Planar-Shape Prototype Generation Using a Tree-Based Random Greedy Algorithm

Wen-Yao Chen, *Student Member, IEEE*, Wen-Liang Hwang, *Senior Member, IEEE*, and Tien-Ching Lin

***Abstract*—A prototype is representative of a set of similar objects. This paper proposes an approach that formulates the problem of prototype generation as finding the mean from a given set of objects, where the prototype solution must satisfy certain constraints. These constraints describe the important perceptual features of the sample shapes that the proposed prototype must retain. The contour prototype generated from a set of planar objects was used as an example of the approach, and the corners were used as the perceptual features to be preserved in the proposed prototype shape. However, finding a prototype solution for more than two contours is computationally intractable. A tree-based approach is therefore proposed in which an efficient greedy random algorithm is used to obtain a good approximation of the proposed prototype and analyze the expected complexity of the algorithm. The proposed prototype-generation process for hand-drawn patterns is described and discussed in this paper.**

***Index Terms*—Pattern recognition, prototype generation, random algorithm.**

I. Introduction

VARIOUS techniques have been developed for generating the mean shape of a class of objects. Among them, morphing produces the mean shape of perceptually different shapes. Prototype-shape generation, on the other hand, is a technique that generates a prototype shape from the shapes of similar objects. It has already found many applications in industrial design, medical imaging, computer animation, machine vision, and pattern recognition [3], [11], [16]. The technique is comprised of two components: the dissimilarity measurement between sample objects, and the methodology for finding the prototype. Many approaches have been proposed for generating a prototype shape. In [4], [10], and [19], a structured description created by the symbolic representations of sample shapes is proposed. Other approaches include using deformable models [5], [20], and graph models [7].

Our prototype shape is defined as the mean of a given set of sample shapes. This mean is constrained by certain salient perceptual features within the sample shapes that must be preserved in the prototype. The prototype definition does not necessarily yield a prototype that matches the perceptual quality of a prototype. However, with the definition, we can analyze the prototype-generation process and generate a prototype with some properties that are required by all the objects. Our definition of a prototype is a combination of the proposed model in [15], in which a prototype is defined as the mean shape of the sample shapes, and the proposed model in [19] in which a prototype must retain many common visual properties within the sample shapes. For example, let X be the random variable that produces pentagons and $\{C_i | i = 1, \ldots, n\}$ be n realizations thereof. If the contour of each C_i contains five sides and five angles, then our prototype pentagon must have five sides and five angles and is the mean shape of the samples after the angles of the sample polygons have been aligned.

Let $\mathbf{C}$ be a set of shapes that preserves the common salient features of sample shapes $\{C_i | i = 1, \ldots, n\}$. The proposed prototype can be defined as the mean shape that produces the minimum of some types of distortion if it is used to approximate all the sample shapes. We assume that the prototype $\bar{X}$ is found in $\mathbf{C}$ so that the distortion D is

$$\bar{X} = \arg\min_{C \in \mathbf{C}} \frac{1}{K} \sum_{i=1}^{n} D(C, C_i) \tag{1}$$

where K is n if C is not one of the sample shapes; otherwise, K is $n-1$. Since finding the solution of our mean shape is a minimization problem that is constrained by preserving the salient perceptual features within the sample shapes, the proposed prototype-generation problem is a constrained-optimization problem. Generally, the problem can be solved by using a technique from mathematical programming or variational calculus. However, for our problem, which finds the prototype from a set of discrete planar contours, the common features in different contours must be aligned. Thus, we are unable to solve the proposed problem by applying mathematical programming or variational calculus directly. Let C_i be a realization of a planar shape, where the contour of C_i contains m_i salient features (corners). Before we can find the prototype of n realizations, we need to find the common salient features of $\{C_i\}$, because our prototype must be in space $\mathbf{C}$, where any instance preserves the common salient features of the realizations. Thus, the corresponding corners of all shapes must be identified and labeled first. The optimal algorithm that yields the minimal cost of aligning the corners between C_i and C_j has a complexity of at least $\Omega(m_i m_j)$, because the cost of any corner pair of C_i and C_j must be calculated at least once in the optimal algorithm. Following the same reasoning as that used for aligning two contours, finding the optimal alignment of $\{C_i | i = 1, \ldots, n\}$ is at least $\Omega(\prod_{i=1}^{n} m_i)$. The complexity of aligning the common salient features of multiple shapes grows

Manuscript received April 2, 2004; revised December 27, 2004 and May 20, 2004. This paper was recommended by Associate Editor Q. Zhu.

W.-Y. Chen is with the Institute of Information Science, Academia Sinica, Taipei, Taiwan, R.O.C., and also with National Tsing Hua University, Hsinchu, Taiwan, R.O.C. (e-mail: wenyao@iis.sinica.edu.tw).

W.-L. Hwang is with the Institute of Information Science, Academia Sinica, Taipei, Taiwan, R.O.C. (e-mail: whwang@iis.sinica.edu.tw).

T.-C. Lin is with the Institute of Information Science, Academia Sinica, Taipei, Taiwan, R.O.C., and also with National Taiwan University, Taipei, Taiwan, R.O.C.

Digital Object Identifier 10.1109/TSMCB.2005.861880

exponentially as the number of realizations increases; thus, it is infeasible for practical applications. Instead, we use a balanced-binary-tree approach, where a sequence of prototypes for two shapes is organized as a balanced binary tree. Each node is associated with a contour, and the contour of an internal node is the result of aligning the contours of its children. Although this approach avoids the high computational cost of finding the optimal salient features corresponding to more than two shapes simultaneously, it restricts our prototype to a suboptimal and tree-dependent solution.

Because there are $n!$ combinations of possible balanced binary trees for n samples, an exhaustive search of all possible balanced binary trees for the optimum prototype is impossible. Therefore, to find a balanced tree that yields a good solution, we use a greedy iterative random algorithm, because it is simple to implement and yields a satisfactory result. At each iteration, a pair of shapes at any leaf nodes in the current balanced tree is chosen at random. The shapes' positions are then swapped to obtain a new balanced tree. If this tree is of smaller distortion, we substitute it for the current tree. We analyze our random algorithm and show that the expected complexity of performing one iteration of the proposed random algorithm is $\mathcal{O}(m^2 N \log_2 n)$, where n, m, and N are, respectively, the number of sample shapes, the maximum number of corners in a sample shape, and the total number of sample points of the sample shapes. When a leaf is deleted from, (or inserted into) a tree, all the contours of the internal nodes on the paths from the root to the leaf will be modified. To implement the deletion or insertion of a leaf efficiently, the contour alignment of an internal node is approximated by a sequence of subcontour alignments.

We use the above approach to find the prototype from the contours of planar objects. The corners of the prototype contour are used to segment it into a sequence of smooth subcontours. A distortion is then introduced to measure the error if one subcontour is matched to the other subcontour by a similarity transform. The total distortion when matching two contours is the sum of all the distortions found when matching pairs of subcontours.

Because our algorithm for generating a prototype is motivated by the approach in [15], we review and compare latter with our approach. In [15], the contour of a planar object is represented as a cyclic string of symbols. The distance of two strings is the edited distance between the strings [17]. The editing operations are insert a symbol, delete a symbol, or substitute a symbol with a string. The mean string (prototype) is defined by (1), where $\mathbf{C}$ is the set of all possible cyclic strings. However, finding the mean string is an NP-hard problem [6]. For practical purposes, an algorithm is proposed that progressively updates the prototype. The result is a binary tree in which the root is the prototype. Compared to this algorithm, the unique feature of our algorithm is that our prototype is constructed by an iteratively greedy random algorithm, which successively modifies the initial prototype to obtain the final prototype.

The remainder of the manuscript is organized as follows. In Section II, we introduce the distortion measurement between planar contours and present a solution for generating a prototype for two contours. In Section III, we present a random algorithm for the generation of a prototype for more than two objects. The experimental results of our algorithm are given in Section IV. Finally, in Section V, we present our conclusions.

II. Distortion and Prototype of Two Contours

Here, we consider the generation of a prototype contour from two contours of a planar object. We assume that both the contours are closed, contain at least two corner points, and are partitioned into a sequence of subcontours separated by corner points (the endpoints of the subcontours). The prototype of the two contours is generated by finding the optimal alignment between the sequences of subcontours of the two contours. Before we propose our two-contour alignment algorithm, we introduce a measure of the similarity between two subcontours (curves). Our measurement is similar to that in [5]. There are other measurements such as the editing distance [15], the graph-matching distance [7], the distance derived from multiscale representation [19], and an energy-minimization approach based on the length and curvature of contours [17].

A. Dissimilarity Between Two Curves

Many methods have been used to measure the similarity of two curves. We propose a measure based on the similarity transform, because it can be implemented easily and achieves a satisfactory result. The similarity transform relates two points, p and q, by combining translation $\mathbf{t}$, scaling s, and rotation $\mathbf{r}_\theta$

$$q = s\mathbf{r}_\theta p + \mathbf{t} \tag{2}$$

where

$$\mathbf{r}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

The similarity transform between two curves can be obtained by aligning the end points between the curves. Let p_1 and p_2 be the end points of one curve, and q_1 and q_2 be the end points of the other curve. Aligning p_1 to q_1 and p_2 to q_2, we have

$$q_1 = s\mathbf{r}_\theta p_1 + \mathbf{t} \tag{3}$$
$$q_2 = s\mathbf{r}_\theta p_2 + \mathbf{t}. \tag{4}$$

Subtracting (3) from (4), we obtain

$$q_2 - q_1 = s\mathbf{r}_\theta (p_2 - p_1).$$

Since the end points p_1, p_2, q_1, and q_2 are given planar vectors, the above equation can be used to solve the scaling s and rotation $\mathbf{r}_\theta$

$$s = \frac{\|q_2 - q_1\|}{\|p_2 - p_1\|} \tag{5}$$

$$\mathbf{r}_\theta = \frac{(q_2 - q_1)(p_2 - p_1)^{\mathrm{T}}}{\|q_2 - q_1\|\|p_2 - p_1\|.} \tag{6}$$

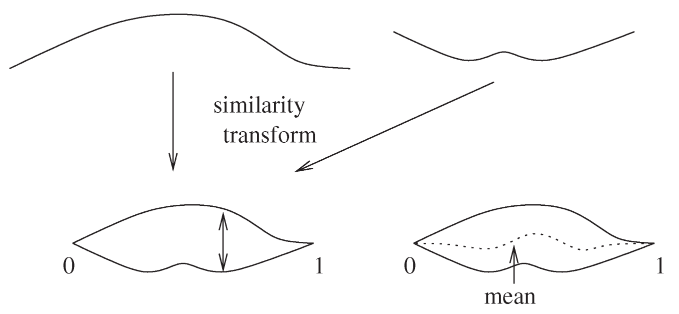


Fig. 1. Bottom-left: The two endpoints of the two top subcontours are aligned. Bottom-right: The mean subcontour of the aligned subcontours.

The translation vector $\mathbf{t}$ can be obtained after substituting s and $\mathbf{r}_\theta$ into (3). We use S to denote the similarity transform that takes the curve c^2 to c^1 by aligning their end points. The error curve e_{12} is $c^1 - Sc^2$. If we take the inverse similarity transform S^{-1} to the curve c^1 and align the end points of the resultant curve with c^2, we obtain the error curve $e_{21} = c^2 - S^{-1}c^1$. In general, e_{12} is not the same as e_{21}, since their supports may have different lengths. Therefore, we should normalize the error curves so that the dissimilarity measure of curves c^1 and c^2 does not depend on which curve is transformed and aligned with the other curve first. We thus align both curves with the line segment whose end points are at $[0\ 0]^{\mathrm{T}}$ and $[1\ 0]^{\mathrm{T}}$ by using the similarity transform. The parameters of the obtained similarity can be derived from (2), (5) and (6). We use $\mathcal{N}$ to denote the procedure shown as the alignment of two curves in Fig. 1. The dissimilarity measure of curves c^1 and c^2 is then defined as

$$\begin{aligned} d(c^1, c^2) &= \|\mathcal{N}c^1 - \mathcal{N}c^2\|_2^2 \\ &= d(c^2, c^1) \end{aligned}$$

where $\|\mathcal{N}c^1 - \mathcal{N}c^2\|_2^2$ can be represented as parameterized curves with the arc-length parameter $t \in [0, 1]$, and then evaluated by $\int_0^1 [\mathcal{N}c^1(t) - \mathcal{N}c^2(t)]^2 dt$. This is approximated numerically by integration with a complexity $\Theta(N(c^1) + N(c^2))$, where $N(c^1)$ and $N(c^2)$ are, respectively, the number of sample points in curves c^1 and c^2.

Now, let us assume that contour C_i has m_i subcontours, contour C_j has m_j subcontours, and $m_i \le m_j$. Furthermore, let the end points of the subcontours in C_i and C_j be in sequential order, namely, $\{p_1, p_2, \ldots, p_{m_i}\}$ and $\{q_1, q_2, \ldots, q_{m_j}\}$, respectively. We use the notation $[a, b]$ to denote the curve between end points a and b. Let $\mathcal{A}$ be the alignment function that aligns closed contours C_i and C_j. The function performs a sequence of curve alignments that take a curve in C_i to a curve in C_j. In order to retain the order of subcontours in a closed contour, $\mathcal{A}$ must have the following properties:

1) $a(k)$ must be a monotonic function: $b(k) = a(k) \bmod m_j$.
2) $\mathcal{A} : [p_k, p_{(k+1) \bmod m_i}] \rightarrow [q_{b(k)}, q_{b(k+1)}]$, where the corners $q_{b(k)}$ and $q_{b(k+1)}$ may not be adjacent to each other for $k = 1, 2, \ldots, m_i$.

We define a legal alignment as the alignment of two contours that satisfies the above properties. Let $(\mathcal{A}, b)$ be a legal alignment and the notation $C_i \sim C_j = \{(p_1, q_{b(1)}), (p_2, q_{b(2)}), \ldots, (p_{m_i}, q_{b(m_i)})\}$ represent the ordered sequence of matched corners between C_i and C_j. The dissimilarity measure of the two contours C_i and C_j is defined such that

$$D(C_i, C_j) = \min_{(\mathcal{A}, b)} \sum_{k=1}^{m_i} d\left(\left[p_k, p_{(k+1) \bmod m_i}\right], \left[q_{b(k)}, q_{b(k+1)}\right]\right). \quad (7)$$

Thus, the dissimilarity of two contours is the minimum of the sum of the subcontour distances of all possible legal alignments of the two contours.

Without loss of generality, to analyze the computational complexity, let us assume that p_1 matches q_1. We use the notations P_k^1 and $Q_{b(k)}^1$ to denote the curves $[p_k, \ldots, p_{m_i}, p_1]$ and $[q_{b(k)}, \ldots, q_{b(m_i)}, q_{b(1)}]$, respectively. We then have

$$D(C_i, C_j) = \min_{b(2)} \left\{ d\left([p_1, p_2], \left[q_{b(1)}, q_{b(2)}\right]\right) + D_s\left(P_2^1, Q_{b(2)}^1\right) \right\} \quad (8)$$

where $D_s(P_2^1, Q_{b(2)}^1)$ is the cost of aligning $[p_2, \ldots, p_{m_i}, p_1]$ and $[q_{b(2)}, \ldots, q_{b(m_i)}, q_{b(1)}]$. D_s can be recursively derived for $k = 3, \ldots, m_i - 1$:

$$D_s\left(P_k^1, Q_{b(k)}^1\right) = \min_{b(k+1)} \left\{ d\left([p_k, p_{k+1}], \left[q_{b(k)}, q_{b(k+1)}\right]\right) + D_s\left(P_{k+1}^1, Q_{b(k+1)}^1\right) \right\}. \quad (9)$$

When $k = m_i$, the recursion should be stopped; therefore

$$D_s\left(P_{m_i}^1, Q_{b(m_i)}^1\right) = d\left([p_{m_i}, p_1], \left[q_{b(m_i)}, q_{b(1)}\right]\right). \quad (10)$$

There are at most m_j choices for each $b(i)$ with $i = 2, \ldots, m_i$ in the above equations. Thus, in the case where p_1 matches q_1, the complexity of finding the optimal alignment from curve $\{p_2, \ldots, p_{m_i}\}$ to curve $\{q_2, \ldots, q_{m_j}\}$ is $\mathcal{O}(m_j^2(N(C_i) + N(C_j)))$, where $N(C_i)$ and $N(C_j)$ are, respectively, the number of sample points of C_i and C_j.

The recursions in (8)–(10) can be implemented effectively by using dynamic programming [1], [14] based on the assumption that p_1 matches q_1. In our applications, p_1 may not match q_1, thus, we need to calculate the optimal alignment for each of the assumptions that p_1 matches q_i with $i = 1, \ldots, m_j$. We then choose the optimal alignment that yields the minimal dissimilarity value from the results of these assumptions. As a consequence, an $\mathcal{O}(m_j^3(N(C_i) + N(C_j)))$ complexity is applied to find the solution of (7). For convenience, in the following discussion, we assume that p_1 and q_1 are aligned; thus, the complexity of our two-contour alignment algorithm is $\mathcal{O}(m_j^2(N(C_i) + N(C_j)))$.

B. Prototype of Two Contours

We now consider the case where the prototype of two contours, C_i and C_j, is to be generated. C_i has $N(C_i)$ sample points and m_i subcontours, C_j has $N(C_j)$ sample points and m_j subcontours, and $m_i \leq m_j$. According to the definition in (1), the prototype of two contours $\bar{X} \in \mathbf{C}$ satisfies

$$D(\bar{X}, C_i) + D(\bar{X}, C_j) \leq D(X, C_i) + D(X, C_j), \quad \forall X \in \mathbf{C} \tag{11}$$

where D is measured by (7), in which we impose the constraint that the prototype of C_i and C_j must have m_i subcontours. Therefore, $\mathbf{C}$ represents all possible contours that have m_i subcontours. The prototype $\bar{X}$ cannot be obtained easily, because it must simultaneously match contours C_i and C_j and achieve minimal distortion. A simple way to obtain a good approximation is to use the optimal-alignment function $(\mathcal{A}^*, b^*)$ between contours C_i and C_j to align the contours of $\bar{X}$ and C_i, and $\bar{X}$ and C_j. Then, the prototype contour $\bar{X}$ of C_i and C_j is formed by concatenating the m_i average (mean) subcontours of the matched subcontours of C_i and C_j according to $(\mathcal{A}^*, b^*)$.

Let S be the similarity transform that takes the curve $[q_{b^*(k)}, q_{b^*(k+1)}]$ in C_j to the curve $[p_k, p_{(k+1) \bmod m_i}]$ in C_i. The kth subcontour of $\bar{X}$, which is denoted as $\bar{x}_k$, is the mean curve and is calculated by

$$\frac{1}{2}\left(\left[p_k, p_{(k+1) \bmod m_i}\right] + S\left[q_{b^*(k)}, q_{b^*(k+1)}\right]\right).$$

Using variational calculus, it is easy to show that $\bar{x}_k$ is the curve, among all the curves whose end points are, respectively, p_k and $p_{(k+1) \bmod m_i}$, which minimizes the root-mean-square error when it is used to approximate curves $[p_k, p_{(k+1) \bmod m_i}]$ and $S[q_{b^*(k)}, q_{b^*(k+1)}]$ simultaneously. We present the results in the Appendix. Since the prototype's subcontour is the mean of the aligned subcontours, our prototype is the mean contour after C_i and C_j have been aligned. By concatenating the prototype subcontours sequentially, we obtain the prototype of two contours, i.e., $\bar{X} = \bar{x}_1\bar{x}_2 \cdots \bar{x}_{m_i}$. The proposed two-contour alignment algorithm needs to find the optimal alignment of contours C_i and C_j to obtain the prototype of the two contours. Consequently, its complexity is $\mathcal{O}(m_j^2(N(C_i) + N(C_j)))$.

The two-contour alignment algorithm could be extended to align more than two contours simultaneously; however, the complexity would be too high. To find the optimal alignment that simultaneously aligns n contours, we need to visit every combination of the contours' corners at least once. Let the number of sample points and the number of corners be $N(C_1), N(C_2), \ldots, N(C_n)$ and $m_1, m_2, \ldots, m_n$, respectively; and let $N = \sum_{i=1}^n N(C_i)$ and $m = \max(m_1, m_2, \ldots, m_n)$. There is a total of $\prod_{i=1}^n m_i$ combinations of the corners and each combination requires an order of N to calculate the cost of the alignment. The complexity of the optimal alignment is at least $\Omega(m^n N)$. Thus, an efficient method must be developed, even though it may give a suboptimal-alignment solution.

III. Prototype Based on a Hierarchical Tree

Let $\mathbf{C}$ be all possible prototype candidates for our constraints. Our approach to finding the prototype for more than two contours is to limit our prototype solution to the subset of $\mathbf{C}$ that has an efficient algorithm for a suboptimal prototype solution in the subset. To find a suboptimal solution for certain constrained problems, we use an approach that is popular in many numerical methods. Starting with the initial prototype, we modify it slightly, such that the modified version reduces the distortion of the current prototype. The process is repeated until we find a prototype in which the distortion cannot be reduced further. The final prototype is our solution. This greedy iterative algorithm can be efficiently performed by using a balanced binary tree.

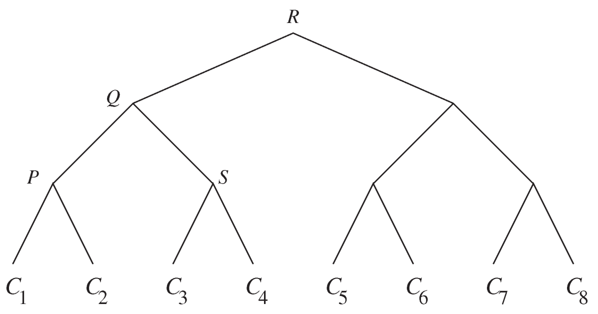


Fig. 2. Contour associated with node Q is built from the contours of nodes P and S and is the prototype of contours C_1, C_2, C_3, and C_4. The contour of root R is the prototype of all the C's.

We build a balanced binary tree so that all the sample contours are on the leaves of the tree and each internal node is associated with a contour that is the prototype of all the sample contours on the leaves of the internal nodes. Fig. 2 shows an example of our balanced binary tree with eight samples. A binary tree of n leaves has $n-1$ internal nodes. Because the prototype contour at an internal node is obtained from the two contours of its children, we need to perform $n-1$ two-contour alignments to obtain our balanced binary tree. For n sample contours, there is an $n!$ number of combinations for a possible balanced binary tree. An exhaustive search of all possible balanced binary trees for the optimum prototype is impossible. We therefore use a heuristic to modify the current balanced binary tree with the requirement that the distortion of the new root prototype decreases. The heuristic is implemented with a randomization approach. At each iteration, we randomly choose a pair of leaves of the current tree and swap their positions to obtain a new tree. This process is repeated several times until the distortion of the resultant tree cannot be reduced further, or a computational limit is reached. Our proposed algorithm is as follows.

```
Algorithm Prototype
1) Let $\Gamma$ and $\partial\Gamma$ be the initial balanced binary tree and the leaves of the tree, respectively, and let $j = 0$.
2) Randomly select a pair of leaves $C_l$ and $C_r \in \partial\Gamma$. Let the positions of $C_l$ and $C_r$ be at $p(l)$ and $p(r)$, respectively.
3) Swap $C_l$ and $C_r$ by deleting them from $\Gamma$ and inserting $C_l$ into $p(r)$ and $C_r$ into $p(l)$.
4) If the distortion decreases, the new tree is used for the next iteration.
5) $j = j + 1$.
6) If $j >$ the threshold value, STOP, else GOTO 2.
```

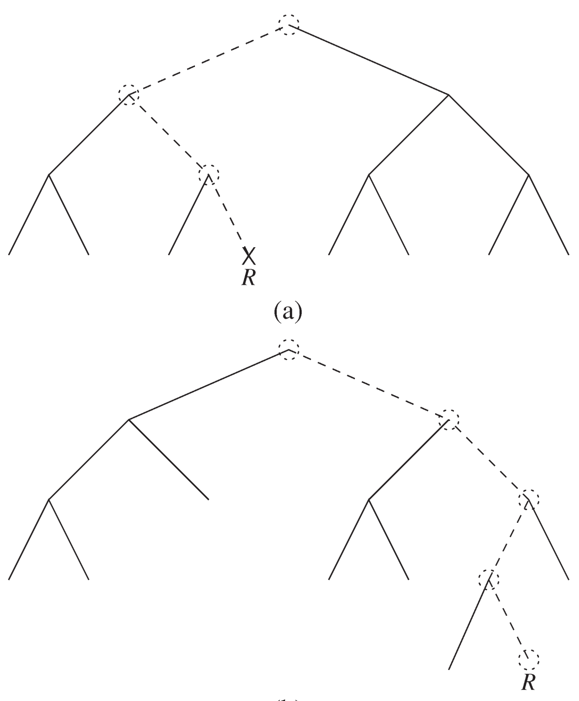


Fig. 3. When node R is deleted from (a) and inserted in (b), the prototype associated with any node on the dash-dot paths must be updated.

When leaf R is removed from Γ in Step 3, the contour of any internal node on the path from the root of Γ to the leaf is modified by performing one alignment of its children's contours. As with the deletion of a leaf, when R is inserted as a leaf in $\partial\Gamma$, one two-contour alignment is performed on each node from the root of $\partial\Gamma$ to R. Fig. 3 shows that the nodes on the dash-dot paths are modified when a node is deleted from one subtree and inserted in another subtree. We begin with a balanced binary tree in step 1 and, at each iteration, modify the current balanced binary tree by randomly picking a pair of leaves and swapping them in step 5. After each iteration, the balanced tree Γ either remains unchanged, or it is replaced by a new balanced binary tree. Thus, at each iteration, the algorithm takes $\mathcal{O}(\log_2 n)$ times two-contour alignments. If $\bar{i}$ is the expected number of iterations in the algorithm, then our random algorithm will perform, on average, no more than $\bar{i}\log_2 n$ two-contour alignments. Therefore, the expected complexity of the algorithm is $\mathcal{O}(\bar{i}m^2N\log_2 n + m^2nN)$, where $m = \max\{m_1, m_2, \ldots\}$ and $N = \sum_i N(C_i)$ are, respectively, the maximum number of corners and the total sample points of all contours $\{C_i\}$. $\mathcal{O}(m^2nN)$ is the complexity of building the initial binary tree. In Section III-A, we show that this complexity can be reduced by using a tree-based approximation method to align two contours.

This algorithm has an upper bound on the number of sample contours that produce the prototype. The bound can be approximated by the following reasoning. If we assume that the probability of a corner to be aligned in the two-contour-algorithm is $p < 1$ and m is the maximum number of corners in the sample contours, then the number of corners in the mean contour is mp. The number of corners of a prototype that is l levels above the leaf level is thus mp^l. Because we do not want to oversmooth the root prototype, we impose a constraint that the number of corners in a prototype must be at least k. Thus, a prototype at the l level above the leaf level must meet the constraint

$$mp^l \geq k.$$

After rearranging the terms in the above equation, we have

$$l \leq \frac{\log\left(\frac{m}{k}\right)}{\log\left(\frac{1}{p}\right)}.$$

Because, the root prototype is at the level $\log_2 n$, where n is the number of samples above the leaf level, we have

$$n \leq 2^{\frac{\log\left(\frac{m}{k}\right)}{\log\left(\frac{1}{p}\right)}}. \tag{12}$$

Equation (12) gives us the limitation of samples for our algorithm, if we impose the constraint that the root prototype must have at least k corners.

A. *Multilevel Approximation of Contour Alignments*

For computational efficiency, the cost associated with finding the prototype of two contours at each iteration when a leaf contour is deleted and reinserted into a tree must be small. In the following, we show that aligning two contours of the internal nodes in the tree can be approximated by aligning a sequence of subcontours. The approximation reduces the complexity of aligning two contours.

When a leaf node is removed from (or inserted into) a tree, we need to find the new contour on the path from the deleted leaf (or inserted leaf) to the root. If the prototype contour is generated from the contours of the internal nodes, then, based on the current alignments in the tree, we can apply a sequence of subcontour alignments to obtain a good approximation of the new alignment. Aligning a sequence of subcontours reduces the complexity of aligning whole contours from $\mathcal{O}(c^2)$ to $\mathcal{O}(\sum_i c_i^2)$, where $c = \sum_i c_i$ and c_i is the number of the corner in the ith subcontour.

Here, we describe a typical example of the approximation to demonstrate its efficiency. The formulation is given afterwards. We use PS(A, B) to represent the prototype of two contours A and B. In our binary tree, a PS associated with a parent node is the prototype of the contours associated with its children nodes. Since the PS is the mean of the aligned contours, it is coarser than any contour of a child node. As shown in Fig. 4(a), once A is deleted, B is substituted for P. Since B is a finer structure than P, removing a leaf contour is the same as replacing the PS with a finer structure than the previous PS. The node Q is also modified to Q' in a similar way. Thus, Q' is also finer than Q, since Q' is PS(B, C), Q is PS(P, C), and B is finer than P. Thus, removing a leaf from a tree corresponds to replacing the PS in any internal node on the path from the root of the tree to the leaf with a finer PS.

On the other hand, as shown in Fig. 4(b), if C is inserted, then $Q = $ PS(A, C) is substituted for A, which is finer than Q.

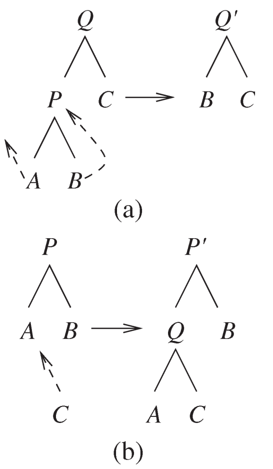


Fig. 4. (a) Node A is deleted from the tree. (b) Node C is inserted into the tree.

As a result, inserting a leaf into a tree corresponds to replacing the PS of each internal node on the path from the root of the tree to the leaf with a coarser structure than the previous PS.

Fig. 5 gives a simple example of this case, where optimum contour matching at any internal node in a tree can be obtained by subcontour alignments. Fig. 5(a) shows that B replaces P after A is deleted. Since b_1 matches p_1 and p_1 matches c_1, we have an approximation in that b_1 matches c_1. Similarly, since b_3 matches p_2 and p_2 matches c_4, we can deduce by approximation that b_3 matches c_4. Note that all the matches between B and C in Q are retained in Q'. The subcontours from b_1 to b_3 and from c_1 to c_4 are realigned to give a new match between b_2 and c_2 in Q'. Fig. 5(b) shows that subcontour alignment is applied to the subcontours between b_1 and b_4 in B, and q_1 and q_2 in Q.

We now formulate the above example. Let A and B be two contours. As defined in Section II-A, $A \sim B = \{(a_1,b_1),(a_2,b_2),\ldots,(a_{n_P},b_{n_P})\}$ denotes a legal alignment of matched corners of A and B. We make the following basic assumptions in order to implement our approximation:

1) **BA 1.** If $P = \mathrm{PS}(A,B)$ and $A \sim B = \{(a_1,b_1),(a_2,b_2),\ldots,(a_{n_P},b_{n_P})\}$, then $P \sim A = \{(p_1,a_1),(p_2,a_2),\ldots,(p_{n_P},a_{n_P})\}$ and $P \sim B = \{(p_1,b_1),(p_2,b_2),\ldots,(p_{n_P},b_{n_P})\}$. This assumption indicates that the match between the corners of the parent and those of its children can be derived from the match of the children.
2) **BA 2.** If $P \sim A = \{(p_1,a_1),(p_2,a_2),\ldots,(p_{n_P},a_{n_P})\}$ and $P \sim B = \{(p_1,b_1),(p_2,b_2),\ldots,(p_{n_P},b_{n_P})\}$, then $A \sim B = \{(a_1,b_1),(a_2,b_2),\ldots,(a_{n_P},b_{n_P})\}$. Thus, if $P \sim A$ and $P \sim B$, then $A \sim B$.

Illustrations of these two basic assumptions are given in Fig. 6. We now show that by applying **BA 1** and **BA 2**, we can locally update the contour of each internal node in a tree when a leaf is either inserted into, or deleted from, the tree.

Deleting a Node: Without loss of generality, let us use Fig. 4(a) as our example of deleting a node. In the left tree, we have $P = \mathrm{PS}(A,B)$ with $A \sim B$, and $Q = \mathrm{PS}(P,C)$ with $P \sim C$. However, if A is deleted, then we have the following.

1) B is substituted for $P = \mathrm{PS}(A,B)$.
2) The new root $Q' = \mathrm{PS}(B,C)$ can be obtained as follows. From $P = \mathrm{PS}(A,B)$ and $A \sim B$, by applying **BA 1**, we can derive $P \sim A$ and $P \sim B$. Because $Q = \mathrm{PS}(P,C)$ in the left tree, we have $P \sim C$. Then, from $P \sim C$ and $P \sim B$, we can, according to **BA 2**, derive that $B \sim C$; so, $Q' = (B,C)$.

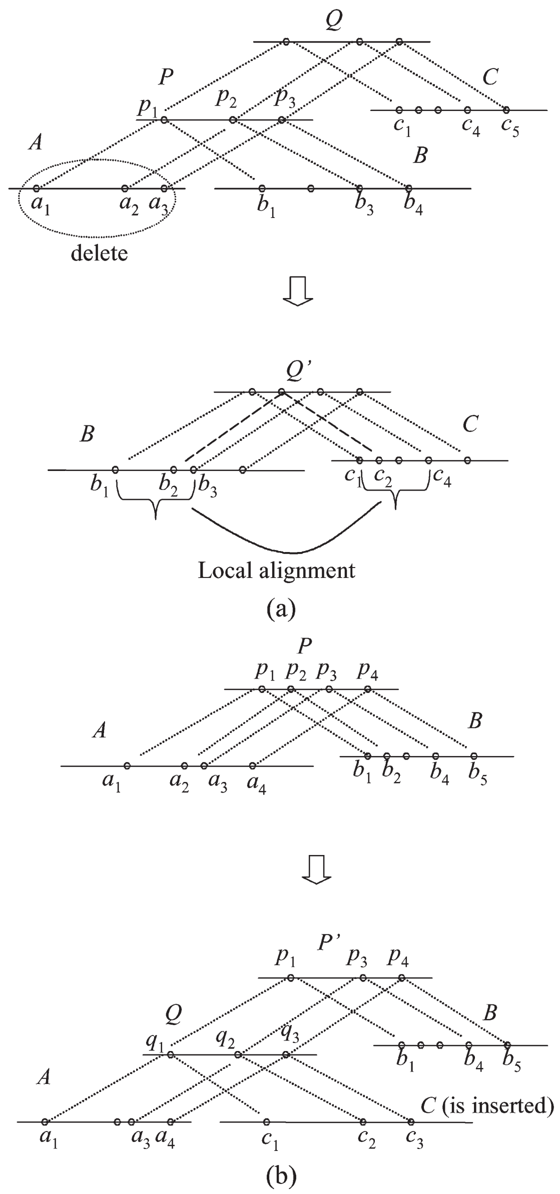


Fig. 5. (a) A is removed. Q' is obtained by aligning the subcontours between b_1 and b_3 in B, and c_1 and c_4 in C. (b) C is inserted. P' is obtained by aligning the subcontours between q_1 and q_2 in Q, and b_1 and b_4 in B.

Using the above procedure, the deletion of a node A can be efficiently implemented by subcontour alignment. Let us suppose that $A \sim B = \{(a_{u(1)},b_{v(1)}),(a_{u(2)},b_{v(2)}),\ldots\}$. Then, because $P = \mathrm{PS}(A,B)$, by applying **BA 1**, we have

$$P \sim B = \left\{\left(p_1,b_{v(1)}\right),\left(p_2,b_{v(2)}\right),\ldots\right\}.$$

Let us also suppose that $\{p_{w(1)},p_{w(2)},\ldots\}$ is a subsequence of $\{p_1,p_2,\ldots\}$, w is an increasing function, and $P \sim C = \{(p_{w(1)},c_{y(1)}),(p_{w(2)},c_{y(2)}),\ldots\}$. By applying **BA 2**, we can derive from $P \sim C$ and $P \sim B$ that

$$B \sim C = \left\{\left(b_{v(w(1))},c_{y(1)}\right),\left(b_{v(w(2))},c_{y(2)}\right),\ldots\right\}.$$

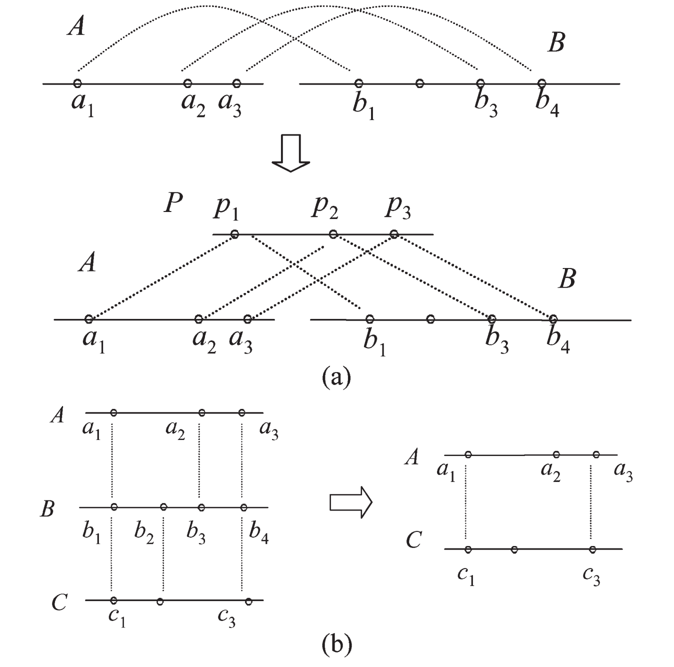


Fig. 6. Illustration of our two basic assumptions. (a) **BA 1**. If $A \sim B = \{(a_1, b_1), (a_2, b_3), (a_3, b_4)\}$, then $P \sim A = \{(p_1, a_1), (p_2, a_2), (p_3, a_3)\}$, and $P \sim B = \{(p_1, b_1), (p_2, b_3), (p_3, b_4)\}$. (b) **BA 2**. If $A \sim B = \{(a_1, b_1), (a_2, b_3), (a_3, b_4)\}$, and $B \sim C = \{(b_1, c_1), (b_2, c_2), (b_4, c_3)\}$, then $A \sim C = \{(a_1, c_1), (a_3, c_3)\}$.

To obtain $P' = \text{PS}(B, C)$, we only align the subcontours segmented by corners $b_{v(w(i))}$ and $b_{v(w(i+1))}$ in B and corners $c_{y(i)}$ and $c_{y(i+1)}$ in C for all i. Thus, we use a sequence of subcontour alignments to obtain the contour Q'. Note that according to (8)–(10), the alignment of two subcontours can be efficiently implemented by dynamic programming.

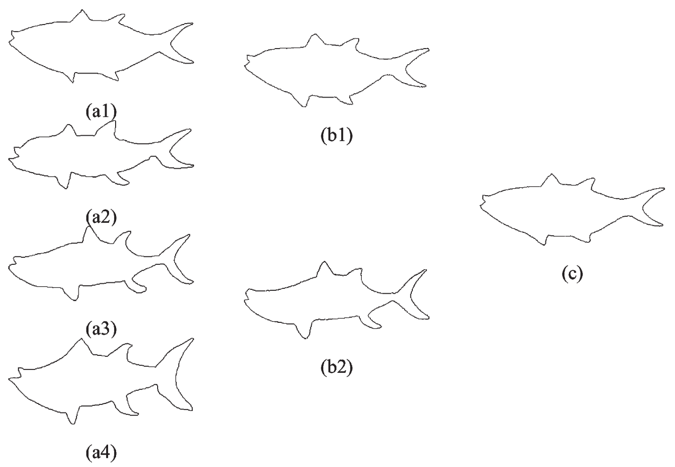
Inserting a Node: Without loss of generality, let us use Fig. 4(b) as our example of inserting a node. In the left tree, we have $A \sim B$. However, if C is inserted, then we have the following.

1) Let $Q = \text{PS}(A, C)$, and compute $A \sim C$ by aligning contours A and C. Because **BA 1** and **BA 2** cannot be applied in this case, A and C must be aligned by the two-contour alignment algorithm.
2) The contour for the new root P' can be obtained as follows:According to **BA 1**, we know $Q \sim A$; and according to **BA 2**, we can find $Q \sim B$ by $Q \sim A$ and $A \sim B$. Thus, if C is inserted, we have P'.

The second step in the above procedure can be efficiently implemented by a sequence of subcontour alignments. Let us suppose that $A \sim B = \{(a_{u(1)}, b_{v(1)}), (a_{u(2)}, b_{v(2)}), \ldots\}$. For convenience, we introduce a function z so that $z(u(i)) = v(i)$ for all i. Because $Q = \text{PS}(A, C)$, by applying **BA 1**, we have

$$Q \sim A = \left\{ \left(q_1, a_{w(1)}\right), \left(q_2, a_{w(2)}\right), \ldots \right\}.$$

Let us suppose that y is an increasing function and $\{q_{y(1)}, q_{y(2)}, \ldots\}$ is a subsequence of $\{q_1, q_2, \ldots\}$ so that



Prototype	a1	a2	a3	a4	Total Distortion	Mean
a1	0	66	72	80	218	72.6
a2	66	0	92	113	271	90
a3	72	92	0	87	251	83
a4	80	113	87	0	280	93
b1	47.6	30	124.9	126.7	329.2	82.3
b2	71.1	81.5	44.8	47.6	245	61.25
c	12	91.8	66.6	70.2	240.6	60.15

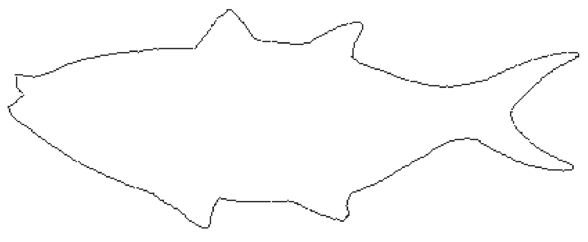
Fig. 7. Hierarchical binary tree that builds the prototype of the four sample-fish contours shown in the left column. (b1) is the prototype of (a1) and (a2); (b2) is the prototype of (a3) and (a4); and (c) is the prototype of all the contours. The table measures the distortion between a candidate prototype in the first column and a sample, either a_1, a_2, a_3, or a_4. The mean distortion is obtained by dividing the total distortion by 3 if the candidate is one of the samples, or by 4 if it is not in the sample set. Note that prototype (c) gives the smallest mean distortion.

$\{a_{w(y(1))}, a_{w(y(2))}, \ldots\}$ is a subsequence of $\{a_{u(1)}, a_{u(2)}, \ldots\}$. By applying **BA 2**, from $Q \sim A$ and $A \sim B$ we can derive

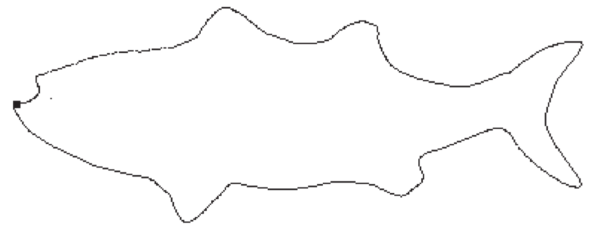
$$Q \sim B = \left\{ \left(q_{y(1)}, b_{z(w(y(1)))}\right), \left(q_{y(2)}, b_{z(w(y(2)))}\right), \ldots \right\}.$$

To obtain $Q' = \text{PS}(B, C)$, for all i, we only align the subcontours segmented by corners $q_{y(i)}$ and $q_{y(i+1)}$ in Q, and corners $b_{z(w(y(i)))}$ and $b_{z(w(y(i+1)))}$ in B. Thus, we use a sequence of subcontour alignments to obtain the contour Q'.

Complexity Analysis: The complexity of the proposed approximation algorithm depends on the distribution of corners in a sequence of random trees. A complete complexity analysis of the approximation algorithm is difficult. Here, we perform a simplified analysis for a special case in which we assume that the probability of a corner being an end point of a subcontour is p. Thus, the number of subcontours in a contour forms a binomial distribution, with the expected number mp. Also, the expected number of corners in a subcontour is $m/mp = 1/p$. Therefore, the expected complexity of aligning a sequence of mp subcontours is $\mathcal{O}(\sum_{i=1}^{mp}(1/p)^2 N) = \mathcal{O}((m/p)N)$, where N is the number of sample points of all contours. As noted previously, at each iteration, the algorithm prototype takes at most $\log_2 n$ two-contour alignments, in which the alignment of two leaves needs whole contours. The remainder can be approximated by a sequence of subcontour alignments. Thus,



(a)



(b)

Node	a1	a2	a3	a4	b1	b2	c
Corner number	22	26	20	24	14	14	6

(c)

Level	1	2	3
Average Corner number	23	14	6

(d)

Fig. 8. (a) Our prototype fish. (b) The simple mean fish obtained by averaging the four sample contours after aligning the marked corner and normalizing the lengths of the contours. Our prototype preserves corners, but in (b) all the corners, except the marked corner, are smoothed. (c) and (d), respectively, give the number of corners of the fishes in the final tree and the average number of corners of the fishes at each level in Fig. 7.

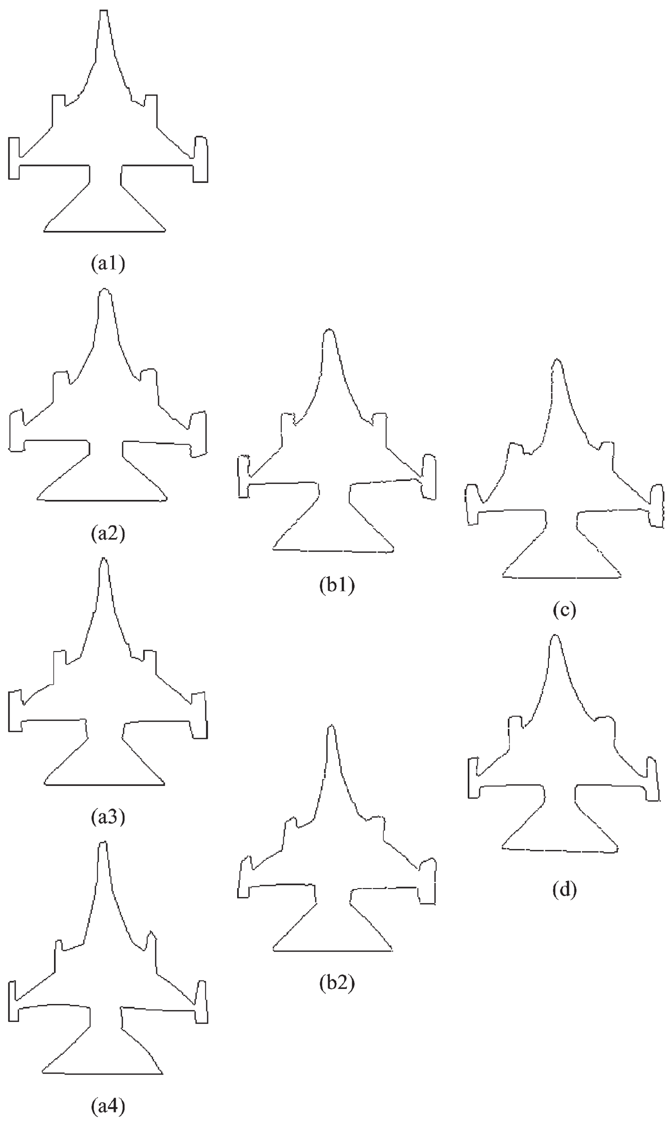


Fig. 9. Hierarchical binary tree that builds the prototype of the four sample-fighter contours shown in the left column. (b1) is the prototype of (a1) and (a2); (b2) is the prototype of (a3) and (a4); (c) is the prototype of all the contours; and (d) is the prototype generated by using fewer corners in the samples.

with the approximation, the complexity of aligning $\log_2 n$ pairs of contours is reduced from $\mathcal{O}(m^2 N \log_2 n)$ to $\mathcal{O}(m^2 N + (m/p) N \log_2 n)$.

IV. Experimental Results

In this section, we demonstrate our method by applying our prototype-generation algorithm to a set of planar patterns. Fig. 7 shows an example of our hierarchical binary tree of sample fishes. The left column of Fig. 7 is the lowest level of the tree and shows four sample-fish contours. Our subjects used a marker to hand draw the contours of the fishes on paper, without constraints on size. We then used a scanner to digitize and process them into binary images. Before applying our prototype-generation algorithm, we had to preprocess the binary images to extract the corners of their contours. To detect corners, we used a chain-code algorithm to extract a contour and a singularity-detection algorithm from the modulus maxima of wavelet coefficients of the contour [2], [13], [18]. There are many other corner-detection methods [9], [12], [21] that could be used. The middle column in Fig. 7 shows the intermediate prototypes of the four fishes. Each fish in the middle column is the prototype of two corresponding fishes in the left column. The right column in Fig. 7 is the prototype fish of our sample fishes. The table in the figure gives the dissimilarity distance between a candidate prototype fish in the first column and a sample fish: either a_1, a_2, a_3, or a_4. The fifth column is the total distortion between the candidate fish and the sample set. If the candidate prototype is in the sample set, the mean distortion is obtained by dividing the total distortion by 3 to exclude bias due to self-matching. If the candidate prototype is either b_1, b_2, or c, the total distortion is divided by 4, i.e., the size of the sample set, to obtain the mean distortion. Note that our root prototype gives the smallest value.

Fig. 8(a) is our prototype, and Fig. 8(b) is obtained by taking the average of the four sample-fish contours. We first choose the same starting point for each fish. We then align the chosen point in each fish's contour, normalize the contours so they are equal to the mean of the contours of the four fishes, and take the average of the normalized contours to obtain Fig. 8(b). Because our algorithm preserves the corners in the sample fishes, our prototype has a few sharp corners; however, in other methods,

Prototype	a1	a2	a3	a4	Total Distortion	Mean
a1	0	66.2	61.5	56.7	184.4	61.5
a2	66.2	0	90.2	82.1	238.5	79.5
a3	61.5	90.2	0	91.3	243	81.0
a4	56.7	82.3	91.3	0	230.3	76.8
b1	37.6	37.9	62	62.3	199.8	50
b2	48.2	77.8	46	37.1	209.1	52.3
c	31.5	56.1	51.5	56.4	195.5	48.9

(a)

Node	a1	a2	a3	a4	b1	b2	c
Corner number	27	28	29	27	23	19	15

(b)

level	1	2	3
average corner number	27.5	21	15

(c)

level	1	2	3
average corner number	21	17.5	13

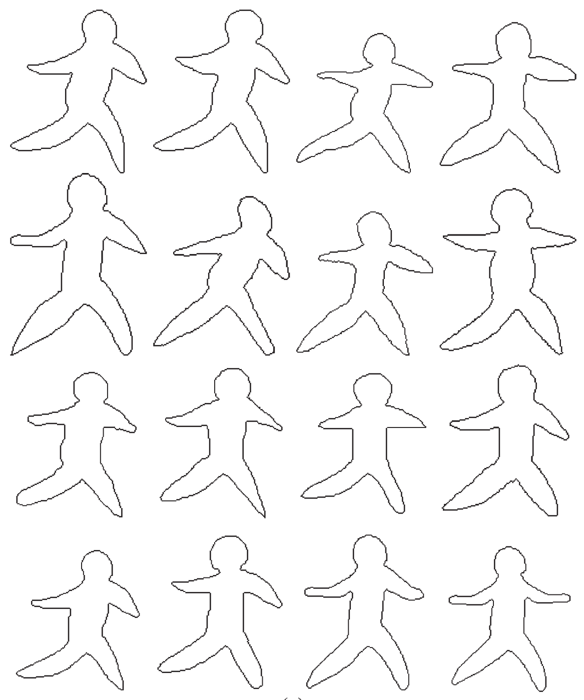
(d)

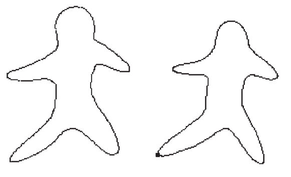
Fig. 10. (a) Distortion measured between a candidate prototype in the first column and a sample, either a_1, a_2, a_3, or a_4, in the first row. The mean distortion is obtained by dividing the total distortion by 3 if the candidate is one of the samples, or by 4 if it is not in the sample set. Note that prototype (c) gives the smallest mean distortion. (b) and (c), respectively, give the number of corners of the fighters in the final tree and the average number of corners of the fighters at each level in Fig. 9. (d) gives the number of corners at each level when the corners of samples are reduced for the prototype in Fig. 9(d).

all the corners, except the marked corner, become smooth. The average number of corners of the prototypes at each level provides a useful index to measure the coarseness of the prototypes at that level. Note that the number of corners of each prototype is measured in Fig. 8(c), while the average number of corners at each level is given in Fig. 8(d). Note that the number of corners decreases as the tree level increases. From Fig. 8(d), we can deduce that the probability that a corner exists at the next level is $p = 0.5$. p is an important index, because it can be used to derive the upper bound of the number of samples in (12), and to estimate the complexity of our approximation algorithm in Section III-A.

Figs. 9 and 10 demonstrate another example of our prototype generation. Fig. 9(c) is the prototype of the fighters, which also generates the smallest distortion, as shown in Fig. 10(a). Meanwhile, Fig. 10(c) gives the average number of corners of the prototype fighters at each level. The average number of corners at level 1 provides an approximate estimation of the complexity of the samples. The number of corners of the fighters at level 1 is larger than that of the fishes at the same level. This indicates that the fighter example has a more complicated structure than the fish example. To examine the quality of our prototypes for missing corners, we reduced the detected corners of the samples. For this case, the number of detected corners at each level is given in Fig. 10(d), while the corresponding prototype is shown in Fig. 9(d).

Fig. 11 demonstrates another example of prototype generation in which Fig. 11(b) is the prototype of the 16 hand-drawn samples in Fig. 11(a). The corner features of the samples are preserved in our prototype [see Fig. 11(b)]. Fig. 11(c) is obtained by taking the mean of the samples without imposing any constraints. We observe that the corners on the head of Fig. 11(c) disappear. Fig. 11(d) gives the average number of corners of the prototype at each level. The probability that a corner exists at the next highest level (above) is $p = 0.6$.


(a)


(b) (c)

Level	1	2	3	4	5
Average corner number	16.3	13.4	7.5	5	2

(d)

Fig. 11. (a) Sixteen hand-drawn samples. (b) Our prototype. (c) Simple mean in which all contours are aligned at the marked point. (d) Average number of corners at each level of the final binary tree. The samples are at level 1 and the prototype (b) is at level 5.

Finally, we present a preliminary result of a pattern-recognition application that uses a prototype as the representative of a set of objects. Readers should refer to [3], [8], [15], and [19] for other applications of prototypes. We experimented on the application of prototype generation in planar-shape recognition, and chose nine fish contours as the reference template. We asked our subjects to draw these contours on transparencies, and then scanned the contours as images. The sizes of the contours were not constrained. Our subjects were then divided into two groups, A and B, each of which had 20 members. Group A drew contours for fish1 to fish5, while group B drew contours for fish6 to fish9. Four subjects were chosen from each group to generate the prototype fish images. Fig. 12 shows the prototype contours of all the fishes.

We then tested the performance of our fish prototype. In the test, all the contours, including the contours used to generate prototype fishes, were used as query images. Each query fish was compared by measuring the dissimilarity between it and the nine prototype shapes. In the next stage, each fish was

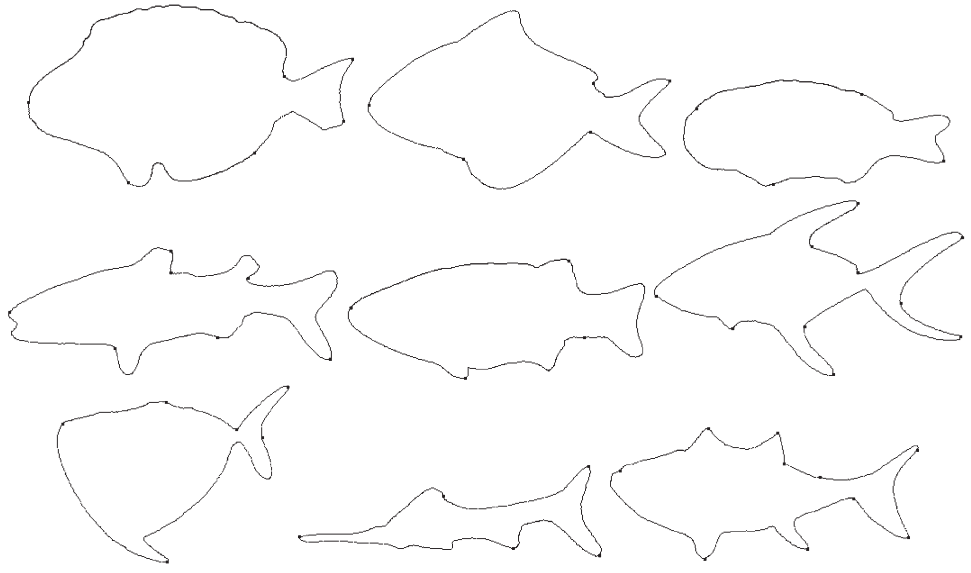


Fig. 12. Prototype fishes 1, 2, and 3 are shown in the first row; 4, 5, and 6 are in the second row; and 7, 8, and 9 are in the third row.

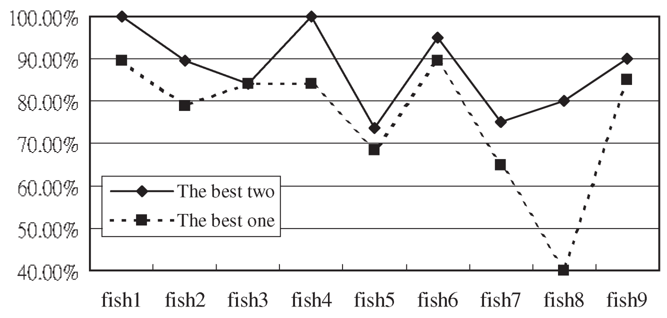


Fig. 13. "The best one" is the curve of detection probability indicating that the correct fish is the top candidate. The curve "the best two" is the detection probability that the correct fish is one of the top two candidates.

assigned to the prototype number whose dissimilarity measure was the minimum among all nine distortions. The detection probability, denoted as the best one, is plotted in Fig. 13. Another measurement is to assign a query fish to the two best prototype fishes that it matches. This yields the other plot (denoted as the best two) in the same figure. One can see that the recognition rate of all fishes, except fish5 and fish7 in the best two, is approximately 80%. This result indicates that the prototype could be used in unsupervised pattern recognition to distinguish different objects.

V. Conclusion

A prototype is defined as an object that satisfies the constraints and minimizes the distortion between similar sample objects. The constraints are important features of the sample shapes that the prototype object must retain. We use this approach to obtain a prototype of hand-drawn planar objects. In addition, we show that a good approximation of our prototype can be obtained efficiently by applying a random algorithm to obtain the contours of a hierarchical binary tree. We also derive an upper bound on the number of samples in order to produce a prototype whose corners are imposed as a constraint. This method can be extended to align multiple deoxyribonucleic-acid (DNA) sequences and other applications. There are, however, some unsolved issues, such as 1) whether the prototype derived by using the mean distortion matches that derived by a perceptual quality measurement and 2) whether our algorithm would be robust if a corner was missing. We pose these interesting issues for further study.

Appendix

Let $c_1(t)$, $c_2(t)$, and $z(t)$ be planar curves, $0 \le t \le 1$ and the end points of these curves be aligned. That is, $c_1(0) = c_2(0) = z(0)$, and $c_1(1) = c_2(1) = z(1)$. Then

$$u(t) = \arg\min_{z(t)} \left(d\left(z(t), c_1(t)\right) + d\left(z(t), c_2(t)\right)\right)$$

where

$$u(t) = \frac{c_1(t) + c_2(t)}{2}.$$

Proof: For any given $t \in [0,1]$, the value that minimizes $(\mathcal{N}z(t) - \mathcal{N}c_1(t))^2 + (\mathcal{N}z(t) - \mathcal{N}c_2(t))^2$ can be obtained by taking the derivative of the above with respect to $\mathcal{N}z(t)$ and setting the result to zero. This gives us the optimal curve

$$\mathcal{N}u(t) = \frac{\mathcal{N}c_1(t) + \mathcal{N}c_2(t)}{2}.$$

Let $\mathcal{N}^{-1}$ be the inverse similarity transform that takes $[0\ 0]^{\mathrm{T}}$ and $[1\ 0]^{\mathrm{T}}$ to $c_1(0)$, and $c_1(1)$, respectively. Then, by applying $\mathcal{N}^{-1}$ to $\mathcal{N}u(t)$, we obtain the optimal curve $u(t)$. ■

Acknowledgment

The authors would like to express their gratitude to the reviewers for their invaluable suggestions, especially Reviewer 1 whose insights helped them improve their random-algorithm prototype.

REFERENCES

[1] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[2] H. Asada and M. Brady, "The curvature primal sketch," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-8, no. 1, pp. 2–14, Jan. 1986.

[3] S. E. Chen and R. E. Parent, "Shape averaging and its applications to industrial design," *IEEE Comput. Graph. Appl.*, vol. 9, no. 1, pp. 47–54, Jan. 1989.

[4] J. H. Connell and M. Brady, "Generating and generalizing models of visual objects," *Artif. Intell.*, vol. 31, no. 2, pp. 159–183, Feb. 1987.

[5] T. Cootes, C. Taylor, and J. Graham, "Active shape models—Their training and applications," *Comput. Vis. Image Underst.*, vol. 61, no. 1, pp. 38–59, Jan. 1995.

[6] C. de la Higuera and F. Casacuberta, "Topology of strings: Median string is NP-complete," *Theor. Comp. Sci.*, vol. 230, no. 1/2, pp. 39–48, Jan. 2000.

[7] X. Jiang, A. Münger, and H. Bunke, "Synthesis of representative graphical symbols by computing generalized median graph," in *Lecture Notes on Computer Science*, vol. 1941. London, UK: Springer-Verlag, 2000, pp. 183–192.

[8] R. S. Kashi, P. Bhoj-Kavde, R. S. Nowakowski, and T. V. Papathomas, "2-D shape representation and averaging using normalized wavelet descriptors," *Simulation*, vol. 66, no. 3, pp. 164–178, Mar. 1996.

[9] J. S. Lee, Y. N. Sun, and C. H. Chen, "Multiscale corner detection by using wavelet transform," *IEEE Trans. Image Process.*, vol. 4, no. 1, pp. 100–104, Jan. 1995.

[10] M. Leyton, "A process-grammar for shape," *Artif. Intell.*, vol. 34, no. 2, pp. 213–247, Mar. 1988.

[11] R. S. Michalski, "Pattern recognition as rule-guided inductive inference," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-2, no. 4, pp. 349–361, Jul. 1980.

[12] F. Mokhtarian and A. Mackworth, "A theory of multiscale, curvature-based shape representation for planar curves," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 8, pp. 789–805, Aug. 1992.

[13] S. Mallat and W. L. Hwang, "Singularity detection and processing with wavelets," *IEEE Trans. Inf. Theory*, vol. 38, no. 2, pp. 617–643, Mar. 1992.

[14] H. Ney and S. Ortmanns, "Dynamic programming search for continuous speech recognition," *IEEE Signal Process. Mag.*, vol. 16, no. 5, pp. 64–83, Sep. 1999.

[15] G. Sánchez, J. Lladós, and K. Tombre, "A mean string algorithm to compute the average among a set of 2D shapes," *Pattern Recogn. Lett.*, vol. 23, no. 1–3, pp. 203–213, Jan. 2002.

[16] R. Singh and N. P. Papanikolopoulos, "Planar shape recognition by shape morphing," *Pattern Recogn.*, vol. 33, no. 10, pp. 1683–1699, Oct. 2000.

[17] T. B. Sebastian, P. N. Klein, and B. B. Kimia, "On aligning curves," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 1, pp. 116–124, Jan. 2003.

[18] C. L. Tu, W. L. Hwang, and J. Ho, "Analysis of singularities from modulus maxima of complex wavelets," *IEEE Trans. Inf. Theory*, vol. 51, no. 3, pp. 1049–1062, Mar. 2005.

[19] N. Ueda and S. Suzuki, "Learning visual models from shape contours using multiscale convex/concave structure matching," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, no. 4, pp. 337–352, Apr. 1993.

[20] E. Valveny and E. Martí, "Learning of structural descriptions of graphic symbols using deformable template matching," in *IEEE Int. Conf. Document Analysis and Recognition (ICDAR)*, Seattle, WA, 2001, pp. 455–459.

[21] M. Worrinng and A. W. M. Smeulders, "Digital curvature estimation," *CVGIP, Image Underst.*, vol. 58, no. 3, pp. 366–382, Nov. 1993.

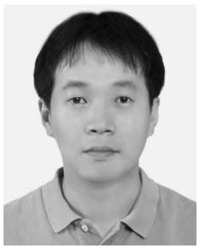
Wen-Yao Chen (S'05) was born in Taiwan, R.O.C. He received the B.S. and M.S. degrees in mathematics from National Central University, Taoyuan, Taiwan, in 1990 and 1992, respectively. He is currently working toward the Ph.D. degree in electrical engineering at National Tsing Hua University, Hsinchu, Taiwan.

He was formerly a Research Assistant at the Institute of Information Science, Academia Sinica, Taipei, Taiwan. His current research interest, and the subject of his doctoral thesis, is error-correction codes.

Wen-Liang Hwang (M'96–SM'05) received the B.S. degree in nuclear engineering from National Tsing Hua University, Hsinchu, Taiwan, R.O.C., in 1981, the M.S. degree in electrical engineering from the Polytechnic Institute of New York, Brooklyn, in 1989, and the Ph.D. degree in computer science from New York University, New York, in 1993.

He was a Postdoctoral Researcher with the Department of Mathematics, University of California, Irvine, in 1994. In January 1995, he became a Member of the Institute of Information Science, Academia Sinica, Taipei, Taiwan, where he is currently a Research Fellow. He is a coauthor of the book "*Practical Time-Frequency Analysis*" (Academic, 1998). His research interests include wavelet analysis, signal and image processing, and multimedia compression and transmission.

Dr. Hwang was awarded the Academia Sinica Research Award for Junior Research in 2001.



Tien-Ching Lin received the M.S. degree in applied mathematics from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1996. Currently, he is pursuing the Ph.D. degree with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan.

He is currently a Research Assistant at the Institute of Information Science, Academia Sinica, Taipei. His research interests include the design and analysis of algorithms, computational geometry, and bioinformatics.