Multiple Polyline to Polygon Matching

Mirela Tănase

Remco C. Veltkamp

Herman Haverkort

institute of information and computing sciences, utrecht university

technical report UU-CS-2005-017

www.cs.uu.nl

Multiple Polyline to Polygon Matching

Mirela Tănase[1] Remco C. Veltkamp[1] Herman Haverkort[2]
[1]Department of Information & Computing Sciences
Utrecht University, The Netherlands
[2] Department of Mathematics and Computing Science
TU Eindhoven, The Netherlands

Abstract

This paper addresses the partial shape matching problem, which helps identifying similarities even when a significant portion of one shape is occluded, or seriously distorted. We introduce a measure for computing the similarity between multiple polylines and a polygon, that can be computed in $O(km^2n^2)$ time with a straightforward dynamic programming algorithm. We then present a novel fast algorithm that runs in time $O(kmn \log mn)$. Here, m denotes the number of vertices in the polygon, and n is the total number of vertices in the k polylines that are matched against the polygon. The effectiveness of the similarity measure has been demonstrated in a part-based retrieval application with known ground-truth.

1 Introduction

The motivation for multiple polyline to polygon matching is twofold. Firstly, the matching of shapes has been done mostly by comparing them as a whole [3, 14, 18, 23, 24, 25, 26, 28, 31]. This fails when a significant part of one shape is occluded, or distorted by noise. In this paper, we address the partial shape matching problem, matching portions of two given shapes. Secondly, partial matching helps identifying similarities even when a significant portion of one shape boundary is occluded, or seriously distorted. It could also help in identifying similarities between contours of a non-rigid object in different configurations of its moving parts, like the contours of a sitting and a walking cat. Finally, partial matching helps alleviating the problem of unreliable object segmentation from images, over or undersegmentation, giving only partially correct contours.

Despite its usefulness, considerably less work has been done on partial shape matching than on global shape matching. One reason could be that partial shape matching is more involved than global shape matching, since it poses two more difficulties: identifying the portions of the shapes to be matched, and achieving invariance to scaling. If the shape representation is not scale invariant, shapes at different scales can be globally matched after scaling both shapes to the same length or the same area of the minimum bounding box. Such solutions work reasonably well in practice for many global similarity measures. When doing partial matching, however, it is unclear how the scaling should be done since there is no way of knowing in advance the relative magnitude of the matching portions of the two shapes. This can be seen in figure 1: any two of the three shapes in the figure should give a high score when partially matched but for each shape the scale at which this matching should be done depends on the portions of the shapes that match.

Contribution Firstly, we introduce a measure for computing the similarity between multiple polylines and a polygon. The polylines could for example be pieces of an object contour incompletely extracted from an image, or could be boundary parts in a decomposition of an object contour. The measure we propose is based on the turning function representation of the polylines and the polygon. We then derive a number of non-trivial properties of the similarity measure.

Secondly, based on these properties we characterize the optimal solution that leads to a straightforward $O(km^2n^2)$-time dynamic programming algorithm. We then present a novel $O(kmn \log mn)$-time algorithm. Here, m denotes the number of vertices in the polygon, and n is the total number of vertices in the k polylines that are matched against the polygon.

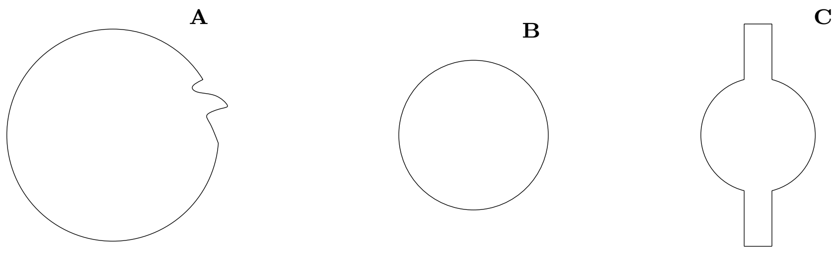


Figure 1: Scale invariance in partial matching is difficult.

Thirdly, we have experimented with a part-based retrieval application. Given a large collection of shapes and a query consisting of a set of polylines, we want to retrieve those shapes in the collection that best match our query. The set of polylines forming the query are boundary parts in a decomposition of a database shape – both this database shape and the parts in the query are selected by the user. The evaluation on the basis of a known ground-truth indicate that a part-based approach to matching improves the global matching performance for difficult categories of shapes.

2 Related Work

Arkin et al. [3] describe a metric for comparing two whole polygons that is invariant under translation, rotation and scaling. It is based on the L_2-distance between the turning functions of the two polygons, and can be computed in $O(mn \log mn)$ time, where m is the number of vertices in one polygon and n is the number of vertices in the other.

Most partial shape matching methods are based on computing local features of the contour, and then looking for correspondences between the features of the two shapes, for example points of high curvature [2, 17]. Other local features-based approaches make use of genetic algorithms [22], k-means clustering [7], or fuzzy relaxation techniques [20]. Such local features-based solutions work well when the matched subparts are almost equivalent up to a transformation such as translation, rotation or scaling, because for such subparts the sequences of local features are very similar. However, parts that we perceive as similar, may have quite different local features (different number of curvature extrema for example).

Geometric hashing [32] is a method that determines if there is a transformed subset of the query point set that matches a subset of a target point set, by building a hash table in transformation space. Also the Hausdorff distance [11] allows partial matching. It is defined for arbitrary non-empty bounded and closed sets A and B as the infimum of the distance of the points in A to B and the points in B to A. Both methods are designed for partial matching, but do not easily transform to our case of matching multiple polylines to a polygon.

Partially matching the turning angle function of two polylines under scaling, translation and rotation, can be done in time $O(m^2n^2)$ [8]. Given two matches with the same squared error, the match involving the longer part of the polylines has a lower dissimilarity. The dissimilarity measure is a function of the scale, rotation, and the shift of one polyline along the other. However, this works for only two single polylines.

Latecki et al [15], establish the best correspondence of parts in a decomposition of the matched shapes. The best correspondence between the maximal convex arcs of two simplified versions of the original shapes gives the partial similarity measure between the shapes. One serious drawback of this approach is the fact that the matching is done between parts of simplified shapes at “the appropriate evolution stage” [14]. How these evolution stages are identified is not indicated in their papers, though it certainly has an effect on the quality of the matching process. Also, by looking for correspondences between parts of the given shapes, false negatives could appear in the retrieval process. An example of such a case is in figure 2, where two parts of the query shape $\mathbf{Q}$ are denoted by $\mathbf{Q}_1$ and $\mathbf{Q}_2$, and two parts of the database shape $\mathbf{P}$ are denote by $\mathbf{P}_1$ and $\mathbf{P}_2$. Though $\mathbf{P}$ has a portion of $\mathbf{P}_1 \cup \mathbf{P}_2$ similar to a portion of $\mathbf{Q}_1 \cup \mathbf{Q}_2$, no correspondence among these four parts succeeds to recognize this. Considering different stages of the curve evolution (such that $\mathbf{P}_1 \cup \mathbf{P}_2$ and $\mathbf{Q}_1 \cup \mathbf{Q}_2$ constitute individual parts) does not solve the problem either. This is because the method used for matching two chains normalizes the two chains to unit length and thus no shifting of

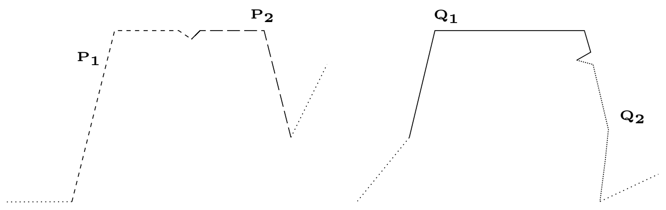


Figure 2: Partial matching based on correspondence of parts may fail to identify similarities.

one endpoint of one chain over the other is done. Also notice that if instead of making correspondences between parts, we shift the parts of one shape over the other shape and select those that give a close match, the similarity between the shapes in figure 2 is reported.

In our approach, we also use a decomposition into parts in order to identify the matching portions, but we do this only for one of the shapes. In the matching process between a (union of) part(s) and an undecomposed shape, the part(s) are shifted along the shape, and the optimal placement is computed based on their turning functions.

In the next section, we introduce a similarity measure for matching a union of possibly disjoint parts and a whole shape. This similarity measure is a turning angle function-based similarity, minimized over all possible shiftings of the endpoints of the parts over the shape, and also over all independent rotations of the parts. Since we allow the parts to rotate independently, this measure could capture the similarity between contours of non-rigid objects, with parts in different relative positions.

3 Polylines-to-Polygon Matching

We concentrate on the problem of matching an ordered set $\{P_1, P_2, \dots, P_k\}$ of k polylines against a polygon P. We want to compute how close an ordered set of polylines $\{P_1, P_2, \dots, P_k\}$ is to being part of the boundary of P in the given order in counter-clockwise direction around P (see figure 3). For this purpose, the polylines are rotated and shifted along the polygon P, in such a way that the pieces of the boundary of P "covered" by the k polylines are mutually disjoint except possibly at their endpoints. In this section we define a similarity measure for such a polylines-to-polygon matching. The measure is based on the turning function representation of the given polylines and it is invariant under translation and rotation. We first give an $O(km^2n^2)$-time dynamic programming algorithm for computing this similarity measure, where m is the number of vertices in P and n is the total number of vertices in the k polylines. We then refine this algorithm to obtain a running time of $O(kmn \log mn)$. Note that the fact that P is a polygon, and not an open polyline, comes from the intended application of part based shape retrieval.

3.1 Similarity between polyline and polygon

The *turning function* Θ_A of a polygon A measures the angle of the counterclockwise tangent with respect to a reference orientation as a function of the arc-length s, measured from some reference point on the boundary of A. It is a piecewise constant function, with jumps corresponding to the vertices of A. A rotation of A by an angle θ corresponds to a shifting of Θ_A over a distance θ in the vertical direction. Moving the location of the reference point $A(0)$ over a distance $t \in [0, l_A)$ along the boundary of A corresponds to shifting Θ_A horizontally over a distance t.

The distance between two polygons A and B is defined as the L_2 distance between their two turning functions Θ_A and Θ_B, minimized with respect to the vertical and horizontal shifts of these functions (in other words, minimized with respect to rotation and choice of reference point). More formally, suppose A and B are two polygons with perimeter length l_A and l_B, respectively, and the polygon B is placed over A in such a way that the reference point $B(0)$ of B coincides with point $A(t)$ at distance t along A from the reference point $A(0)$, and B is rotated clockwise by an angle θ with respect to the reference orientation. A pair $(t, \theta) \in \mathbb{R} \times \mathbb{R}$ will be referred to as a *placement* of B over A. The first component t of a placement (t, θ) is also called a horizontal shift, since it corresponds to a horizontal shifting of Θ_A over a distance t,

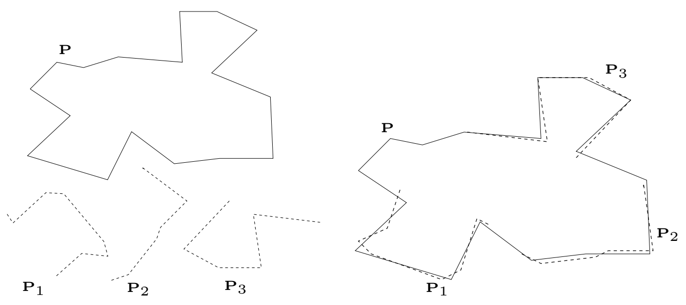


Figure 3: Matching an ordered set $\{P_1, P_2, P_3\}$ of polylines against a polygon P.

while the second component θ is also called a vertical shift, since it corresponds to a vertical shifting of Θ_A over a distance θ. We define the *quadratic similarity* $\mathsf{f}(A, B, t, \theta)$ *between A and B for a given placement (t, θ) of B over A*, as the square of the L_2-distance between their two turning functions Θ_A and Θ_B:

$$\mathsf{f}(A, B, t, \theta) = \int_0^{l_B} (\Theta_A(s + t) - \Theta_B(s) + \theta)^2 ds.$$

The similarity between two polygons A and B is then given by:

$$\min_{\theta \in \mathbb{R},\, t \in [0, l_A)} \sqrt{\mathsf{f}(A, B, t, \theta)}.$$

To achieve invariance under scaling, Arkin et al. [3] propose scaling the two polygons to unit length prior to the matching.

For measuring the difference between a polygon A and a polyline B the same measure can be used (for matching two polylines, some slight adaptations would be needed). Notice that, for the purpose of our part-based retrieval application, we want that a polyline B included in a polygon A to match perfectly this polygon, that is: their similarity should be zero. But if we scale A and B to the same length, their turning functions will no longer match perfectly. For this reason we do not scale the polyline or the polygon prior to the matching process. Thus, our similarity measure is not scale-invariant.

In a part-based retrieval application (see section 4) using a similarity measure based on the above quadratic similarity (see section 3.2), we achieve robustness to scaling by normalizing all shapes in the collection to the same diameter of their circumscribed circle. This is a reasonable solution for our collection, which does not contain occluded, or incomplete shapes. Moreover these shapes are object contours and not synthetic shapes.

3.2 Similarity between multiple polylines and a polygon

Let $\Theta : [0, l] \to \mathbb{R}$ be the turning function of a polygon P of m vertices, and of perimeter length l. Since P is a closed polyline, the domain of Θ can be easily extended to the entire real line, by $\Theta(s+l) = \Theta(s) + 2\pi$. Let $\{P_1, P_2, \ldots P_k\}$ be a set of polylines, and let $\Theta_j : [0, l_j] \to \mathbb{R}$ denote the turning function of the polyline P_j of length l_j. If P_j is made of n_j segments, Θ_j is piecewise-constant with $n_j - 1$ jumps.

For simplicity of exposition, $\mathsf{f}_j(t, \theta)$ denotes the quadratic similarity $\mathsf{f}_j(t, \theta) = \int_0^{l_j} (\Theta(s+t) - \Theta_j(s) + \theta)^2 ds$.

We assume the polylines $\{P_1, P_2, \ldots, P_k\}$ satisfy the condition $\sum_{j=1}^{k} l_j \leq l$. The similarity measure, which we denote by $\mathsf{d}(P_1, \ldots, P_k; P)$, is the square root of the sum of quadratic similarities f_j, minimized over all valid placements of $P_1, \ldots, P_k$ over P (or in other words, minimized over all valid horizontal and vertical shifts of their turning functions):

$$\mathsf{d}(P_1, \ldots, P_k; P) = \min_{\substack{\text{valid placements} \\ (t_1, \theta_1) \ldots (t_k, \theta_k)}} \left(\sum_{j=1}^{k} \mathsf{f}_j(t_j, \theta_j) \right)^{1/2}.$$

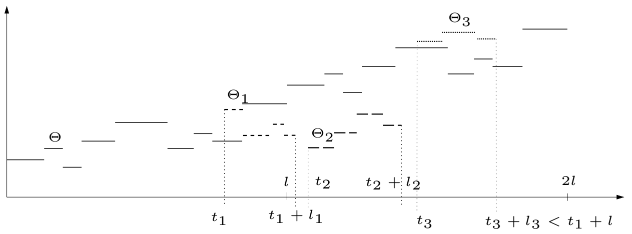


Figure 4: To compute $\mathsf{d}(P_1, \ldots, P_k; P)$ between the polylines $P_1, \ldots, P_3$ and the polygon P, we shift the turning functions Θ_1, Θ_2, and Θ_3 horizontally and vertically over Θ.

It remains to define what the valid placements are. The horizontal shifts $t_1, \ldots, t_k$ correspond to shiftings of the starting points of the polylines $P_1, \ldots, P_k$ along P. We require that the starting points of $P_1, \ldots, P_k$ are matched with points on the boundary of P in counterclockwise order around P, that is: $t_{j-1} \leq t_j$ for all $1 < j \leq k$, and $t_k \leq t_1 + l$. Furthermore, we require that the matched parts are disjoint (except possibly at their endpoints), sharpening the constraints to $t_{j-1} + l_{j-1} \leq t_j$ for all $1 < j \leq k$, and $t_k + l_k \leq t_1 + l$ (see figure 4).

The vertical shifts $\theta_1, \ldots, \theta_k$ correspond to rotations of the polylines $P_1, \ldots, P_k$ with respect to the reference orientation, and are independent of each other. Therefore, in an optimal placement the quadratic similarity between a particular polyline P_j and P depends only on the horizontal shift t_j, while the vertical shift must be optimal for the given horizontal shift. We can thus express the similarity between P_j and P for a given positioning t_j of the starting point of P_j over P as: $\mathsf{f}_j^*(t_j) = \min_{\theta \in \mathbb{R}} \mathsf{f}_j(t_j, \theta)$.

The similarity between the polylines $P_1, \ldots, P_k$ and the polygon P is thus:

$$\mathsf{d}(P_1, \ldots, P_k; P) = \min_{\substack{t_1 \in [0,l),\, t_2, \ldots, t_k \in [0,2l); \\ \forall j \in \{2, \ldots, k\}:\ t_{j-1} + l_{j-1} \leq t_j; \quad t_k + l_k \leq t_1 + l}} \left(\sum_{j=1}^{k} \mathsf{f}_j^*(t_j) \right)^{1/2}. \tag{1}$$

3.3 Properties of the similarity function

In this section we give a few properties of $\mathsf{f}_j^*(t)$, as functions of t, that constitute the basis of the algorithms for computing $\mathsf{d}(P_1, \ldots, P_k; P)$ in sections 3.5 and 3.6. We also give a simpler formulation of the optimization problem in the definition of $\mathsf{d}(P_1, \ldots, P_k; P)$. Arkin et al. [3] have shown that for any fixed t, the function $\mathsf{f}_j(t, \theta)$ is a quadratic convex function of θ. This implies that for a given t, the optimization problem $\min_{\theta \in \mathbb{R}} \mathsf{f}_j(t, \theta)$ has a unique solution, given by the root $\theta_j^*(t)$ of the equation $\partial \mathsf{f}_j(t, \theta)/\partial \theta = 0$. Since

$$\frac{\partial \mathsf{f}_j(t, \theta)}{\partial \theta} = \int_0^{l_j} 2(\Theta(s+t) - \Theta_j(s) + \theta) ds = \int_0^{l_j} 2(\Theta(s+t) - \Theta_j(s)) ds \ + \ 2\theta l_j, \tag{2}$$

we get:

Lemma 1 *For a given positioning t of the starting point of P_j over P, the rotation that minimizes the quadratic similarity between P_j and P is given by $\theta_j^*(t) = -\int_0^{l_j} (\Theta(s+t) - \Theta_j(s)) ds / l_j$.*

We now consider the properties of $\mathsf{f}_j^*(t) = \mathsf{f}_j(t, \theta_j^*(t))$, as a function of t.

Lemma 2 *The quadratic similarity $\mathsf{f}_j^*(t)$ has the following properties:*

i) it is periodic, with period l;

ii) it is piecewise quadratic, with mn_j breakpoints within any interval of length l; moreover, the parabolic pieces are concave.

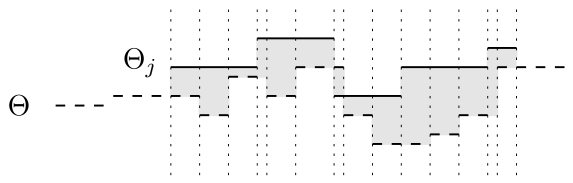


Figure 5: For given horizontal and vertical shifts of Θ_j over Θ, the discontinuities of Θ_j and Θ form a set of rectangular strips.

Proof: i) Since $\Theta(t+l) = 2\pi + \Theta(t)$, from lemma 1 we get that $\theta^*(t+l) = \theta^*(t) - 2\pi$. Thus

$$\mathsf{f}_j^*(t+l) = \int_0^{l_j} (\Theta(s+t+l) - \Theta_j(s) + \theta^*(t) - 2\pi)^2 ds = \mathsf{f}_j^*(t).$$

ii) Since $\mathsf{f}_j^*(t) = \mathsf{f}(P, P_j, t, \theta_j^*(t))\mathsf{f}_j(t, \theta_j^*(t))$, we have:

$$\begin{aligned} \mathsf{f}_j^*(t) = \int_0^{l_j} (\Theta(s+t) - \Theta_j(s) + \theta_j^*(t))^2 ds &= \int_0^{l_j} (\Theta(s+t) - \Theta_j(s))^2 ds \quad + \\ & \int_0^{l_j} 2(\Theta(s+t) - \Theta_j(s))\theta_j^*(t) ds \quad + \\ & \int_0^{l_j} (\theta_j^*(t))^2 ds \\ &= \int_0^{l_j} (\Theta(s+t) - \Theta_j(s))^2 ds + \\ & 2\theta_j^*(t) \int_0^{l_j} (\Theta(s+t) - \Theta_j(s)) ds + \\ & l_j\,(\theta_j^*(t))^2. \end{aligned}$$

Applying lemma 1 we get:

$$\mathsf{f}_j^*(t) = \int_0^{l_j} (\Theta(s+t) - \Theta_j(s))^2 ds - \frac{\left(\int_0^{l_j} (\Theta(s+t) - \Theta_j(s)) ds\right)^2}{l_j}. \tag{3}$$

For the rest of this proof we restrict our attention to the behaviour of f_j^* within an interval of length l. For a given value of the horizontal shift t, the discontinuities of Θ and Θ_j define a set S of rectangular strips. Each strip is defined by a pair of consecutive discontinuities of the two turning functions and is bounded from below and above by these functions.

As Θ_j is shifted horizontally with respect to Θ, there are at most mn_j critical events within an interval of length l where the configuration of the rectangular strips defined by Θ and Θ_j changes. These critical events are at values of t were a discontinuity of Θ_j coincides with a discontinuity of Θ. Inbetween these critical events, the height of a strip σ is a constant h_σ, while the width w_σ is a linear function of t with coefficient -1, 0 or 1.

For a given configuration of strips, the value of $\int_0^{l_j} (\Theta(s+t) - \Theta_j(s))^2 ds$ is thus given by the sum over the strips in the current configuration of the width of the strip times the square of its height:

$$\int_0^{l_j} (\Theta(s+t) - \Theta_j(s))^2 ds = \sum_{\sigma \in \mathsf{S}} w_\sigma(t) h_\sigma^2.$$

Similarly, we have:

$$\int_0^{l_j} (\Theta(s+t) - \Theta_j(s)) ds = \sum_{\sigma \in \mathsf{S}} w_\sigma(t) h_\sigma.$$

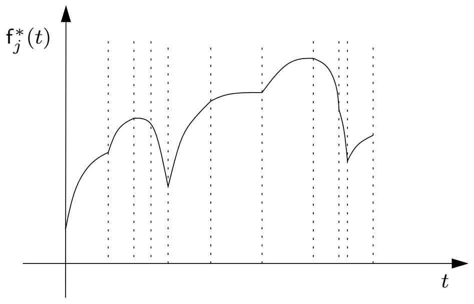


Figure 6: The quadratic similarity f_j^* is a piecewise quadratic function of t, with concave parabolic pieces.

And thus equation (3) can be rewritten as:

$$\mathsf{f}_j^*(t) \quad = \quad \sum_{\sigma \in \mathsf{S}} w_\sigma(t) h_\sigma^2 \quad - \quad \frac{1}{l_j} \left(\sum_{\sigma \in \mathsf{S}} w_\sigma(t) h_\sigma \right)^2 . \tag{4}$$

Since the functions w_σ are linear functions of t, the function f_j^* is a piecewise quadratic function of t, with at most mn_j breakpoints within an interval of length l. Moreover, since the coefficient of t^2 in equation (4) is negative, the parabolic pieces of f_j^* are concave (see figure 6). □

The following corollary indicates that in computing the minimum of the function f_j^*, it suffices to restrict our attention to a discrete set of at most mn_j points.

Corollary 1 *The local minima of the function f_j^* are among the breakpoints between its parabolic pieces.*

We now give a simpler formulation of the optimization problem in the definition of $\mathsf{d}(P_1, \ldots, P_k; P)$. In order to simplify the restrictions on t_j in equation (1), we define:

$$\bar{\mathsf{f}}_j(t) := \mathsf{f}_j^* \left(t + \sum_{i=1}^{j-1} l_i \right) .$$

In other words, the function $\bar{\mathsf{f}}_j$ is a copy of f_j^*, but shifted to the left with $\sum_{i=1}^{j-1} l_i$. Obviously, $\bar{\mathsf{f}}_j$ has the same properties as f_j^*, that is: it is a piecewise quadratic function of t that has its local minima in at most mn_j breakpoints in any interval of length l. With this simple transformation of the set of functions f_j^*, the optimization problem defining $\mathsf{d}(P_1, \ldots, P_k; P)$ becomes:

$$\mathsf{d}(P_1, \ldots, P_k; P) = \min_{\substack{t_1 \in [0,l),\, t_2, \ldots, t_k \in [0,2l); \\ \forall j \in \{2,\ldots,k\}:\ t_{j-1} \le t_j; \quad t_k \le t_1 + l_0}} \left(\sum_{j=1}^{k} \bar{\mathsf{f}}_j(t_j) \right)^{1/2} , \tag{5}$$

where $l_0 := l - \sum_{i=1}^{k} l_i$. Notice that if $(\bar{t}_1^*, \ldots, \bar{t}_k^*)$ is a solution to the optimization problem in equation (5), then $(t_1^*, \ldots, t_k^*)$, with $t_j^* := \bar{t}_j^* + \sum_{i=1}^{j-1} l_i$, is a solution to the optimization problem in equation (1).

3.4 Characterization of an optimal solution

In this section we characterize the structure of an optimal solution to the optimization problem in equation (5), and give a recursive definition of this solution. This definition forms the basis of a straightforward dynamic programming solution to the problem.

Let $(\bar{t}_1^*, \ldots, \bar{t}_k^*)$ be a solution to the optimization problem in equation (5).

Lemma 3 *The values of an optimal solution $(\overline{t}_1^*, \ldots, \overline{t}_k^*)$ are found in a discrete set of points $X \subset [0, 2l)$, which consists of the breakpoints of the functions $\overline{\mathsf{f}}_1, ..., \overline{\mathsf{f}}_k$, plus two copies of each breakpoint: one shifted left by l_0 and one shifted right by l_0.*

Proof: If $\overline{t}_{j-1}^* \neq \overline{t}_j^*$ and $\overline{t}_j^* \neq \overline{t}_{j+1}^*$, the constraints on the choice of $\overline{t}_1^*, ..., \overline{t}_k^*$ do not prohibit decreasing or increasing $\overline{t}_j^*$, thus $\overline{t}_j^*$ must give a local minimum of function $\overline{\mathsf{f}}_j$.

Notice now, that $\overline{t}_{j-1}^* = \overline{t}_j^*$ means that $t_j^* = t_{j-1}^* + l_{j-1}$, thus the positioning of the ending point of P_{j-1} and of the starting point of P_j coincide. Then, in the given optimal placement the polylines P_{j-1} and P_j are "glued" together. Similarly, if $\overline{t}_k^* = \overline{t}_1^* + l_0$, we have that P_k and P_1 are glued together.

For any $j \in \{1, ..., k\}$, let $\mathbf{G}_j$ be the maximal set of polylines containing P_j and glued together in the given optimal placement, that is: no polyline that is not in $\mathbf{G}_j$ is glued to a polyline in $\mathbf{G}_j$. Observe that $\mathbf{G}_j$ must have either the form $\{P_g, ..., P_h\}$ (with $j \in \{g, ..., h\}$) or the form $\{P_1, ..., P_h, P_g, ..., P_k\}$ (with $j \in \{1, ..., h, g, ..., k\}$), for some g and h in $\{1, ..., k\}$.

In the first case, we have $\overline{t}_i^* = \overline{t}_j^*$ for all $i \in \{g, ..., h\}$. Since $\mathbf{G}_j$ is a maximal set of polylines glued together, the constraints on the choice of $\overline{t}_1^*, ..., \overline{t}_k^*$ do not prohibit decreasing or increasing $\overline{t}_g^*, ..., \overline{t}_h^*$ simultaneously, so $\overline{t}_j^*$ must give a local minimum of the function $\overline{\mathsf{f}}_{gh} : \mathbb{R} \to \mathbb{R}$, $\overline{\mathsf{f}}_{gh}(t) = \sum_{i=g}^{h} \overline{\mathsf{f}}_i(t)$. This function is obviously piecewise quadratic, with concave parabolic pieces, and the breakpoints of $\overline{\mathsf{f}}_{gh}$ are the breakpoints of the constituent functions $\overline{\mathsf{f}}_g, ..., \overline{\mathsf{f}}_h$. Note that a local minimum of $\overline{\mathsf{f}}_{gh}$ may not be a local minimum of any of the constituent functions $\overline{\mathsf{f}}_i$ $(g \leq i \leq h)$, but nevertheless it will always be given by a breakpoint of one of the constituent functions.

In the second case, we can argue in a similar manner that $\overline{t}_j^*$ must give a local minimum of the function $\overline{\mathsf{f}}_{gh} : \mathbb{R} \to \mathbb{R}$, $\overline{\mathsf{f}}_{gh}(t) = \sum_{i=g}^{k} \overline{\mathsf{f}}_i(t) + \sum_{i=1}^{h} \overline{\mathsf{f}}_i(t - l_0)$, (when $j \in \{g, ..., k\}$) or the function $\overline{\mathsf{f}}_{gh} : \mathbb{R} \to \mathbb{R}$, $\overline{\mathsf{f}}_{gh}(t) = \sum_{i=g}^{k} \overline{\mathsf{f}}_i(t + l_0) + \sum_{i=1}^{h} \overline{\mathsf{f}}_i(t)$ (when $j \in \{1, ..., h\}$). Thus, $\overline{t}_j^*$ must be a breakpoint of one of the functions $\overline{\mathsf{f}}_g, ..., \overline{\mathsf{f}}_h$, or it must be such a breakpoint shifted left or right by l_0. □

We call a point in $[0, 2l)$, which is either a breakpoint of one of the functions $\overline{\mathsf{f}}_1, ..., \overline{\mathsf{f}}_k$, or such a breakpoint shifted left or right by l_0, a *critical point*. Since function $\overline{\mathsf{f}}_j$ has $2mn_j$ breakpoints, the total number of critical points in $[0, 2l)$ is at most $6m \sum_{i=1}^{j} n_i = 6mn$. Let $X = \{x_0, \ldots, x_{N-1}\}$ be the set of critical points in $[0, 2l)$.

With the observations above, the optimization problem we have to solve is:

$$\mathsf{d}(P_1, \ldots, P_k; P) = \min_{\substack{t_1, \ldots, t_k \in X \\ \forall j > 1 \,:\, t_{j-1} \leq t_j \,;\, t_k - t_1 \leq l_0}} \left(\sum_{j=1}^{k} \overline{\mathsf{f}}_j(t_j) \right)^{1/2} . \tag{6}$$

We now show that an optimal solution of this problem can be constructed recursively. For this purpose, we denote:

$$D[j, a, b] = \min_{\substack{t_1, \ldots, t_j \in X \\ x_a \leq t_1 \leq \ldots \leq t_j \leq x_b}} \sum_{i=1}^{j} \overline{\mathsf{f}}_i(t_i) \,, \tag{7}$$

where $j \in \{1, \ldots, k\}$, $a, b \in \{0, \ldots, N-1\}$, and $a \leq b$. Equation (7) describes the subproblem of matching the set $\{P_1, \ldots, P_j\}$ of j polylines to a subchain of P, starting at $P(x_a)$ and ending at $P(x_b + \sum_{i=1}^{j} l_i)$. We now show that $D[j, a, b]$ can be computed recursively. Let $(t_1^{\circledast}, \ldots t_j^{\circledast})$ be an optimal solution for $D[j, a, b]$. Regarding the value of $t_j^{\circledast}$ we distinguish two cases:

- $t_j^{\circledast} = x_b$, in which case $(t_1^{\circledast}, \ldots t_{j-1}^{\circledast})$ must be an optimal solution for $D[j-1, a, b]$, otherwise $(t_1^{\circledast}, \ldots t_j^{\circledast})$ would not give a minimum for $D[j, a, b]$; thus in this case, $D[j, a, b] = D[j-1, a, b] + \overline{\mathsf{f}}_j(x_b)$;

- $t_j^{\circledast} \neq x_b$, in which case $(t_1^{\circledast}, \ldots t_j^{\circledast})$ must be an optimal solution for $D[j, a, b-1]$; otherwise $(t_1^{\circledast}, \ldots t_j^{\circledast})$ would not give a minimum for $D[j, a, b]$; thus in this case $D[j, a, b] = D[j, a, b-1]$.

We can now conclude that :

$$D[j,a,b] = \min\left(D[j-1,a,b] + \bar{\mathsf{f}}_j(x_b),\ D[j,a,b-1]\right), \text{for j} \geq 1 \wedge \text{a} \leq \text{b}, \tag{8}$$

where the boundary cases are $D[0,a,b] = 0$ and $D[j,a,a-1]$ has no solution.

A solution of the optimization problem (6) is then given by

$$\mathsf{d}(P_1,\ldots,P_k;P) = \min_{x_a,x_b \in X,\ x_b - x_a \leq l_0} \sqrt{D[k,a,b]}\,. \tag{9}$$

3.5 A straightforward algorithm

Equations (8) and (9) lead to a straightforward dynamic programming algorithm for computing the similarity measure $\mathsf{d}(P_1,\ldots,P_k;P)$:

```
Algorithm SimpleCompute $\mathsf{d}(P_1,\ldots,P_k;P)$
1. Compute the set of critical points $X = \{x_0,\ldots,x_{N-1}\}$, and sort them.
2. For all $j \in \{1,\ldots,k\}$ and all $i \in \{0,\ldots,N-1\}$, evaluate $\bar{\mathsf{f}}_j(x_i)$.
3. $MIN \leftarrow \infty$
4. for $a \leftarrow 0$ to $N-1$ do
5.     for $j \leftarrow 1$ to $k$ do $D[j,a,a-1] \leftarrow \infty$
6.     $b \leftarrow a$
7.     while $x_b - x_a \leq l_0$ do
8.         $D[0,a,b] \leftarrow 0$
9.         for $j \leftarrow 1$ to $k$ do
10.            $D[j,a,b] \leftarrow \min\left(D[j-1,a,b] + \bar{\mathsf{f}}_j(x_b), D[j,a,b-1]\right)$
11.        $b \leftarrow b+1$
12.    $MIN \leftarrow \min(D[k,a,b-1], MIN)$.
13. return $\sqrt{MIN}$
```

Lemma 4 *The running time of algorithm SimpleCompute* $\mathsf{d}(P_1,\ldots,P_k;P)$ *is* $O(km^2n^2)$*, and the algorithm uses* $O(km^2n^2)$ *storage.*

Proof: Step 1 of this algorithm requires first identifying all values of t such that a vertex of a turning function Θ_j is aligned with a vertex of Θ. For a fixed vertex v of a turning function Θ_j, all m values of t such that v is aligned with a vertex of Θ can be identified in $O(m)$ time; the results for all n_j vertices of Θ_j can thus be collected in $O(mn_j)$ time. We then have to shift these values left by $\sum_{i=1}^{j-1} l_i$ and make copies of them shifted left and right by l_0. Overall, the identification of the set X of critical points takes $O(\sum_{j=1}^{k} mn_j) = O(mn)$, and they can be sorted in $O(mn\log(mn))$.

We now show that, for a particular value of j, evaluating $\bar{\mathsf{f}}_j(x_i)$, for all $i \in \{0,\ldots,N-1\}$ takes $O(mn)$. The algorithm is an adaptation of the algorithm used in [3], and we thus indicate only its main ideas. As equation (4) from the proof of lemma 2 indicates, the function $\bar{\mathsf{f}}_j(t)$ is the difference of two piecewise linear functions with the same set of breakpoints. Thus these functions can be completely determined by evaluating an initial value and the slope of each linear piece. It remains to indicate how to compute the slopes of these linear pieces.

Notice that when Θ_j is horizontally shifted over Θ such that the configuration S of the rectangular strips does not change, the only strips whose width is changing are those bounded by a jump of Θ and a jump of Θ_j. More precisely, let us denote by $\mathrm{S}_{\Theta\Theta_j}$ the set of rectangular strips in the current configuration S that are bounded left by a jump of Θ and right by a jump of Θ_j (see figure 7), by $\mathrm{S}_{\Theta_j\Theta}$ the set of rectangular strips that are bounded left by a jump of Θ_j and right by a jump of Θ, and by $\mathrm{S}_{\Theta\Theta}$ and $\mathrm{S}_{\Theta_j\Theta_j}$ the set of of rectangular strips that are bounded left and right by a jump of Θ and Θ_j, respectively. When shifting Θ_j such that the configuration S does not change, the widths of the strips in $\mathrm{S}_{\Theta_j\Theta_j}$ and $\mathrm{S}_{\Theta\Theta}$ do not change, while the width of those in $\mathrm{S}_{\Theta\Theta_j}$ increases and of those in $\mathrm{S}_{\Theta_j\Theta}$ decreases. Thus the slope of each linear piece of the sum functions in equation (4) is determined only by the rectangular strips in $\mathrm{S}_{\Theta\Theta_j}$ and $\mathrm{S}_{\Theta_j\Theta}$.

Notice also that at each breakpoint of $\bar{\mathsf{f}}_j$, only three strips in S change their type. The evaluation of the function $\bar{\mathsf{f}}_j$ thus starts with computing an initial configuration of strips, with strips divided in the above

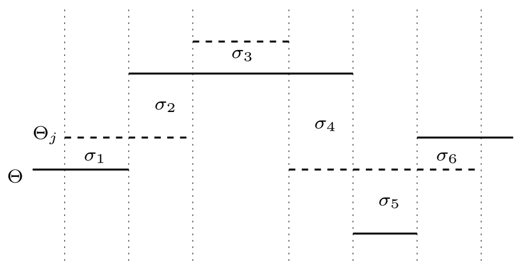


Figure 7: The four types of strips formed by the discontinuities of two turning functions Θ_j and Θ: $\sigma_1, \sigma_4 \in \mathrm{S}_{\Theta_j\Theta}$, $\sigma_2, \sigma_6 \in \mathrm{S}_{\Theta\Theta_j}$, $\sigma_3 \in \mathrm{S}_{\Theta_j\Theta_j}$, and $\sigma_5 \in \mathrm{S}_{\Theta\Theta}$.

mentioned four groups. An initial value $\bar{\mathsf{f}}_j(0)$ can be computed from this configuration in $O(m + n_j)$. We then go through the set X of critical events in order, and when necessary we update the configuration of strips (and thus the sets $\mathrm{S}_{\Theta\Theta_j}$ and $\mathrm{S}_{\Theta_j\Theta}$) and also compute the value of the function $\bar{\mathsf{f}}_j$, based on the slopes computed from $\mathrm{S}_{\Theta\Theta_j}$ and $\mathrm{S}_{\Theta_j\Theta}$. The updates of the configuration and the function values computation take constant time per critical point. Thus we can evaluate the function $\bar{\mathsf{f}}_j$ at all critical points in x in $O(N) = O(mn)$, and thus the step 2 of the algorithm takes $O(kmn)$ time.

The operations in the rest of the algorithm all take constant time each. Their total time is determined by the number of times line 10 is executed, which is at most N times for the loop in line 4, times N for the loop on line 6, times k for the loop on line 8, for a total of $O(kN^2) = O(km^2n^2)$ times.

Thus, line 4 to 12 dominate the running time of algorithm SimpleCompute, and this is $O(km^2n^2)$. The amount of storage used by the algorithm is dominated by the amount of space needed to store the matrix M, which is $O(kN^2) = O(km^2n^2)$. □

3.6 A fast algorithm

The above time bound on the computation of the similarity measure $\mathsf{d}(P_1, \ldots, P_k; P)$ can be improved to $O(kmn \log mn)$. The refinement of the dynamic programming algorithm is based on the following property of equation (8):

Lemma 5 *For any polyline P_j, $j \in \{1, \ldots, k\}$, and any critical point x_b, $b \in \{0, \ldots, N-1\}$, there is a critical point x_z, $0 \le z \le b$, such that:*

i) $D[j, a, b] = D[j, a, b-1]$, for all $a \in \{0, ..., z-1\}$, and

ii) $D[j, a, b] = D[j-1, a, b] + \bar{\mathsf{f}}_j(x_b)$, for all $a \in \{z, ..., b\}$.

Proof: For fixed values of j and b, let $(t_1^a, ..., t_j^a)$ denote the lexicographically smallest solution among the optimal solutions for the subproblem $D[j, a, b]$. For ease of notation, define $t_0^a = x_a$ and $t_{j+1}^a = x_b$. The proof of i) and ii) is based on the following observation: as a increases from 0 to b, the value t_i^a can only increase as well, or remain the same, for any $i \in \{1, \ldots, j\}$. We first prove this observation.

For the sake of contradiction, assume that there is an a ($0 < a \le b$) and an i ($1 \le i \le j$) such that $t_i^a < t_i^{a-1}$. For such an a, let i' be the smallest such i, and let i'' be the largest such i.

By our choice of i' and i'', we have $t_{i'-1}^a \le t_{i'}^a < t_{i'}^{a-1} \le t_{i''}^{a-1} \le t_{i''+1}^{a-1} \le t_{i''+1}^a$. It follows that $(t_1^a, ..., t_{i'-1}^a, t_{i'}^{a-1}, ..., t_{i''}^{a-1}, t_{i''+1}^a, ..., t_j^a)$ is a valid solution for j, a and b . From the fact that the lexicographically smallest optimal solution $(t_1^a, ..., t_j^a)$ is different, we must conclude that it is at least as good, i.e. $\sum_{i=i'}^{i''} \bar{\mathsf{f}}_i(t_i^a) \le \sum_{i=i'}^{i''} \bar{\mathsf{f}}_i(t_i^{a-1})$.

But we also have $t_{i'-1}^{a-1} \le t_{i'-1}^a \le t_{i'}^a \le t_{i''}^a < t_{i''}^{a-1} \le t_{i''+1}^{a-1}$, from which it follows that $(t_1^{a-1}, ..., t_{i'-1}^{a-1}, t_{i'}^a, ..., t_{i''}^a, t_{i''+1}^{a-1}, ..., t_{i''}^{a-1})$ is a valid solution for j, $a-1$ and b. From the fact that this solution is lexicographically smaller than the lexicographically smallest *optimal* solution $(t_1^{a-1}, ..., t_j^{a-1})$, we must conclude that it is suboptimal, so $\sum_{i=i'}^{i''} \bar{\mathsf{f}}_i(t_i^a) > \sum_{i=i'}^{i''} \bar{\mathsf{f}}_i(t_i^{a-1})$. This contradicts the conclusion of the previous paragraph. It follows that $t_i^a \geq t_i^{a-1}$, for all $0 < a \leq b$ and $1 \leq i \leq j$, and thus, by induction, that $t_i^{a'} \leq t_i^a$, for all $0 \leq a' \leq a$.

We can now prove the lemma item i) and ii). Let z be the smallest value in $\{0, ..., b\}$ for which $t_j^z = x_b$ (since t_j^b must be x_b, such a value always exists). Since, by our choice of z, we have $t_j^a \neq x_b$ for all $a < z$, we have $D[j, a, b] = D[j, a, b-1]$ for all $a \in \{0, ..., z-1\}$. Moreover, for $a \in \{z, ..., b\}$, we have $x_b = t_j^z \leq t_j^a \leq t_j^b = x_b$, so $t_j^a = x_b$, and thus, $D[j, a, b] = D[j-1, a, b] + \bar{\mathsf{f}}_j(x_b)$. □

For given j and b, we consider the function $\mathcal{D}[j, b] : \{0, N-1\} \to \mathbb{R}$, with $\mathcal{D}[j, b](a) = D[j, a, b]$. Lemma 5 expresses the fact that the values of function $\mathcal{D}[j, b]$ can be obtained from $\mathcal{D}[j, b-1]$ up to some value z, and from $\mathcal{D}[j-1, b]$ (while adding $\bar{\mathsf{f}}_j(x_b)$) from this value onwards. This property allows us to improve the time bound of the dynamic programming algorithm in the previous section. Instead of computing arrays of scalars $D[j, a, b]$, we will compute arrays of functions $\mathcal{D}[j, b]$. The key to success will be to represent these functions in such a way that they can be evaluated fast and $\mathcal{D}[j, b]$ can be constructed from $\mathcal{D}[j-1, b]$ and $\mathcal{D}[j, b-1]$ fast. Before describing the data structure used for this purpose, we present the summarized refined algorithm:

Algorithm FastCompute d$(P_1, \ldots, P_k; P)$

```
1. Compute the set of critical points $X = \{x_0, \ldots, x_{N-1}\}$, and sort them
2. For all $j \in \{1, ..., k\}$ and all $b \in \{0, ..., N-1\}$, evaluate $\bar{\mathsf{f}}_j(x_b)$
3. $ZERO \leftarrow$ a function that always evaluates to zero
4. $INFINITY \leftarrow$ a function that always evaluates to $\infty$
5. $MIN \leftarrow \infty$
6. $a \leftarrow 0$
7. for $j \leftarrow 1$ to $k$ do
8.     $\mathcal{D}[j, -1] \leftarrow INFINITY$
9. for $b \leftarrow 0$ to $N-1$ do
10.    $\mathcal{D}[0, b] \leftarrow ZERO$
11.    for $j \leftarrow 1$ to $k$ do
12.        Construct $\mathcal{D}[j, b]$ from $\mathcal{D}[j-1, b]$ and $\mathcal{D}[j, b-1]$
13.    while $x_a < x_b - l_0$ do
14.        $a \leftarrow a + 1$
15.    val $\leftarrow$ evaluation of $\mathcal{D}[k, b](a)$
16.    $MIN \leftarrow \min(\text{val}, MIN)$
17. return $\sqrt{MIN}$
```

The running time of this algorithm depends on how the functions $\mathcal{D}[j, b]$ are represented. In order to make especially steps 12 and 15 of the above algorithm efficient, we represent the functions $\mathcal{D}[j, b]$ by means of balanced binary trees. Asano et al. [4] used an idea similar in spirit.

3.6.1 An efficient representation for function $\mathcal{D}[j, b]$

We now describe the tree $\mathcal{T}_{j,b}$ used for storing the function $\mathcal{D}[j, b]$. Each node ν of $\mathcal{T}_{j,b}$ is associated with an interval $[a_\nu^-, a_\nu^+]$, with $0 \leq a_\nu^- \leq a_\nu^+ \leq N-1$. The root ρ is associated with the full domain, that is: $a_\rho^- = 0$ and $a_\rho^+ = N-1$. Each node ν with $a_\nu^- < a_\nu^+$ is an internal node that has a split value $a_\nu = \lfloor (a_\nu^- + a_\nu^+)/2 \rfloor$ associated with it. Its left and right children are associated with $[a_\nu^-, a_\nu]$ and $[a_\nu + 1, a_\nu^+]$, respectively. Each node ν with $a_\nu^- = a_\nu^+$ is a leaf of the tree, with $a_\nu = a_\nu^- = a_\nu^+$. For any index a of a critical point x_a, we will denote the leaf ν that has $a_\nu = a$ by λ_a. Note that so far, the tree looks exactly the same for each function $\mathcal{D}[j, b]$: they are balanced binary trees with N leaves, and $\log N$ height. Moreover, all trees have the same associated intervals, and split values in their corresponding nodes.

With each node ν we also store a weight w_ν, such that $\mathcal{T}_{j,b}$ has the following property: $\mathcal{D}[j, b](a)$ is the sum of the weights on the path from the tree root to the leaf λ_a. Such a representation of a function $\mathcal{D}[j, b]$ is not unique. A trivial example is to set w_ν equal to $\mathcal{D}[j, b](a_\nu)$ for all leaves ν, and to set it to zero for all internal nodes. It will become clear below why we also allow representations with non-zero weights on internal nodes.

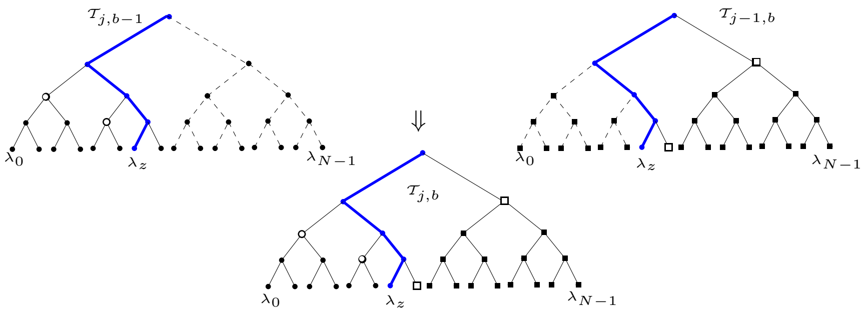


Figure 8: The tree $\mathcal{T}_{j,b}$ is contructed from $\mathcal{T}_{j,b-1}$ and $\mathcal{T}_{j-1,b}$ by creating new nodes along the path from the root to the leaf λ_z, and adopting the subtrees to the left of the path from $\mathcal{T}_{j,b-1}$, and the subtrees to the right of the path from $\mathcal{T}_{j-1,b}$.

Furthermore, we store with each node ν a value m_ν which is the sum of the weights on the path from the left child of ν to the leaf λ_{a_ν}, that is: the rightmost descendant of the left child of ν. This concludes the description of the data structure used for storing a function $\mathcal{D}[j,b]$.

Lemma 6 *The data structure $\mathcal{T}_{j,b}$ for the representation of function $\mathcal{D}[j,b]$ can be operated on such that:*

(i) The representation of a zero-function (i.e. a function that always evaluates to zero) can be constructed in $O(N)$ time. Also the representation of a function that always evaluates to ∞ can be constructed in $O(N)$ time.

(ii) Given $\mathcal{T}_{j,b}$ of $\mathcal{D}[j,b]$, evaluating function $\mathcal{D}[j,b](a)$ takes $O(\log N)$ time.

(iii) Given the representations $\mathcal{T}_{j-1,b}$ and $\mathcal{T}_{j,b-1}$ of the functions $\mathcal{D}[j-1,b]$ and $\mathcal{D}[j,b-1]$, respectively, a representation $\mathcal{T}_{j,b}$ of $\mathcal{D}[j,b]$ can be computed in $O(\log N)$ time.

Proof: (i) Constructing the representation of a zero-function involves nothing more than building a balanced binary tree on N leaves and setting all weights to zero. This can be done in $O(N)$ time. The representation of a function that always evaluates to ∞ takes also $O(N)$ time since it involves only the building a balanced binary tree on N leaves and setting the weights of all leaves to ∞, and of internal nodes to zero.

(ii) Once $\mathcal{T}_{j,b}$ representing the function $\mathcal{D}[j,b]$ has been constructed, the weights stored allow us to evaluate $\mathcal{D}[j,b](a)$ for any a in $O(\log N)$ time by summing up the weights of the nodes on the path from the root to the leaf λ_a that corresponds to a.

(iii) To construct $\mathcal{T}_{j,b}$ from $\mathcal{T}_{j,b-1}$ and $\mathcal{T}_{j-1,b}$ efficiently, we take the following approach. We find the sequences of left and right turns that lead from the root of the trees down to the leaf λ_z, where z is defined as in lemma 5. Note that the sequences of left and right turns are the same in the trees $\mathcal{T}_{j,b}$, $\mathcal{T}_{j,b-1}$, and $\mathcal{T}_{j-1,b}$, only the weights on the path differ. Though we do not compute z explicitly, we will show below that we are able to construct the path from the root of the tree to the leaf λ_z corresponding to z, by identifying, based on the stored weights, at each node node along this path whether the path continues into the right or left subtree of the current node.

Lemma 5 tells us that for each leaf left of λ_z, the total weight on the path to the root in $\mathcal{T}_{j,b}$ must be the same as the total weight on the corresponding path in $\mathcal{T}_{j,b-1}$. At λ_z itself and right of λ_z, the total weights to the root in $\mathcal{T}_{j,b}$ must equal those in $\mathcal{T}_{j-1,b}$, plus $\bar{\mathsf{f}}_j(x_z)$. We construct the tree $\mathcal{T}_{j,b}$ with these properties as follows.

We start building $\mathcal{T}_{j,b}$ by constructing a root ρ. If the path to λ_z goes into the right subtree, we adopt as left child of ρ the corresponding left child ν of the root from $\mathcal{T}_{j,b-1}$. There is no need to copy ν: we just add a pointer to it, thus at once making all descendants of ν in $\mathcal{T}_{j,b-1}$ into descendants of the corresponding node in the tree $\mathcal{T}_{j,b}$ under construction. Furthermore, we set the weight of ρ equal to the weight of the root of $\mathcal{T}_{j,b-1}$. If the path to λ_z goes into the left subtree, we adopt the right child from $\mathcal{T}_{j-1,b}$ and take the weight of ρ from there, now adding $\bar{\mathsf{f}}_j(x_z)$.

Then we make a new root for the other subtree of the root ρ, i.e. the one that contains λ_z, and continue the construction process in that subtree. Every time we go into the left branch, we adopt the right child from $\mathcal{T}_{j-1,b}$, and every time we go into the right branch, we adopt the left child from $\mathcal{T}_{j,b-1}$ (see figure 8). For every constructed node ν, we set its weight w_ν so that the total weight of ν and its ancestors equals the total weight of the corresponding nodes in the tree from which we adopt ν's child — if the subtree adopted comes from $\mathcal{T}_{j-1,b}$, we increase w_ν by $\bar{\mathsf{f}}_j(x_z)$.

By keeping track of the accumulated weights on the path down from the root in all the trees, we can set the weight of each newly constructed node ν correctly in constant time per node. The accumulated weights on the path down from the root, together with the stored weights for the paths down to left childrens' rightmost descendants, also allow us to decide in constant time which is better: $\mathcal{D}[j, b-1](a_\nu)$ or $\mathcal{D}[j-1, b](a_\nu) + \bar{\mathsf{f}}_j(x_z)$. This will tell us if λ_z is to be found in the left or in the right subtree of ν.

The complete construction process only takes $O(1)$ time for each node on the path from ρ to λ_z. Since the trees are perfectly balanced, this path has only $O(\log N)$ nodes, so that $\mathcal{T}_{j,b}$ is constructed in time $O(\log N)$. □

Theorem 1 *The similarity* $\mathsf{d}(P_1, \ldots, P_k; P)$ *between* k *polylines* $\{P_1, \ldots, P_j\}$ *with* n *vertices in total, and a polygon* P *with* m *vertices, can be computed in* $O(kmn \log(mn))$ *time using* $O(kmn \log(mn))$ *storage.*

Proof: We use algorithm Fastcompute $\mathsf{d}(P_1, \ldots, P_k; P)$ with the data structure described above. Step 1 and 2 of the algorithm are the same as in the algorithm from section 3.5 and can be executed in $O(kmn + mn \log n)$ time. From lemma 6, we have that the zero-function $ZERO$ can be constructed in $O(N)$ time (line 3). Similarly, the infinity-function $INFINITY$ can be constructed in $O(N)$ time (line 4). Lemma 6 also insures that constructing $\mathcal{D}[j, b]$ from $\mathcal{D}[j-1, b]$ and $\mathcal{D}[j, b-1]$ (line 12) takes $O(\log N)$ time, and that the evaluation of $\mathcal{D}[k, b](a)$ (line 15) takes $O(\log N)$ time.

Notice that no node is ever edited after it has been constructed: no tree is ever updated, and a new tree is always constructed by making $O(\log N)$ new nodes and for the rest by referring to nodes from existing trees as they are. Therefore, an assignment as in lines 8 or 10 of the algorithm can safely be done by just copying a reference in $O(1)$ time, without copying the structure of the complete tree. Thus, the total running time of the above algorithm will be dominated by $O(kN)$ executions of line 12, taking in total $O(kN \log N) = O(kmn \log(mn))$ time.

Apart from the function values of $\bar{\mathsf{f}}_j$ computed in step 2, we have to store the $ZERO$ and the $INFINITY$ function. All these require $O(kN) = O(kmn)$ storage. Notice that any of the functions constructed in step 12 requires only storing $O(\log(N)) = O(\log(mn))$ new nodes and pointers to nodes in previously computed trees, and thus we need $O(kmn \log(mn))$ for all the trees computed in step 12. So the total storage required by the algorithm is $O(kmn \log(mn))$. □

We note that the problem resembles a general edit distance type approximate string matching problem [19]. Global string matching under a general edit distance error model can be done by dynamic programming in $O(kN)$ time, where k and N represent the lengths of the two strings. The same time complexity can be achieved for partial string matching through a standard "assign first line to zero" trick [19]. This trick however does not apply here due to the condition $x_b - x_a \leq l_0$. Indeed, by the above trick to convert global matching into partial matching we would have to fill in a table D of $O(kN)$ elements, in which each entry $D[j, b]$ gives the cost of the best alignment of the first j polylines ending at critical point x_b (over all possible starting points x_a satisfying the condition $x_b - x_a \leq l_0$). This table however cannot be filled efficiently in a column-wise manner, because an optimal $D[j, b]$ might be given by an alignment of the first j polylines, such that the alignment of the first $j-1$ polylines is sub-optimal for $D[j-1, c]$, where x_c is the placement of the polyline $j-1$. This is also the case for a different formulation of the dynamic programming problem, where $D[j, b]$ gives the best alignment of the first j polylines ending at a critical point within the interval $[x_0, x_b]$.

4 A Part-based Retrieval Application

In this section we describe a part-based shape retrieval application. The retrieval problem we are considering is the following: given a large collection of polygonal shapes, and a query consisting of a set of disjoint

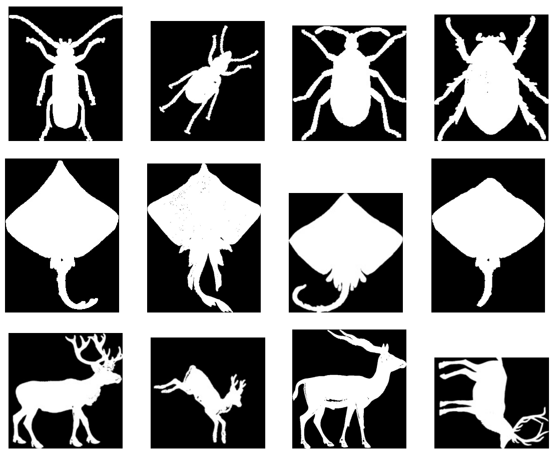


Figure 9: Examples of images from Core Experiment "CE-Shape1", part B. Images in the same row belong to the same class.

polylines, we want to retrieve those shapes in the collection that best match the query. The query represents a set of disjoint boundary parts of a single shape, and the matching process evaluates how closely these parts resemble disjoint pieces of a shape in the collection. Thus, instead of querying with complete shapes, we make the query process more flexible by allowing the user to search only for certain parts of a given shape. The parts in the query are selected by the user from an automatically computed decomposition of a given shape. More specifically, in the application we discuss below, these parts are the result of applying the boundary decomposition framework introduced in [29]. For matching the query against a database shape we use the similarity measure described in section 3.

The shape collection used by our retrieval application comes from the MPEG-7 shape silhouette database. Specifically, we used the Core Experiment "CE-Shape-1" part B [13], a test set devised by the MPEG-7 group to measure the performance of similarity-based retrieval for shape descriptors. This test set consists of 1400 images: 70 shape classes, with 20 images per class. Some examples of images in this collection are given in figure 9, where shapes in each row belong to the same class. The outer closed contour of the object in each image was extracted. In this contour, each pixel corresponds to a vertex. In order to decrease the number of vertices, we then used the Douglas-Peuker [9] polygon approximation algorithm. Each resulting simplified contour was then decomposed, as described in [29]. The instantiation of the boundary decomposition framework described there uses the medial axis. For the medial axis computation we used the Abstract Voronoi Diagram LEDA package [1]. The running time for a single query on the MPEG-7 test set of 1400 images is typically about one second on a 2 GHz PC.

The matching of a query consisting of k polylines to an arbitrary database contour is based on the similarity measure described in section 3. This similarity measure is not scale invariant, as we noticed in section 3.1. Our shape collection, however, contains shapes at different scales. In order to achieve some robustness to scaling, we scaled all shapes in the collection to the same diameter of the circumscribed disk. Given the nature of our collection (each image in "CE-Shape-1" is one view of a complete object), this is a reasonable approach. The reason we opted for a normalization based on the circumscribed disk, instead of the bounding box, for example, is related to the fact that a class in the collection may contain images of an object at different rotational angles.

4.1 Experimental Results

The selection of parts comprising the query has a big influence on the results of a part-based retrieval process. Our collection for example includes classes, such as those labelled "horse", "dog", "deer", "cattle", whose shapes have similar parts, such as limbs. Querying with such parts have shown to yield shapes from all these classes. Figure 10 depicts two part-based queries on the same shape (belonging to the "beetle" class) with very different results. In the first example, the query consists of five chains, each representing a

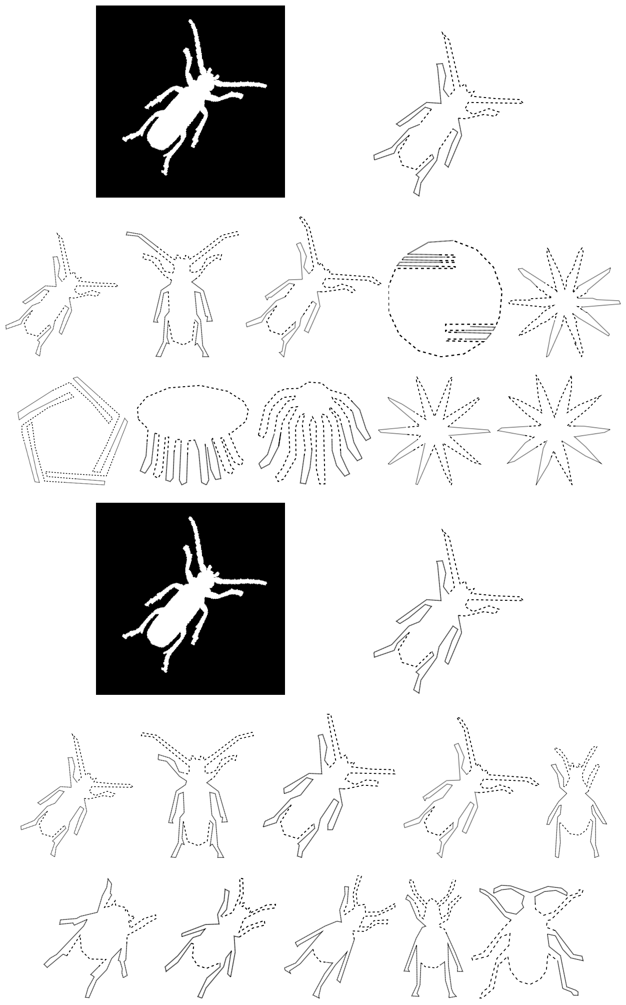


Figure 10: (Upper half) Original image, decomposed countour with five query parts in solid line style, each representing a leg of a beetle contour, and the top ten retrieved shapes when quering with these five polylines. (Lower half) The same original image and decomposed contour, and the top ten retrieved shapes when quering with two polylines, one containing three legs and the other two legs.

leg of the beetle. Among the first ten retrieved results, only three come from the same class. In the second example, the same parts are contained in the query, but they were concatenated, so that they form two chains (three legs on one side, two of the other). All first ten retrieved results are this time from the same class. By concatenating the leg parts in the second query, a spatial arrangement of these parts is searched in the database, and this gives the query more power to discriminate between beetle shapes and other shapes containing leg-like parts.

The MPEG group initiated the MPEG-7 Visual Standard in order to specify standard content-based descriptors that allow to measure similarity in images or video based on visual criteria. Each visual descriptor incorporated in the MPEG-7 Standard was selected from a few competing proposals, based on an evaluation of their performance in a series of tests called Core Experiments. The Core Experiment "CE-Shape-1" was devised to measure the performance of 2D shape descriptors. The performance of several shape descriptors, proposed for standardization within MPEG-7, is reported in [21]. The performance of each shape descriptor was measured using the so-called "bulls-eye" test: each image is used as a query, and the number of retrieved images belonging to the same class was counted in the top 40 (twice the class size) matches.

The shape descriptor selected by MPEG-7 to represent a closed contour of a 2D object or region in an image is based on the Curvature Scale Space (CSS) representation [18]. The CSS of a closed planar curve is computed by repeatedly convolving the contour with a Gaussian function, and representing the curvature zero-crossing points of the resulting curve in the (s, σ) plane, where s is the arclength. The matching process has two steps. Firstly, the eccentricity and circularity of the contours are used as a filter, to reduce the amount of computation. The shape similarity measure between two shapes is computed, in the second step, by relating the positions of the maxima of the corresponding CSSs. The reported similarity-based retrieval performance of this method is 75.44% [21]. In the years following the selection of this descriptor for the MPEG-7 Visual Standard, people have reported better performances on the same set of data for other shape descriptors. Belongie et al. [6] reported a retrieval performance of 76.51% for their descriptor based on shape contexts. Even better performances, of 78.18% and 78.38%, were reported by Sebastian et al. [27] and Grigorescu and Petkov [10], respectively. The matching in [27] is based on aligning the two shapes in a deformation-based approach, while in [10] a local descriptor of an image point is proposed that is determined by the spatial arrangement of other image points around that point. Latecki at al. mention in [16] a retrieval performance of 83.19% for a shape descriptor introduced in [5].

A global turning function-based shape descriptor [12], proposed for MPEG-7, is reported to have a similarity-based retrieval performance of 54.14%. A later variation [33] increases the performance rate up to 65.67%.

For classes in “CE-Shape-1” with a low variance among their shapes, the CSS matching [18] gives good results, with a retrieval rate over 90%, as measured by the “bulls-eye” test. For difficult classes however, for example the class “beetle”, the different relative lengths of the antennas/legs, and the different shapes of the body pose problems for global retrieval. The average performance rate of CSS matching for this class is only 36%. For the “ray” and “deer” classes, the bad results (an average performance rate of the CSS matching of 26% and 33%, respectively) are caused by the different shape and size of the tails and antlers, respectively, of the contours in these classes. A part-based matching, with a proper selection of parts, is significantly more effective in such cases.

We tested the performance of our part-based shape matching. A prerequisite of such a performance of the part-based matching is the selection of query parts that capture relevant and specific characteristics of the shape. This has to be done interactively by the user. An overall performance score for the matching process, like in [21], is therefore untractable, since that would re quire 1400 interactive queries. We therefore compared, on a number of individual queries, our part-based approach with the global CSS matching and with a global matching based on the turning function. These experimental results indicate that for those classes with a low average performance of the CSS matching, our approach consistently performs better. For example, for the shape called “carriage-18”, the CSS method has a bull’s eye score of 70%, the global turning function method a score of 80%, and our multiple polyline to polygon matching method a score of 95%. For the shape called “horse-17”, both the CSS method and the global turning function method have a bull’s eye score of 10%, and our method a score of 70%. (See many examples in the appendix.)

5 Concluding Remarks

In this paper we addressed the partial shape matching problem. We introduced a measure for computing the similarity between a set of pieces of one shape and another shape. Such a measure is usefull in overcoming problems due to occlusion or unreliable object segmentation from images. The similarity measure was tested in a shape-based image retrieval application. We compared our part-based approach to shape matching with two global shape matching technique (the CSS matching, and a turning function-based matching). Experimental results indicate that for those classes with a low average performance of the CSS matching, our approach consistently performs better. Concludingly, our dissimilarity measure provides a powerful complementary shape matching method.

A prerequisite of such a performance of the part-based matching is the selection of query parts that capture relevant and specific characteristics of the shape. This has to be done interactively by the user. Since the user does not usually have any knowledge of the databased content before the retrieval process, we have developed an alternative approach to part-based shape retrieval, in which the user is relieved of the responsibility of specifying shape parts to be searched for in the database, see [30]. The query, in this case, is a polygon, and in order to select among the large number of possible searches in the database with parts of the query, the user interacts with the system in the retrieval process. His interaction, in the form of marking relevant results from those retrieved by the system, has the purpose of allowing the system to guess what the user is looking for in the database.

Acknowledgements

This research was partially supported by the Dutch Science Foundation (NWO) under grant 612.061.006, and by the FP6 IST Network of Excellence 506766 Aim@Shape. Thanks to Veli Mäkinen for the partial string matching reference.

References

[1] AVD LEP, the Abstract Voronoi Diagram LEDA Extension Package. `http://www.mpi-sb.mpg.de/LEDA/friends/avd.html`.

[2] N. Ansari and E. J. Delp. Partial shape recognition: A landmark-based approach. *PAMI*, 12:470–483, 1990.

[3] E. Arkin, L. Chew, D. Huttenlocher, K. Kedem, and J. Mitchell. An efficiently computable metric for comparing polygonal shapes. *PAMI*, 13:209–215, 1991.

[4] T. Asano, M. de Berg, O. Cheong, H. Everett, H.J. Haverkort, N. Katoh, and A. Wolff. Optimal spanners for axis-aligned rectangles. *Computational Geometry, Theory and Applications*, 30(1):59–77, 2005.

[5] E. Attalla. *Shape Based Digital Image Similarity Retrieval*. PhD thesis, Wayne State University, 2004.

[6] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *PAMI*, 24:509–522, 2002.

[7] B. Bhanu and J. C. Ming. Recognition of occluded objects: A cluster-structure algorithm. *Pattern Recognition*, 20(2):199–211, 1987.

[8] Scott D. Cohen and Leonidas J. Guibas. Partial matching of planar polylines under similarity transformations. In *Proc. SODA*, pages 777–786, 1997.

[9] D.H. Douglas and T.K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.

[10] C. Grigorescu and N. Petkov. Distance sets for shape filters and shape recognition. *IEEE Transactions on Image Processing*, 12(10):1274–1286, 2003.

[11] D. Huttenlocher, G. Klanderman, and W. Rucklidge. Comparing images using the hausdorff distance. *PAMI*, 15:850–863, 1993.

[12] IBM. Technical summary of turning angle shape descriptors proposed by IBM. Technical report, IBM, 1999. TR ISO/IEC JTC 1/SC 29/WG 11/ P162.

[13] S. Jeannin and M. Bober. Description of core experiments for MPEG-7 motion/shape. Technical report, MPEG-7, March 1999. Technical Report ISO/IEC JTC 1/SC 29/WG 11 MPEG99/N2690, Seoul.

[14] L. J. Latecki and R. Lakämper. Shape similarity measure based on correspondence of visual parts. *PAMI*, 22:1185–1190, 2000.

[15] L. J. Latecki, R. Lakämper, and D. Wolter. Shape similarity and visual parts. In *Proc. Int. Conf. Discrete Geometry for Computer Imagery (DGCI)*, pages 34–51, 2003.

[16] L. J. Latecki, R. Lakämper, and D. Wolter. Optimal partial shape similarity. *Image and Vision Computing*, 23:227–236, 2005.

[17] H.-C. Liu and M. D. Srinath. Partial shape classification using contour matching in distance transformation. *PAMI*, 12(11):1072–1079, 1990.

[18] F. Mokhtarian, S. Abbasi, and J. Kittler. Efficient and robust retrieval by shape content through curvature scale space. In *Workshop on Image DataBases and MultiMedia Search*, pages 35–42, 1996.

[19] Gonzala Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[20] H. Ogawa. A fuzzy relaxation technique for partial shape matching. *Pattern Recognition Letters*, 15(4):349–355, 1994.

[21] J.-R. Ohm and K. Müller. Results of MPEG-7 Core Experiment Shape-1. Technical report, MPEG-7, July 1999. Technical Report ISO/IEC JTC1/SC29/WG11 MPEG98/M4740.

[22] E. Ozcan and C. K. Mohan. Partial shape matching using genetic algorithms. *Pattern Recognition Letters*, 18(10):987–992, 1997.

[23] A. Pentland, R. W. Picard, and S. Sclaroff. Photobook: Content-based manipulation of image databases. *International Journal of Computer Vision*, 18(3):233–254, June 1996.

[24] E. Persoon and K. S. Fu. Shape discrimination using Fourier descriptors. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(3):170–179, 1977.

[25] R. J. Prokop and A. P. Reeves. A survey of moment-based techniques for unoccluded object representation and recognition. *Computer Vision, Graphics, and Image Processing*, 54(5):438–460, Setember 1992.

[26] P. Rosin. Multiscale representation and matching of curves using codons. *Graphical Models and Image Processing*, 55(4):286–310, July 1993.

[27] T. Sebastian, P. Klein, and B. Kimia. On aligning curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(1):116–125, 2003.

[28] K. Siddiqi, A. Shokoufandeh, S.J. Dickinson, and S.W. Zucker. Shock graphs and shape matching. *IJCV*, 55(1):13–32, 1999.

[29] Mirela Tanase. *Shape Deomposition and Retrieval*. PhD thesis, Utrecht University, Department of Computer Science, February 2005.

[30] Mirela Tanase and Remco C. Veltkamp. Part-based shape retrieval with relevance feedack. In *Proceedings International Conference on Multimedia and Expo (ICME05)*, 2005.

[31] R. Veltkamp and M. Hagedoorn. State-of-the-art in shape matching. In M. Lew, editor, *Principles of Visual Information Retrieval*, pages 87–119. Springer, 2001.

[32] Haim Wolfson and Isidore Rigoutsos. Geometric hashing: an overview. *IEEE Computational Science & Engineering*, pages 10–21, October-December 1997.

[33] C. Zibreira and F. Pereira. A study of similarity measures for a turning angles-based shape descriptor. In *Proc. Conf. Telecommunications, Portugal*, 2001.

Appendix A

We compared our multiple polyline to polygon matching (MPP) approach with the global CSS matching (CSS) and with a global matching based on the turning function (GTA). Table 1 presents the query image of the CSS matching and the query parts of our part-based matching in the first and second column of the table. The retrieval rate measured by the "bulls-eye" test, the percentage of true positives returned in the first 40 (twice the class size) matches, for the three matching techniques are indicated in the following three columns, while the remaining columns give the percentage of true positives returned in the first 20 (class size) matches.

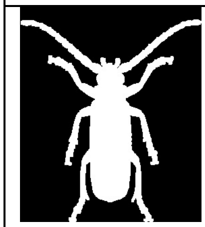
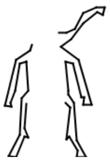
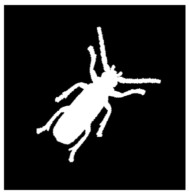
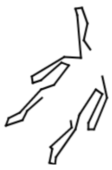
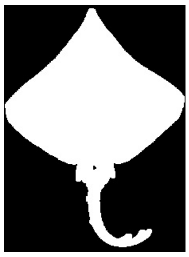
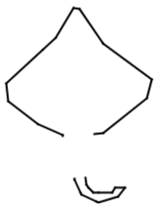
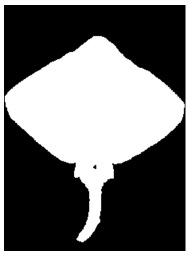
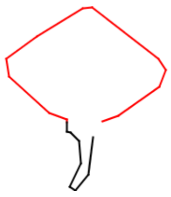
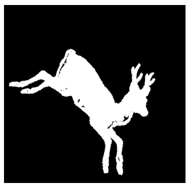
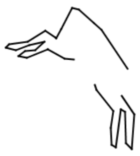
CSS Query Image	Part-base Query Parts	Bull's Eye Performance			True Positives in class size		
		Matching Process			Matching Process		
		CSS	GTA	MPP	CSS	GTA	MPP
 beetle-20		10%	30%	65%	10%	10%	55%
 beetle-10		10%	35%	60%	5%	20%	55%
 ray-3		15%	15%	70%	15%	5%	60%
 ray-17		15%	25%	50%	15%	20%	40%
		15%	40%	50%	5%	30%	35%

Table 1: A comparison of the Curvature Scale Space (CSS), the Global Turning Angle function (GTA), and our Multiple Polyline to Polygon (MPP) matching.

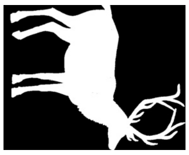
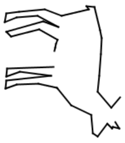
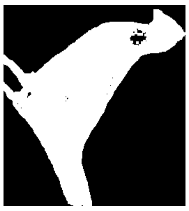
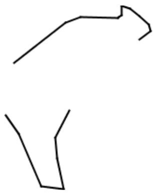
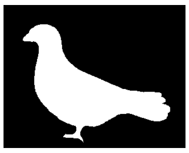
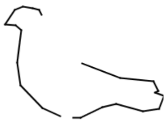
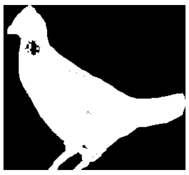
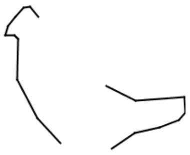
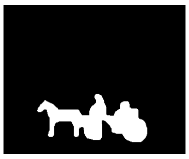
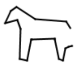
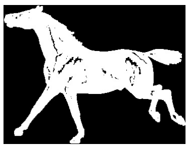
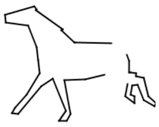
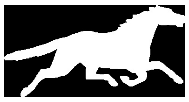
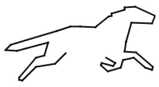
CSS Query Image	Part-base Query Parts	Bull's Eye Performance Matching Process			True Positives in class size Matching Process		
		CSS	GTA	MPP	CSS	GTA	MPP
deer-15  deer-13		20%	40%	45%	15%	40%	40%
 bird-11		5%	20%	45%	5%	15%	35%
 bird-17		20%	15%	60%	20%	15%	40%
 bird-9		20%	25%	50%	15%	25%	40%
 carriage-18		70%	80%	95%	45%	80%	90%
 horse-17		10%	10%	70%	10%	5%	60%
 horse-3		15%	25%	65%	10%	25%	40%

Table 1: A comparison of the Curvature Scale Space (CSS), the Global Turning Angle function (GTA), and our Multiple Polyline to Polygon (MPP) matching.

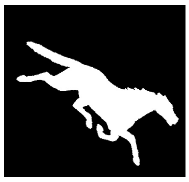
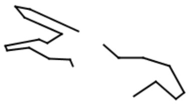
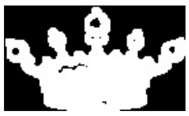
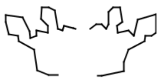
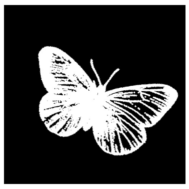
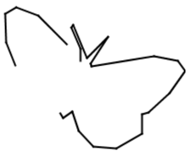
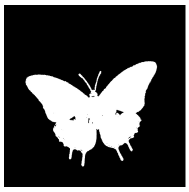
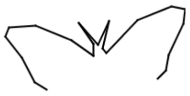
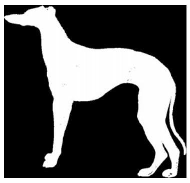
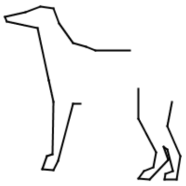
CSS Query Image	Part-base Query Parts	Bull's Eye Performance			True Positives in class size		
		Matching Process			Matching Process		
		CSS	GTA	MPP	CSS	GTA	MPP
 horse-4		25%	30%	50%	20%	20%	45%
 crown-13		30%	30%	50%	25%	25%	40%
 butterfly-11		20%	45%	65%	20%	35%	50%
 butterfly-4		15%	25%	55%	10%	20%	55%
 dog-11		10%	15%	50%	10%	15%	45%

Table 1: A comparison of the Curvature Scale Space (CSS), the Global Turning Angle function (GTA), and our Multiple Polyline to Polygon (MPP) matching.