Shape matching using edit-distance: an implementation

Philip N. Klein* Thomas B. Sebastian† Benjamin B. Kimia ‡

Abstract

We report on our experience with the implementation of an algorithm for comparing shapes by computing the edit-distance between their medial axes. A shape-comparison method that is robust to various visual transformations has several applications in computer vision, including organizing and querying an image database, and object recognition.

There are two components to research on this problem, mathematical formulation of the shape-comparison problem and the computational solution method. We have a clear, well-defined formulation and polynomial-time algorithms for solution. Previous research has involved either ill-defined formulations or heuristic methods for solution.

Our starting-point for the implementation is the edit-distance algorithm of Klein et al. [6]. We discuss how we altered that algorithm to handle rotation-invariance while keeping down the time and storage requirements. Most important, we define costs for the edit-operations and give an algorithm for computing them.

We use a database of shapes to illustrate that our approach performs intuitively in categorization and indexing tasks, and our results are better than previous approaches.

1 Introduction

In this paper, we report on implementation of and experience with an algorithmic approach to comparing two-dimensional shapes by computing the edit-distance between their skeletons. Having a quickly computable metric for shapes that accords with visual intuition is useful in several applications. It can be used in organizing and querying a database of images. It is useful in object recognition and other computer-vision problems. It can provide the basis for measurement and comparison of medical structures by shape, e.g. comparing a patient's bone's shape to the normal and quantifying the degree of difference.

There are three elements to our metric, shock graphs, modified edit-distance, and boundary segment alignment.

1.1 Shock graphs

We represent shapes by their *shock graphs*. The shock graph of a shape is the medial axis, locus of centers of circles that are at least bitangent to the boundary, endowed with geometric and dynamics information.

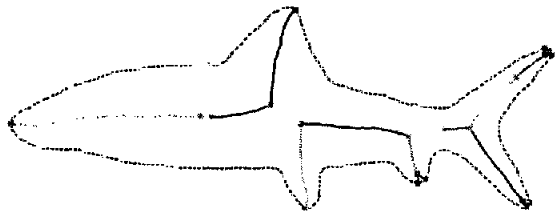

Figure 1: A shape (a bonefish) and its medial axis organized as a shock graph.

The distance to the boundary (radius of the circle) at each shock point can be interpreted as the time of formation of the shock in a wave propagation scheme where waves are send from the boundary. This dynamic interpretation of the shock trajectory allows us to associate a direction of flow, and instantaneous velocity (derivative with respect to radius) to each shock point [5, 11]. See Figure 1 for an example of a shock graph.

Shock graphs are a richer descriptor of shape than the boundary by itself. Shock graphs encode information about the interior of the shape by pairing shape boundary segments. Moreover, shock graphs capture the hierarchical relationships between parts of a shape, making it less sensitive to occlusion, articulation and other visual transformations.

1.2 Modified edit-distance

Our approach to comparing shapes involves computing the edit-distance between their shock graphs. While definitions of edit-distance for trees have appeared in the literature, we gave [6] a definition that was appropriate to our application. Traditional edit-distance would perhaps be defined by three operations, (i) *change* the label of an edge, (ii) *contract* an edge, and (iii) the inverse operation *uncontract* an edge. Given a method of assigning costs to instances of these operations, the edit-distance between two trees is defined as the minimum cost of a set of operations to transform one tree into another.

The idea of using edit-distance to compare shapes is based on observing discrete changes in the shock graph as a shape morphed to another. The shock graph topology in general remains the same for finite extents of deformation, but then experiences a *transition*, as illustrated in Figure 3. The set of these discrete

*Dept. of Computer Science, Brown University, Providence, research supported by NSF Grant CCR-9700146.

†Div. of Engineering, Brown University, Providence.

‡Div. of Engineering, Brown University, Providence.

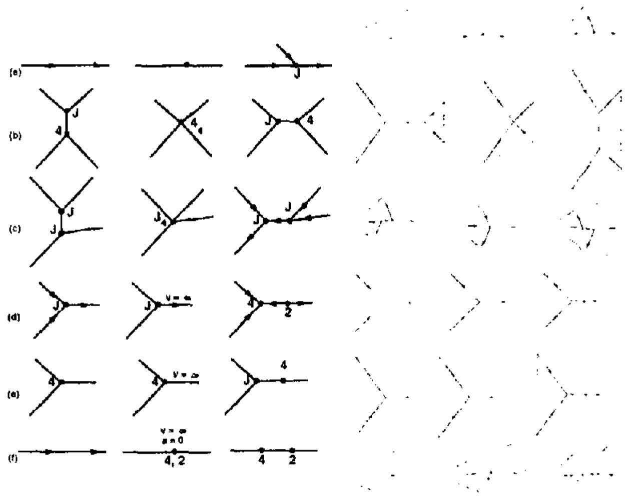
Figure 2: The first three columns show a schematic description of the six possible transitions of shock structure, while the last three columns illustrate corresponding examples of shape deformations. The graph operations to make the right and left columns equivalent are: splice, contract (two types), and merge (three types).

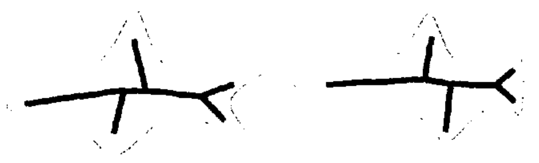
Figure 3: The fish are similar but their shock graphs are topologically different. However, a contraction of the middlemost edge followed by an uncontraction can turn one shock graph into the other.

transitions identify a continuous deformation sequence. Since the similarity between two shapes can be viewed as the extent of deformation needed to deform one shape to the another, similarity can be measured by summing up the cost of deformations between the shock transitions.

The shock transitions have been completely classified in [3] and displayed in Figure 2. Since each shock transition, should correspond to an edit operation, the traditional edit operations are not sufficient. A small protrusion in the boundary can result in new edges and nodes that "interrupt" an edge; this affects matching since every point along the interrupted edge carries geometric information. As shown in Figure 4, this phenomenon necessitates a *merge* operation that removes one of the edges incident to a node and joins the remaining two to form a single, longer edge. In [6], we gave a tree edit-distance algorithm that incorporated the *merge* operation as well as some technical restrictions arising from the application.

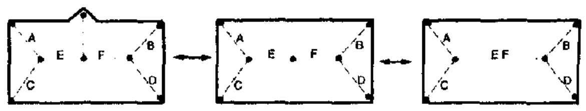

Figure 4: The hooded rectangle can be transformed into the unhooded rectangle by splicing off the hood-edge, and merging the E edge and the F edge into a single EF edge.

1.3 Enhancements to the edit-distance algorithm

We needed to make several enhancements to the algorithm of [6] to make it useful. Most important, that algorithm was designed for comparing *rooted* trees; in order that our shape metric be rotation-invariant, we needed to compare *unrooted* trees. One could fix the root of one tree and then try all possible roots of the other, but that would multiply the time required by $O(n)$ where n is the tree size. We take a slightly more sophisticated approach that increases the running time by a sublinear factor. We combine this with use of a heuristic due to Zhang and Shasha [15] for reducing the running time for rooted trees.

The second issue is storage requirements. Zhang and Shasha give a technique for keeping the storage requirement $O(n^2)$, and the algorithm of [6] can make use of this technique. However, our technique for handling unrooted trees seems to conflict with the storage-saving technique. We show that by a careful ordering of subproblems we can combine the two techniques.

1.4 Assignment of costs

One key issue not addressed in [6] was assignment of costs to edit operations. Naive costs (e.g. unit costs for all operations) results in terrible match quality. In this paper, we discuss a definition of costs and a dynamic-programming algorithm for computing costs that result in high-quality matches. Our approach builds on experience [9] with a metric for comparing curves. Conceptually, the approach to comparing two curves consists of minimizing a functional over all possible *alignments* between the two curves; an alignment is essentially a pairing of points on the two curves. (See Figure 6.)

Our goal here is to extend this idea to matching two shock-graph edges. A shock-graph edge represents not one curve but two (the shape-boundary segments on opposite sides of the edge, as shown in Figure 5).The notion of alignment extends in a natural way; an alignment is a pairing of points on one edge with points on the other. Such an alignment induces a pair of simultaneous alignments, an alignment between the left boundary segment of one edge and the left boundary segment of the other edge, and an alignment between the right boundary segments. One might think it is sufficient to choose the edge-alignment that minimizes

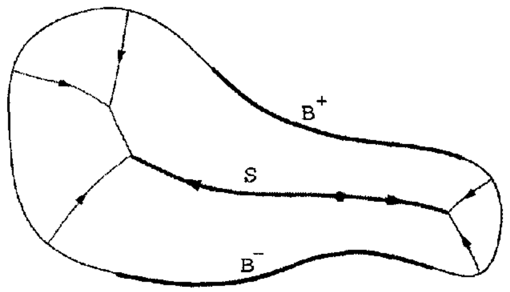

Figure 5: The shock $\mathcal{S}$ and the corresponding shape boundary segments $\mathcal{B}^+$ and $\mathcal{B}^-$ are depicted as thick curves.

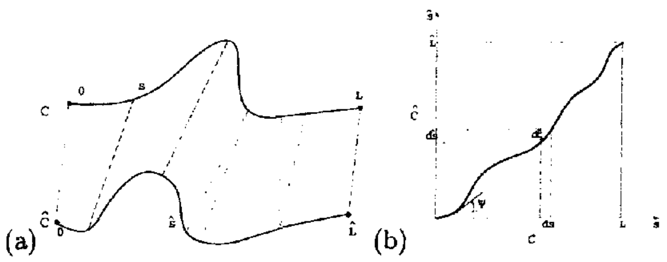

Figure 6: The alignment of two curves C and $\hat{C}$ is represented by the pairing of points on the two curves (a). For a symmetric treatment this pairing is expressed via the alignment curve (b).

cost of the induced boundary-segment alignments. This, however, fails to take into account the distance of the boundary segments from the shock-graph edges and their relative orientations; one must add additional terms for this reason.

1.5 Results

We illustrate the applicability of the proposed algorithm to categorization and indexing tasks on a shape database. The experimental results are visually intuitive. Moreover, our indexing results are better than those presented in [10, 2, 1], when compared on a common database of shapes. Note that the focus of this paper is on the quality of the results, and not on the running time of the current implementation, which we believe that we can significantly improved.

1.6 Previous work in using skeleton-matching for shape comparison

There has been extensive work done in the area of object recognition and shape matching. For a recent review see [14]. In this paper, we will mainly focus on approaches that match the skeletal graphs.

Sharvit *et al.* [10] have used the graduated assignment algorithm [4] to match shock graphs. The approach enforces only pairwise consistency of nodes, and the match process does not necessarily preserve coherence of shapes. Siddiqi *et al.* [12, 8] use subgraph isomorphism to match skeletal trees. Zhu and Yuille [16] have proposed FORMS for matching animate shapes by comparing their skeletal graphs using a branch and bound strategy. These approaches do not model the instabilities of the shock representation explicitly, which is problematic when dealing with visual transformations like occlusion, view point variation, and articulation. Liu and Geiger [7] use A* algorithm to match shape axis trees, which are computed by finding a correspondence between the shape outline and it mirror image. Their algorithm does not preserve ordering of edges at nodes which can result in matches that do not preserve coherence of the shapes.

1.7 Overview

Our shape-comparison code consists of two phases. In the first phase, we compute the costs of all possible edit operations. The main goal here is to compare each path in one shock graph to each path in the other, computing a cost of deforming one to the other. This phase is described in Section 2. In the second phase, we use these costs in computing the edit-distance. This phase is described in Section 3. We summarize our experimental results in Section 4. The appendix contains figures and tables summarizing our results.

2 Cost of Edit Operations

This section describes how we compute the cost of deforming an edge in a shock graph to an edge in the shock graph to be matched, and penalize the various edit operations. We first define the deform cost for a pair of edges. The edit operations can then be viewed as special cases of deform and the cost accordingly computed.

Observe that each edge in the shock graph represents a pair of segments on the boundary of the shape, as shown in Figure 5. Hence, we can view the problem of deforming an edge in a shock graph to an edge in another shock graph in terms of deforming the corresponding boundary segments, *i.e.*, as a "joint curve matching problem", which we tackle by extending our curve matching framework [9].

2.1 Curve alignment

We now briefly summarize how we match a pair of curves *i.e.*, how we find the optimal alignment and deformation cost for a pair of curves is found. For details, see [9]. Let the curves to be matched, C and $\hat{C}$, be parameterized by arclength s and $\hat{s}$, respectively, as shown in Figure 6a. Let $\theta(s)$ and $\hat{\theta}(\hat{s})$ be the orientation of the two curves. The basic premise of the approach is that the cost of matching two curves

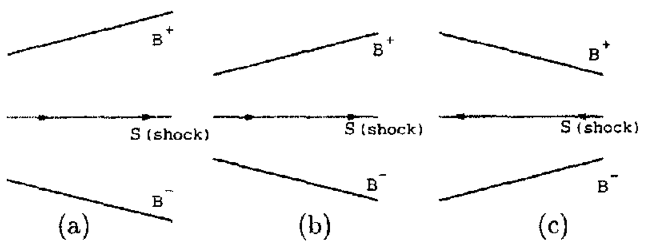

Figure 7: This figure illustrates why joint curve matching of shape boundary segment pairs is not just matching the boundary segments independently. In all three cases (a), (b) and (c) the shape boundary segments are identical up to rotations and translations. Hence, the cost of matching them independently is zero. However, the cost of matching the pair as a whole should not be zero, as (*i*)the segments in (a) are farther apart than those in (b) suggesting that shape in (a) is wider than shape in (b), and (*ii*) the segments in (b) are widening whereas those in (c) are narrowing. Hence, in addition to stretching and bending of the individual boundary segments, we should penalize shape thinning by measuring the relative distances between the segments and relative orientation between the segments.

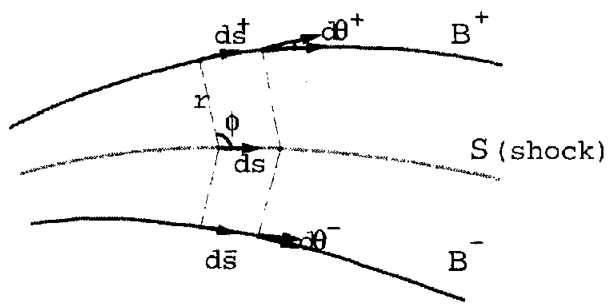

Figure 8: The shock $\mathcal{S}$ and the corresponding shape boundary segments $\mathcal{B}^+$ and $\mathcal{B}^-$.

can be expressed as the sum of the costs of matching infinitesimal subsegments, which is penalized by length and curvature differences. Specifically, we define the cost of matching infinitesimal segments on the two curves as $\mu = |d\hat{s} - ds| + R|d\hat{\theta} - d\theta|$, where R is a constant related to the average size of ds. (Note that curvature $\kappa = \frac{d\theta}{ds}$.) To ensure symmetric treatment of both curves, the alignment is represented as a pairing of points on the two curves. This leads to the notion of an *alignment curve* α, $\alpha(\xi) = (s(\xi), \hat{s}(\xi))$, which is defined by the corresponding points on the two curves, as shown in Figure 6b. Let the alignment curve be parameterized by ξ. Then, the optimal correspondence between the curves can be found by minimizing the functional

$$(2.1) \quad \mu[\alpha] = \int_0^{\tilde{L}} \left[\left| \frac{d\hat{s}}{d\xi} - \frac{ds}{d\xi} \right| + R \left| \frac{d\hat{\theta}(\hat{s})}{d\xi} - \frac{d\theta(s)}{d\xi} \right| \right] d\xi.$$

where $\tilde{L}$ is the length of the alignment curve. The first term in the functional penalizes "stretching" while the second term penalizes "bending". The optimal alignment is the one that minimizes the above functional, and the distance between the curves C and $\hat{C}$ is defined as the cost of the optimal alignment.

2.2 Shock segment alignment

The cost of matching edges of shock graphs is formulated as a joint curve matching problem of matching the corresponding boundary segment pairs. Hence, we penalize stretching and bending of the boundary segments as in the case of curves. However, for joint curve matching of shape boundary segment pairs, matching the boundary segment pairs independently is not enough. In addition to the stretching and bending terms, we also need to include shape thinning terms to penalize the relative distance between the two segments (width of the shape) and the relative orientation of the boundary segments. This is illustrated in Figure 7.

Let the edges of the shock graphs being matched, $\mathcal{S}$ and $\hat{\mathcal{S}}$, be parameterized by s and $\hat{s}$ respectively. Let the boundary segments corresponding to $\mathcal{S}$, $\mathcal{B}^+$ and $\mathcal{B}^-$ be parameterized by s^+ and s^-. Let $\theta^+(s^+)$ and $\theta^-(s^-)$ be orientations of $\mathcal{B}^+$ and $\mathcal{B}^-$, as shown in Figure 8. The boundary segments of $\hat{\mathcal{S}}$ are similarly defined. Consider the cost of matching infinitesimal segments of the shock edges. The stretching term penalizes the differences in lengths of the shape boundary segments and is given by $|d\hat{s}^+ - ds^+| + |d\hat{s}^- - ds^-|$. The bending term penalizes the differences in curvatures of the boundary segments and is given by $|d\hat{\theta}^+ - d\theta^+| + |d\hat{\theta}^- - d\theta^-|$. The relative distance between the two boundary segments is penalized by the differences in the width of the shape at the start node of the edge $|\hat{r}_0 - r_0|$, and the incremental difference at each corresponding shock segment and is given by $|d\hat{r} - dr|$. Similarly, the relative orientation differences are penalized initially by $|\hat{\phi}_0 - \phi_0|$ and incrementally by $|d\hat{\phi} - d\phi|$. Then, the cost of deforming the shock graph edges $\mathcal{S}$ and $\hat{\mathcal{S}}$ is computed by minimizing the functional defined on the alignment curve α parameterized by ξ,

$$(2.2) \quad \begin{aligned} \mu[\alpha] = &\int_0^{\tilde{L}} \left[\left| \tfrac{d\hat{s}^+}{d\xi} - \tfrac{ds^+}{d\xi} \right| + \left| \tfrac{d\hat{s}^-}{d\xi} - \tfrac{ds^-}{d\xi} \right| \right] d\xi \\ &+ \int_0^{\tilde{L}} \left[R\left(\left| \tfrac{d\hat{\theta}^+}{d\xi} - \tfrac{d\theta^+}{d\xi} \right| + \left| \tfrac{d\hat{\theta}^-}{d\xi} - \tfrac{d\theta^-}{d\xi} \right| \right) \right] d\xi \\ &+ \int_0^{\tilde{L}} \left| \tfrac{d\hat{r}}{d\xi} - \tfrac{dr}{d\xi} \right| d\xi + |\hat{r}_0 - r_0| \\ &+ \int_0^{\tilde{L}} R \left| \tfrac{d\hat{\phi}}{d\xi} - \tfrac{d\phi}{d\xi} \right| d\xi + R|\hat{\phi}_0 - \phi_0|, \end{aligned}$$

where $\tilde{L}$ is the length of the alignment curve. The optimal alignment α^* is the one that minimizes the functional, and the distance between the shock edges

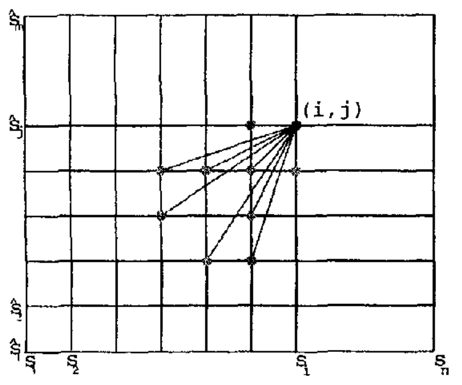

Figure 9: This figure illustrates the template that is used to in the Dynamic Programming implementation of Equation 2.2. The entry at (i,j) is the cost to match the curve segments $s_1, s_2, \ldots, s_i$ and $\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_j$ and denoted by $d(i,j)$. To update the cost at (i,j) (black dot) we limit the choices of the k and l in Equation 2.4, so that only costs at the a limited set of points (grey dots) are considered.

$\mathcal{S}$ and $\hat{\mathcal{S}}$ is defined as the cost of the optimal alignment,

$$d(\mathcal{S}, \hat{\mathcal{S}}) = \min_{\alpha} \mu[\alpha]. \tag{2.3}$$

2.3 Algorithm for shock edge alignment

To find the cost of deforming a pair of shock edges, we minimize the functional in Equation 2.2 by a dynamic-programming method. Let the edges of the shock graphs $\mathcal{S}$ and $\hat{\mathcal{S}}$ be discretized at samples $s_1, s_2, \ldots, s_n$ and $\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_m$, respectively. These are samples along the axes in Figure 9. Let $d(i,j)$ denote the cost of matching the curve segments $s_1, s_2, \ldots, s_i$ and $\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_j$ and let $\delta([k,i],[l,j])$ be the cost of matching the segments $s_k, \ldots, s_i$ and $\hat{s}_l, \ldots, \hat{s}_j$. Then, the distance defined in Equation 2.3 satisfies the following update rule

$$d(i,j) = \min_{k,l} \left[d(i-k, j-l) + \delta([i-k,i],[j-l,j]) \right]. \tag{2.4}$$

We limit the choices of the k and l, as a first approximation, to nine values achieved by using a small template as shown in Figure 9.

In order to compare two shock graphs, we need to compute the deform costs for every pair of paths in the two graphs. This is computationally expensive. However, we use the following heuristic to limit the number of path comparisons that have to be made. Observe that for the edit distance algorithm to consider matching a path in one shock graph to that in the other shock graph, all other edges incident on the intermediate nodes of the two paths have to spliced, as illustrated in Figure 10. If the cost of splicing such incident edges is high, we do not consider the case of deforming that particular pair of paths. In comparing all pairs of paths, there are overlapping subproblems, *i.e.*, when computing the deform cost for a pair of paths, we can use the computations done for the subpaths. Currently, we do not take advantage of this, and treat all pairs of paths independently. We plan to address this issue in later implementations.

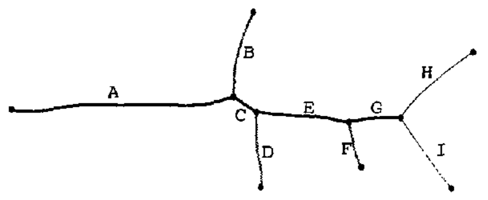

Figure 10: To consider the deform involving path consisting of edges ACEG, the other edges incident on the intermediate nodes (here B, D and F) have to spliced.

2.4 Cost of splice/contract operations

Now we show that splice and contract operations can be viewed as special cases of deforming a shock edge to another, and how their costs can derived from the deform cost. The *splice* operation prunes a leaf edge in the shock graph. In the shape domain, this corresponds to pruning out a protrusion, and replacing it with a circular arc, as shown in Figure 11a. The cost of a splice operation can be viewed as deforming the shape boundary segments corresponding to the protrusion to the circular arc resulting from the splice operation. The cost is derived from Equation 2.2 as follows: The first two terms reduce to $\left|\int ds^+ + \int ds^- - 2\phi r\right|$, the third and fourth terms to 0, the fifth and sixth terms to $2r$, and the last two terms to 0, thus giving

$$\mu_s = \left|\int ds^+ + \int ds^- - 2\phi r\right| + 2r. \tag{2.5}$$

The *contract* operation removes an edge between high degree nodes, as shown in Figure 11b. Unlike other shape edit operations, the contract operation is not a local operation and affects the shape as a whole. However, we can view the contract operation by itself, as deforming the contracted edge to nothing. In addition, the operation introduces a discontinuity in the shape if the width of the adjacent high degree nodes are not the same. This is penalized by the difference in widths at the adjacent high degree nodes. The contract cost is given by

$$\mu_c = \int ds^+ + \int ds^- + 2|r_1 - r_2| + R_1(r_1 + r_2) \tag{2.6}$$

Observe that the merge operation does not affect the topology of the shock graph. It merely changes the properties of the boundary segments, which is handled

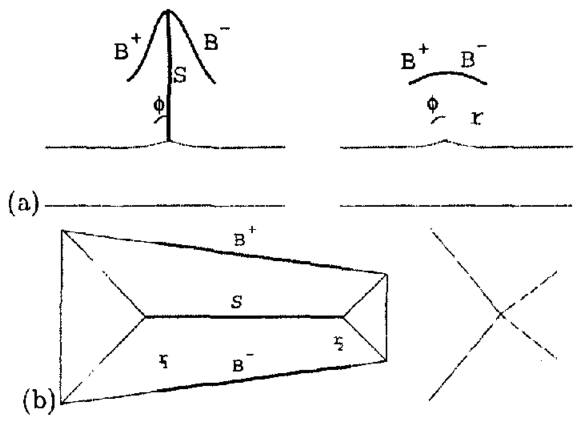

Figure 11: (a) Splice operation results in replacing a protrusion with a circular arc, thus the splice cost can be viewed as the cost of a deformation sequence which shrinks the protrusion away. (b) Contract operation.

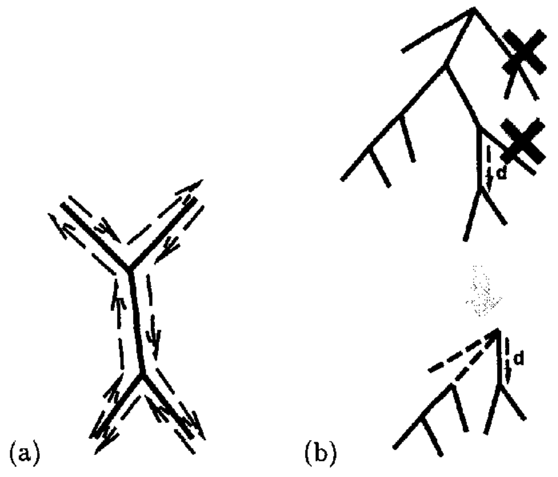

Figure 12: (a) A tree (bold) and its Euler tour (dashed arrows). (b) The upper figure is a tree T with a dart d. The lower figure is the modified tree $T[d]$, obtained by deleting branches to the right of d, designating as dashed the edges to the left of d, and contracting the edges from the root to d.

directly by the deform process. Thus, we do not need to additionally penalize merges.

3 The edit-distance algorithm

There are two edit operations, which modify the topology of the shock graph, *contract* and *splice*, plus their inverses. The first operation contracts an edge. The second operation operates on a node with three incident edges, one of which is a leaf edge. The operation removes the leaf edge and turns the remaining two edges into one edge, concatenating their geometric data. The final edit operation is *deform*, which alters the geometric data on an edge.[1]

Let v be a degree-three node of a tree T, and let e be an incident edge. Let T' be the subtree rooted at v containing e but no other edges incident to v. Starting at the leaves of T', the edges of T' can be spliced away until none is left, and the two other edges incident to v have become one. As a preprocessing step, the cost of carrying out these splices is computed and stored for all nodes v and incident edges e. The time and space required is linear.

3.1 Preliminaries

We start with some terminology and definitions. For any undirected graph G, there is a corresponding directed graph G' with the same nodeset: for each edge $\{u,v\}$ of G, there are two oppositely directed arcs uv and vu in G'. We refer to these arcs as the *darts* of G (corresponding to $\{u,v\}$). The *mate* of uv, written mate(uv), is vu, and vice versa. The head of uv is v, and the tail is u. If the graph is a rooted tree, we say a dart of the graph is *down-pointing* if the dart's head is farther from the root than the tail, and *up-pointing* otherwise.

We say a rooted tree is *ordered* if the tree is endowed with, for each node, a total ordering of the node's children. Similarly, an unrooted tree is ordered if the tree is endowed with, for each node v, a cycle $(v_1\ v_2\ \ldots\ v_k)$ of the neighbors of v. An ordering of an unrooted tree is a combinatorial representation of an embedding of the tree in the plane for such representation of embeddings of general planar graphs).

The *Euler tour* of an ordered tree[13] is a cycle consisting of all its darts, traversing the tree in, say, clockwise order (see Figure 12a). An *Euler string* of the tree is a consecutive subsequence of this cycle. For a dart d of a tree, let next(d) denote the next dart in the tree's Euler tour. We note that an Euler tour of a shock graph represents a traversal of the shape boundary.

Let T be a tree and d a dart of T. $T-d$ consists of two components, one containing the head of d and one containing the tail. Let $T(d)$ denote the component containing the head of d, rooted at that head.

For a rooted ordered tree T and a dart d of T, let $T[d]$ denote the tree obtained from T as follows. Let P be the path from the root of T to the tail of d. As illustrated in Figure 12b, delete each branch hanging off the right of P, designate as "dashed" each edge hanging

[1] In the paper [6], we described an additional operation, *merge*, which operates on a node with two incident edges and turns these two edges into one edge. However, because of the costs we adopt for this paper, it does no harm to carry out all possible merges in a preprocessing step. This rids the input trees of degree-two nodes. We therefore assume that the trees have no such nodes.

off the left of P, and then contract the edges of P.

3.2 The outer dynamic program

We use the algorithm described in [6], with several changes: some simplifications to take into account the cost assignment, a modification to handle unrooted trees, and some tricks for reducing the time and storage required.

The algorithm in [6] handled rooted, ordered trees. One approach to handling unrooted trees is to select a root r for one tree, say T_2 (the choice of root r is discussed later), and try all possible roots of T_1, outputting the minimum edit-distance found. It turns out that this would involve solving some subproblems several times; we can achieve the same effect without increasing the time as much by solving each subproblem once. In order to do so while keeping down the storage requirements, we need to choose carefully the order of subproblems solved.

We use an idea of Zhang and Shasha [15] to achieve $O(n^2)$ space. The idea is essentially to formulate the algorithm as a dynamic program each of whose steps is in turn implemented as a dynamic program. The outer dynamic program is as follows. For each dart b_1 of T_1 and each dart b_2 of T_2 that is down-pointing with respect to the root r, we have a table entry $M[b_1, b_2]$ assigned the rooted-tree edit-distance between $T_1(b_1)$ and $T_2(b_2)$. We refer to the subproblem of computing this distance as b_1-b_2 *subtree edit-distance.* There are $O(n_1 n_2)$ subproblems, where n_i is the number of nodes in T_i. The ordering of solution of all these subproblems is critical, and is discussed later.

Once all these subproblems have been solved, a further $O(n_1)$ subproblems must be solved, called the *complete* subproblems: for every dart c_1 of T_1, consider T_1 as rooted at the head of c_1 and ordered so that the tail of c_1 is the leftmost child of the root. Then compute the rooted edit-distance between this tree and T_2 rooted at r. Finally, take the minimum of these $O(n_1)$ distances as the unrooted distance.

3.3 The inner dynamic program

Now we describe the dynamic program used to compute each b_1-b_2 subtree edit-distance. The same method is used to solve the complete subproblems. For each dart d_1 of $T_1(b_1)$ and each dart d_2 of $T_2(b_2)$, we have a table entry $A[d_1, d_2]$ assigned the rooted edit-distance between $T_1(b_1)[d_1]$ and $T_2(b_2)[d_2]$.[2] In the following, we omit details on base cases. We discuss the ordering of subproblems after giving the code.

[2]For technical reasons, here we allow a subtree rooted at a dashed edge to be spliced out [6].

```
Assign to $A[d_1, d_2]$ the following value:
If $d_1$ is up-pointing then
  if next($d_1$) is up-pointing then
    return $A[\mathrm{next}(d_1), d_2]$
  else (next dart is down-pointing)
    return
      $\min\{A[\mathrm{next}(d_1), d2],$
          $A[\mathrm{next}(\mathrm{mate}(\mathrm{next}(d_1))), d_2]$
            $+ \mathrm{splicecost}(\mathrm{next}(d_1))\}$
  and similarly for $d_2$.
Else (both $d_1$ and $d_2$ are down-pointing)
  $\min\{A[\mathrm{next}(d_1), d2] + \mathrm{contractcost}(d_1),$
      $A[d_1, \mathrm{next}(d_2)] + \mathrm{contractcost}(d_2),$
      $\mathrm{minmatchcost}(d_1, d_2)\}$
```

where
$\mathrm{minmatchcost}(d_1, d_2) =$

$$\min_{f_1, f_2} \text{cost of deforming path in } T_1 \text{ from } d_1 \text{ to } f_1 \text{ into path in } T_2 \text{ from } d_2 \text{ to } f_2 + M[f_1, f_2] + A[\mathrm{next}(\mathrm{mate}(d_1)), \mathrm{next}(\mathrm{mate}(d_2))]$$

and the min is over all down-pointing darts f_1 that are descendents of d_1 and f_2 that are descendents of d_2.

3.4 Ordering the subproblems of the inner dynamic program

Consider a computation of the inner dynamic program used to compute the b_1-b_2 subtree edit-distance. Let a_i be the dart immediately preceeding mate(b_i). A subproblem of the inner dynamic program involves computing the value of $A[d_1, d_2]$, which requires the values of other entries $A[d'_1, d'_2]$ where d'_i comes after d_i but before a_i in the Euler tour of T_i. Thus we should solve these subproblems $A[d_1, d_2]$ starting with $d_1 = a_1$ and $d_2 = a_2$ and working backward through the Euler tours of the two trees.

3.5 Ordering the subproblems of the outer dynamic program

Each subproblem of the outer dynamic program (not including the complete subproblems) has the form "compute (and assign to $M[b_1, b_2]$) the b_1-b_2 subtree edit-distance." In computing this distance (using the inner dynamic program), we need values $M[b'_1, b'_2]$ where b'_i is a down-pointing dart of $T_i(b_i)$, and therefore b_i points toward b'_i in T_i but not vice versa. The relation $<$ among darts of T_i defined by

$$b'_1 < b_1 \text{ if } b_1 \text{ points toward } b'_1 \text{ but not vice versa}$$

is a partial order, so there is a total order consistent with it. We can then take a total order on the pairs (b_1, b_2) such that if $b'_1 < b_1$ and $b'_2 < b_2$ then $M[b'_1, b'_2]$ is computed before $M[b_1, b_2]$.

The complete subproblems are solved after all the incomplete ones. No complete subproblem depends on any other complete subproblem.

3.6 Piggybacking the computation of some subproblems of the outer dynamic program

The subproblems of the outer dynamic program are indexed by pairs b_1, b_2. Here b_1 ranges over all darts in T_1, while b_2 ranges over only those darts that are down-pointing with respect to the root of T_2. We can effectively restrict the range of b_2 further by using a technique of Zhang and Shasha. Let b_2 be a down-pointing dart of T_2. For another down-pointing dart b'_2, we say b_2 *covers* b'_2 if b_2 points to b'_2 and the path from b_2 to b'_2 only goes toward left children (never to right children).

Zhang and Shasha observe that (in our terms) for any dart b_1 of T_1, during the inner dynamic program computing the b_1-b_2 subtree edit-distance, we can also compute the b_1-b'_2 subtree edit-distance.

The latter is the edit-distance between $T_1(b_1)$ and $T_2(b'_2)$. One step in computing the former is assigning to $A[d_1, d_2]$ the edit-distance between $T(b_1)[d_1]$ and $T(b_2)[d_2]$, where $d_1 = \text{next}(b_1)$ and $d_2 = \text{next}(b'_2)$. However, $T(b_1)[d_1]$ is just $T(b_1)$ by choice of d_1, and $T(b_2)[d_2]$ is $T_2(b_2)[b'_2]$ by choice of d_2 and the covering property.

This suggests we select a subset S of down-pointing darts of T_2 such that for any down-pointing dart b'_2 of T_2 that is not in S, there is some dart b_2 in S that covers b'_2. For this purpose, imagine that the rooted tree T_2 is $T(b)$ for some artificial dart b, and include b in S. Then b covers the darts going down the leftmost branch of T_2. (That is, subtree edit-distances involving these darts can be computed while solving complete subproblems.) The next darts to be included in S are the children darts of the root of T_2 (not including the leftmost child dart), and so on.

3.7 Analysis

The time is dominated by the invocations of minmatchcost. There is a term in some invocation of minmatchcost for every tuple $(b_1, d_1, f_1, b_2, d_2, f_2)$, where b_i is a dart of T_i, d_i is a dart in $T_i(b_i)$, and f_i is a dart in $T_i(d_i)$. Thus the time is proportional to the number of triples (b_1, d_1, f_1) times the number of triples (b_2, d_2, f_2).

To count the former, we can choose b_1 in $O(n_1)$ ways and f_1 in $O(n_1)$ ways. Once these are chosen, since d_1 must be between them, there are $O(\text{diameter}(T_1))$ ways to choose d_1.

To count the latter, note that we restrict our attention to darts b_2 that are down-pointing with respect to the root of T_2. We can choose f_2 in $O(n_2)$ ways. Once f_2 is chosen, we can choose each of b_2 and d_2 in $O(\text{depth}(T_2))$ ways.

Thus a very loose upper bound on the time required for computing edit-distance is $O(n_1^2 n_2 \text{diameter}(T_1)(\text{depth}(T_2))^2)$.

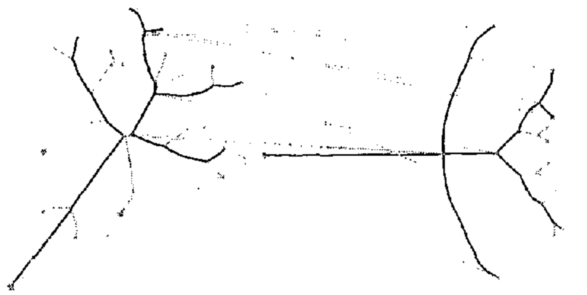
Figure 13: The result of matching the shock graphs of two planes. The lightly colored edges in the shock graphs have been pruned. Observe that the correspondence obtained is intuitive, *i.e.*, the nose, wings, and tail of the first plane is matched to the nose, wings, and tail respectively of the second plane.

3.8 Choice of root

There is a problem with fixing the root of one of the trees: for any down-pointing dart d of T_2, our algorithm prevents splices that would involve the elimination of all edges not in $T_2(d)$. However, by an appropriate choice of root, we hope to ensure that all such prevented splices are prohibitively expensive anyway, and hence that they would not appear in the optimal edit sequence. We choose as the root of T_2 a degree-three node r with outgoing darts g_1, g_2, g_3 so as to maximize $\hat{c} = \min_j\{\text{splicecost}(g_i)\}$. This guarantees that any prevented splice would have cost at least $\hat{c}$.

This heuristic works well; it seems to occur only rarely that the best edit sequence is prevented. A somewhat safer heuristic would be to try several nodes with high values of $\hat{c}$ but we have not yet tried that.

4 Results

In this section, we present the results of applying our matching approach to a database of 36 binary shapes. The database consists of shapes from six different categories (fishes, tools, planes, rabbits, "greebles", and hands) with six different shapes in each category. Comparisons are made between every pair of shapes. Figure 13 shows an example of the resulting correspondence and Figure 14 shows the running times for comparing all the shapes in our database. The pairwise comparison results are reported in Table 1. The five nearest neighbors for each shape are highlighted. Observe that the shapes are categorized intuitively, *i.e.*, the nearest

	0	54	66	67	56	54	72	70	72	70	71	74	98	106	101	79	76	76	95	89	87	91	91	94	94	87	88	80	83	86	89	97	90	88	91	94
	54	0	57	66	62	60	73	71	75	69	70	74	106	111	107	83	80	82	97	87	85	90	90	90	95	84	88	81	79	85	93	97	92	93	93	97
	66	57	0	60	65	59	84	80	89	90	91	78	100	112	101	91	82	86	99	84	90	96	99	98	91	88	87	84	89	89	94	99	93	93	94	105
	67	66	60	0	71	51	81	73	80	73	79	71	112	124	120	97	91	94	108	96	95	98	95	98	106	96	102	94	94	97	100	107	103	100	103	103
	56	62	65	71	0	61	77	70	96	91	94	76	103	108	108	86	87	88	97	87	88	97	99	96	105	93	94	93	93	96	95	97	95	96	95	99
	54	60	59	51	61	0	73	69	84	73	78	69	108	121	112	90	89	93	104	95	90	94	93	94	104	92	95	89	86	91	98	103	98	98	100	104
	72	73	84	81	77	73	0	29	55	52	54	41	128	139	137	123	111	121	109	102	93	89	100	91	109	93	97	91	89	101	118	123	139	129	131	127
	70	71	80	73	70	69	29	0	53	47	56	37	130	145	148	132	122	122	117	104	92	91	97	91	109	95	95	96	91	98	129	139	134	126	122	129
	72	75	89	80	96	84	55	53	0	30	37	54	122	145	148	122	112	117	118	107	97	89	100	93	105	90	101	87	85	99	124	132	128	122	119	123
	70	69	90	73	91	73	52	47	30	0	29	48	127	132	138	118	109	106	105	106	96	89	96	93	103	89	100	85	84	99	120	128	125	117	115	115
	71	70	91	79	94	78	54	56	37	29	0	56	276	268	274	252	260	244	137	110	99	93	95	93	110	93	102	87	88	104	224	233	226	232	223	238
	74	74	78	71	76	69	41	37	54	48	56	0	132	148	145	135	122	123	111	107	98	96	94	95	109	95	103	94	95	103	133	137	136	127	130	130
	98	106	100	112	103	108	128	130	122	127	276	132	0	86	94	77	79	73	101	96	104	113	113	114	114	110	114	108	109	114	92	99	92	93	95	91
	106	111	112	124	108	121	139	145	145	132	268	148	86	0	79	83	83	83	114	109	115	126	123	129	123	122	122	124	123	123	96	102	96	99	104	101
	101	107	101	120	108	112	137	148	148	138	274	145	94	79	0	74	75	96	119	109	119	127	129	130	123	124	119	126	124	122	99	106	99	103	107	106
	79	83	91	97	86	90	123	132	122	118	252	135	77	83	74	0	55	73	96	96	96	108	110	109	106	104	103	107	102	106	86	90	84	87	90	87
	76	80	82	91	87	89	111	122	112	109	260	122	79	83	75	55	0	74	93	86	93	104	102	106	95	97	94	95	97	97	87	95	85	90	94	85
	76	82	86	94	88	93	121	122	117	106	244	123	73	83	96	73	74	0	87	80	83	97	98	101	96	87	91	92	91	92	77	85	79	83	85	77
	95	97	99	108	97	104	109	117	118	105	137	111	101	114	119	96	93	87	0	51	51	60	73	67	97	92	94	96	91	93	93	97	93	91	91	91
	89	87	84	96	87	95	102	104	107	106	110	107	96	109	109	96	86	80	51	0	52	69	61	69	94	91	93	95	83	91	84	91	86	90	87	91
	87	85	90	95	88	90	93	92	97	96	99	98	104	115	119	96	93	83	51	52	0	52	59	52	92	83	87	83	81	87	100	105	99	102	96	98
	91	90	96	98	97	94	89	91	89	89	93	96	113	126	127	108	104	97	60	69	52	0	63	54	88	82	78	80	74	83	104	108	105	102	103	105
	91	90	99	95	99	93	100	97	100	96	95	94	113	123	129	110	102	98	73	61	59	63	0	51	83	84	81	83	76	83	98	107	103	108	106	105
	94	90	98	98	96	94	91	91	93	93	93	95	114	129	130	109	106	101	67	69	52	54	51	0	87	82	80	81	72	81	96	102	98	102	105	105
	94	95	91	106	105	104	109	109	105	103	110	109	114	123	123	106	95	96	97	94	92	88	83	87	0	49	57	49	58	52	92	101	95	96	94	93
	87	84	88	96	93	92	93	95	90	89	93	95	110	122	124	104	97	87	92	91	83	82	84	82	49	0	51	40	51	49	93	100	94	94	93	94
	88	88	87	102	94	96	97	95	101	100	102	103	114	122	119	103	94	91	94	93	87	78	81	80	57	51	0	47	54	48	102	107	102	103	94	94
	80	81	84	94	93	89	91	90	87	85	87	94	108	124	126	107	95	92	96	95	83	80	83	81	49	40	47	0	43	45	102	112	106	101	95	97
	83	79	89	94	93	86	89	91	85	84	88	95	109	123	124	102	97	91	91	83	81	74	76	72	58	51	54	43	0	57	92	101	94	98	95	95
	86	85	89	97	96	91	101	98	99	99	104	103	114	123	122	106	97	92	93	91	87	83	83	81	52	49	48	45	57	0	99	106	100	103	98	98
	89	93	94	100	95	98	118	129	124	120	224	133	92	96	99	86	87	77	93	84	100	104	98	96	92	93	102	102	92	99	0	41	20	32	36	55
	97	97	99	107	97	103	123	139	132	128	233	137	99	102	106	90	95	85	97	91	105	108	107	102	101	100	107	112	101	106	41	0	39	44	41	60
	90	92	93	103	95	98	133	134	128	125	226	136	92	96	99	84	85	79	93	86	99	105	103	98	95	94	102	106	94	100	20	39	0	35	30	57
	88	93	93	100	96	98	129	126	122	117	232	127	93	99	103	87	90	83	91	90	102	102	108	102	96	94	103	101	98	103	32	44	35	0	42	53
	91	93	94	103	95	100	131	122	119	115	223	130	95	104	107	90	94	85	91	87	96	103	106	105	94	93	94	95	95	98	36	41	30	42	0	50
	94	97	105	103	99	104	127	129	123	115	238	130	91	101	106	87	85	77	91	91	98	105	105	105	93	94	94	97	95	98	55	60	57	53	50	0

Table 1: The optimal edit distance costs for pairwise matching shapes in the database.

neighbors of the "tool" shapes are in the "tool" category. The nearest neighbors belong to the same category in 36/36, 36/36, 36/36, 34/36, and 32/36 cases, respectively.

A similar database (of 25 shapes) has been used in [10, 2, 1]. We now compare our performance to theirs. The performance of the matching algorithm on this database has been was reported in terms of the categorization results, *i.e.*, the number of times the three nearest neighbors belonged to the same category. The results in proportions reported are as follows: (23, 21, 20) in [10], (25, 21, 19) in [2], and (25, 24, 23) in [1]. The performance of our approach is (25, 25, 22) which is a significant improvement over the results in [10, 2] and marginally better than those in [1]. We believe however, that when the dimensions of variations of the database include significant occlusion, and articulation these differences will be magnified.

Table 2 shows the results of indexing into the above shape database using user drawn queries. In most cases, the algorithm returns shapes that are visually intuitive, *i.e.*, if the query shape is a "fish" the top matches are "fishes". Our indexing results are a significant improvement over those presented in [10].

We have demonstrated that our approach performs categorization and indexing tasks intuitively, and our

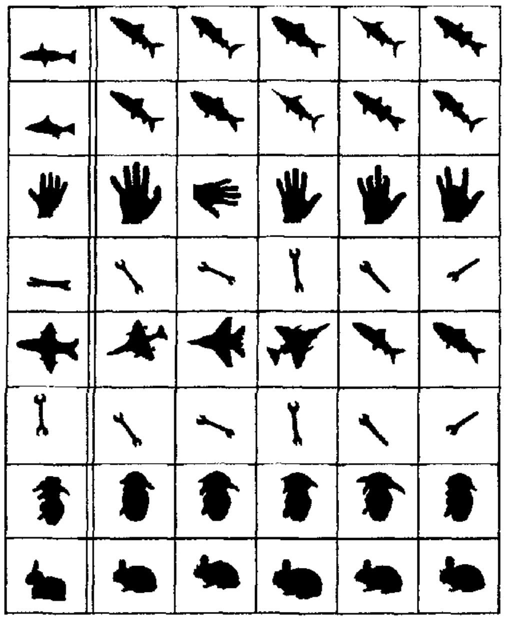

Table 2: This figure depicts the results of indexing user-drawn sketches into our database of shapes. In every row, the left most shape is the filled user sketch, and the top five matches in the database are shown. The sketches in the first five rows were used in [10] to illustrate indexing results. In [10], for all the queries 3/5 top matches were from the same category. In contrast, our approach picks shapes from the correct category in the top five matches in all cases except the plane, where the fourth and fifth choices are incorrect.

experimental results are better than those of previous approaches.

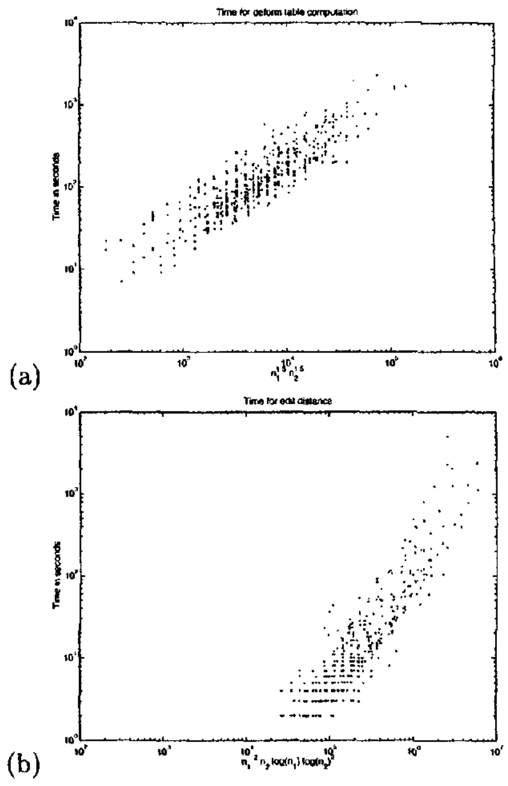

Figure 14: The running times for computing the deform table (a) and edit distance (b). n_1 and n_2 are the number of nodes in two shock graphs being matched.

References

[1] S. Belongie and J. Malik. Matching with shape contexts. In *CBAIVL*, 2000.

[2] Y. Gdalyahu and D. Weinshall. Flexible syntactic matching of curves and its application to automatic hierarchical classification of silhouettes. *PAMI*, 21(12):1312–1328, 1999.

[3] P. J. Giblin and B. B. Kimia. On the local form and transitions of symmetry sets, and medial axes, and shocks in 2D. *ICCV*, pages 385–391, 1999.

[4] S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *PAMI*, 18(4):377–388, 1996.

[5] B. B. Kimia, A. R. Tannenbaum, and S. W. Zucker. Shapes, shocks, and deformations, I: The components of shape and the reaction-diffusion space. *IJCV*, 15:189–224, 1995.

[6] P. Klein, S. Tirthapura, D. Sharvit, and B. Kimia. A tree-edit distance algorithm for comparing simple, closed shapes. In *SODA*, pages 696–704, 2000.

[7] T. Liu and D. Geiger. Approximate tree matching and shape similarity. In *ICCV*, pages 456–462, 1999.

[8] M. Pelillo, K. Siddiqi, and S. Zucker. Matching hierarchical structures using association graphs. *PAMI*, 21(11):1105–1120, 1999.

[9] T. B. Sebastian, J. J. Crisco, P. N. Klein, and B. B. Kimia. Constructing 2D curve atlases. In *MMBIA*, pages 70–77, 2000.

[10] D. Sharvit, J. Chan, H. Tek, and B. B. Kimia. Symmetry-based indexing of image databases. *JVCIR*, 9(4):366–380, 1998.

[11] K. Siddiqi and B. B. Kimia. A shock grammar for recognition. CVPR, pages 507–513, 1996.

[12] K. Siddiqi, A. Shokoufandeh, S. Dickinson, and S. Zucker. Shock graphs and shape matching. *IJCV*, 35(1):13–32, 1999.

[13] R. E. Tarjan and U. Vishkin. An efficient parallel biconnectivity algorithm. *SIAM J. Comput.*, 14(4):862–874, 1985.

[14] R. C. Veltkamp and M. Hagedoorn. State of the art in shape matching. Technical Report UU-CS-1999-27, Utrecht University, Netherlands, 1999.

[15] K. Zhang and D. Sasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18:1245–1262, 1989.

[16] S. C. Zhu and A. L. Yuille. FORMS: A flexible object recognition and modeling system. *IJCV*, 20(3), 1996.