, where (x, y) are the Euclidean coordinates of the point, r is the radius of the maximal bi-tangent circle centered at the point, and α is the angle between the normal to either bitangent and the linear approximation to the skeleton curve at the point.[8] This 4-tuple can be thought of as encoding local shape information of the silhouette.

To convert our shock graphs to shock trees, we compute the minimum spanning tree of the weighted shock graph. Since the edges of the shock graph are weighted based on Euclidean distances of corresponding nodes, the minimum spanning tree will generate a suitable tree approximation for shock graphs. The root of the tree is the node that minimizes the sum of distances to all other nodes. Finally, each node is weighted proportionally to its average radius, with the total tree weight being 1. An illustration of the procedure is given in Figure 8.9. The left portion shows the initial silhouette and its shock points (skeleton). The right portion depicts the constructed shock tree. Darker, heavier nodes correspond to fragments whose average radii are larger. Figure 8.10 illustrates the many-to-many correspondences that our matching algorithm yields for two adjacent views (30° and 40°) of the TEAPOT. Corresponding clusters (many-to-many mappings) have been shaded with the same color. Note that the extraneous branch in the left view was not matched in the right view, reflecting the method's ability to deal with noise. Further details and additional experiments can be found in [221].

8.5 Conclusions

Successful object recognition requires overcoming the restrictive assumption of one-to-one feature correspondence. We introduce three different graph theoretic frameworks for many-to-many feature matching. The first, addressing the problem of generic model acquisition, searches the space of all possible abstractions of an image, and uses support

[7] Note that this representation is closely related to Siddiqi et al.'s *shock graph* [424], except that our nodes (shock points) are neither clustered nor are our edges directed.

[8] Note that this 4-tuple is slightly different from Siddiqi et al.'s shock point 4-tuple, where the latter's radius is assumed normal to the axis.

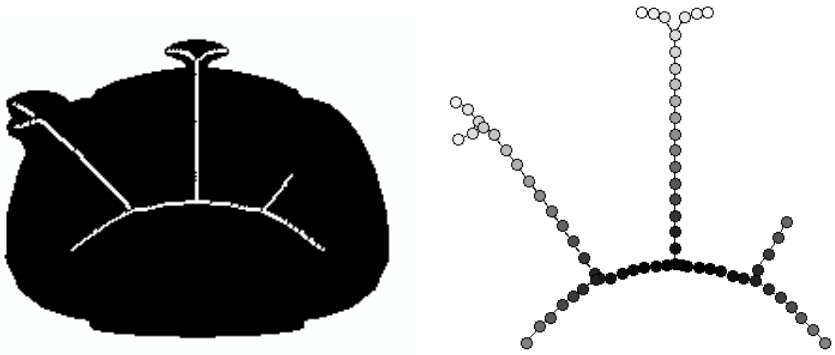


Fig. 8.9. Left: The silhouette of a TEAPOT and its medial axis. Right: the medial axis tree constructed from the medial axis. darker nodes reflect larger radii.

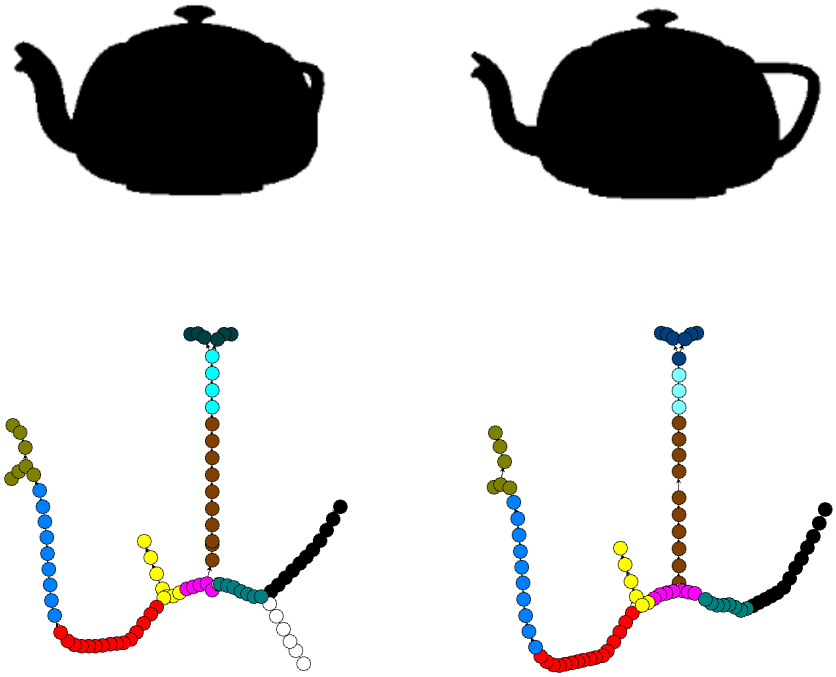


Fig. 8.10. Illustration of the many-to-many correspondences computed for two adjacent views of the TEAPOT. Matched point clusters are shaded with the same color.

from other objects drawn from the same class to help constrain the search. The second, addressing the problem of graph matching, draws on spectral graph theory to generate and match abstractions of directed acyclic graphs. The third, addressing the problem of many-to-many graph matching, attempts to map the graphs into a geometric domain, in which effective algorithms for many-to-many point matching exist. In current work, we are pushing each of these three fronts in order to improve matching in the presence of increased noise, occlusion, articulation, and deformation.

Acknowledgements

The authors gratefully acknowledge the support of NSERC, IRIS, NSF, ONR, and PREA.