Geometric Pattern Matching under Euclidean Motion

L. Paul Chew[1], Michael T. Goodrich[2], Daniel P. Huttenlocher[3], Klara Kedem[3,4], Jon M. Kleinberg[3], and Dina Kravets[5]

Abstract

Given two planar sets A and B, we examine the problem of determining the smallest ε such that there is a Euclidean motion (rotation and translation) of A that brings each member of A within distance ε of some member of B. We establish upper bounds on the combinatorial complexity of this subproblem in model-based computer vision, when the sets A and B contain points, line segments, or (filled-in) polygons. We also show how to use our methods to substantially improve on existing algorithms for finding the *minimum Hausdorff distance* under Euclidean motion.

[1]Author's address: Department of Computer Science, Cornell University, Ithaca, NY 14853. This work was supported by the Advanced Research Projects Agency of the Department of Defense under ONR Contract N00014-92-J-1989, and by ONR Contract N00014-92-J-1839, NSF Contract IRI-9006137, and AFOSR Contract AFOSR-91-0328.

[2]Author's address: Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218. This work was supported by the NSF and DARPA under Grant CCR-8908092, and by the NSF under Grants CCR-9003299 and IRI-9116843.

[3]Authors' address: Department of Computer Science, Cornell University, Ithaca, NY 14853. This work was supported in part by NSF grant IRI-9057928 and matching funds from Kodak, General Electric, and Xerox, and in part by US Air Force contract AFOSR-91-0328.

[4]Author's address: Department of Mathematics and Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel. This work was also supported by the Eshkol grant 04601-90 from The Israeli Ministry of Science and Technology.

[5]Author's address: Computer Science Dept., New Jersey Institute of Technology, University Heights, Newark, NJ 07102. This work was supported in part by the Air Force under Contract AFOSR-89-0271 and by the Defense Advanced Research Projects Agency under Contracts N00014-87-K-825 and N00014-89-J-1988.

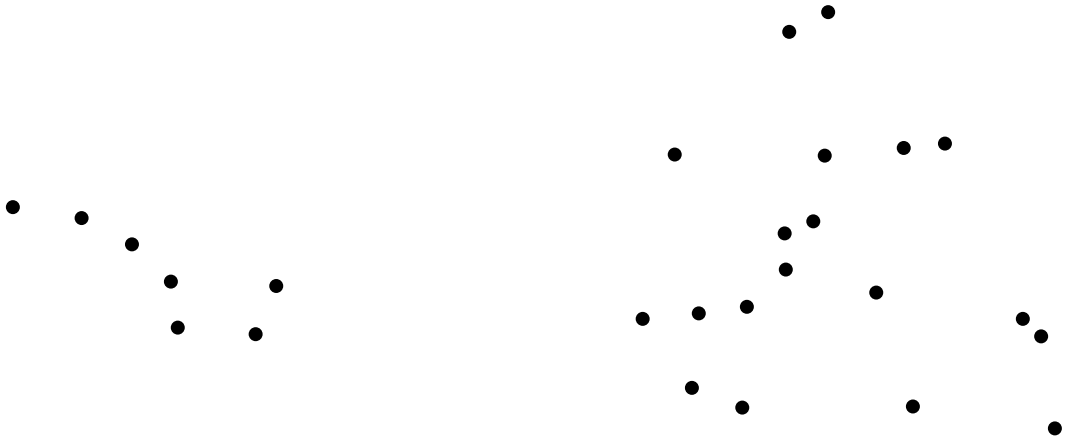

Figure 1: An example pattern and background. In this case there is match of the pattern to the background under Euclidean motion for $\varepsilon = 0.1$ in.

1. Introduction

The problem of determining whether a given planar geometric set can be found in another planar geometric set [2, 3, 4, 5, 6, 7, 8, 11, 19, 20, 21, 22] is central to computer vision and also appears in such diverse fields as biology, astronomy, and robotics. For example, a typical computational problem in astronomy is to locate a certain configuration of stars in the night sky (see Figure 1).

This problem can be modeled as a pattern matching problem for planar sets A and B. We consider B to be fixed and ask if there is a Euclidean motion (translation and rotation) that moves pattern A in such a way that it matches a subset of B. In practice, there are usually small errors in the data that make an exact match unlikely. So our approach is to find the smallest value $\varepsilon > 0$ such that there is a Euclidean motion of A bringing each point of A to within ε of some object in B. We refer to this as the *one-way minimum Hausdorff distance* problem, because it is closely related to the problem of finding the minimum Hausdorff distance: given planar sets A and B find the minimum error tolerance ε so that there is a Euclidean motion of A such that the Hausdorff distance (defined below) between A and B, $H(A, B)$, is less than or equal to ε. By definition, $H(A, B) \leq \varepsilon$ if and only if each point of A is within ε of some point in B and each point of B is within ε of some point in A.

In this paper we present improved algorithms for the one-way and bidirectional minimum Hausdorff distance problems. In particular, if A is a set of m points in the plane and if B is a set of n points in the plane, then we show how to find the one-way minimum Hausdorff distance under Euclidean motion in $O(m^3n^2 \log^2 mn)$ time. More generally, if A (resp., B) is a set of polygons defined by m (resp., n) segments, then we show how to solve one-way minimum Hausdorff distance under Euclidean motion in $O(m^3n^3 \log^2 mn)$ time. These algorithms can be used to solve the (bidirectional) minimum Hausdorff distance problem under Euclidean motion for sets A and B in the plane in time $O((m+n)^5 \log^2 mn)$ when A and B consist only of points, and in time $O((m+n)^6 \log^2 mn)$ when A and B consist of points, segments, or polygons. This improves the previous algorithm of Huttenlocher, Kedem, and Kleinberg [20] for point sets by nearly a linear factor (as their method runs in $O((m +$

$n)^6 \log mn)$ time), and it improves the previous algorithm of Alt, Behrends, and Blömer [2] for line segments by nearly a cubic factor (as their method runs in $O(m^4n^4(m+n)\log mn)$ time). Recent work by Rucklidge [25] shows that our bounds for the one-way minimum Hausdorff distance are nearly tight (i.e., within log factors) among those algorithms that find all motions bringing A to within ε of B. Note though that it may be possible to solve a minimum Hausdorff distance problem without finding all such motions (see, for instance, [11]).

All our methods share the same general approach. We first design a decision procedure that determines if there exists a good Euclidean transformation from A to B for a given value of $\varepsilon > 0$ (and we output such a transformation if one exists). Our methods for solving this decision procedure differ depending upon the geometric properties of A and B, but they are all based on a common idea of examining the structure of intersections among a set of potentially valid Euclidean motions. In each case, our decision procedures have time bounds that are a $\log mn$ factor faster than our bounds for the minimization problems. This is no accident, as our methods for solving a minimization problem are based upon using the corresponding decision method as a black box in performing a "binary search" for the best value of ε allowing a valid Euclidean motion. Of course, we cannot use a standard binary search to find this best ε value, since the range of values is continuous. So, instead, we design a special parallel algorithm to determine a good set of candidate values of ε at which to probe. This technique is known as *parametric searching* [23, 12, 13, 14, 1, 10] and, using a pipelining technique of Cole [12, 13], our application of it to the one-way minimum Hausdorff distance requires only $O(\log mn)$ probes, giving the claimed bounds.

In the next section we give some notation and background. In Section 3 we present our decision procedure for the case when A and B are finite point sets, and in Section 4 extend this to line segments and polygons. We show how to apply parametric searching to these decision procedures to solve the minimization problems in Section 5, and we conclude in Section 6.

2. Notation and Background

Consider the sets $\{a_1, \ldots, a_m\}$ and $\{b_1, \ldots, b_n\}$, where each a_i and b_i is either a point, a segment, or a (filled-in) triangle. Let A be $\cup_{i=1}^m a_i$ and B be $\cup_{i=1}^n b_i$. The Hausdorff distance between two sets A and B is defined as

$$H(A,B) = \max(h(A,B), h(B,A)),$$

where

$$h(A,B) = \sup_{\alpha \in A} \inf_{\beta \in B} d(\alpha, \beta),$$

and $d(\alpha, \beta)$ is the distance between two points α and β. For us, $d(\cdot,\cdot)$ is the standard L_2 distance in the plane. It is well known that $H(A,B)$ is a metric.

The minimum Hausdorff distance under Euclidean motion is then defined as

$$D(A,B) = \min_{g \in E_2} H(g(A), B),$$

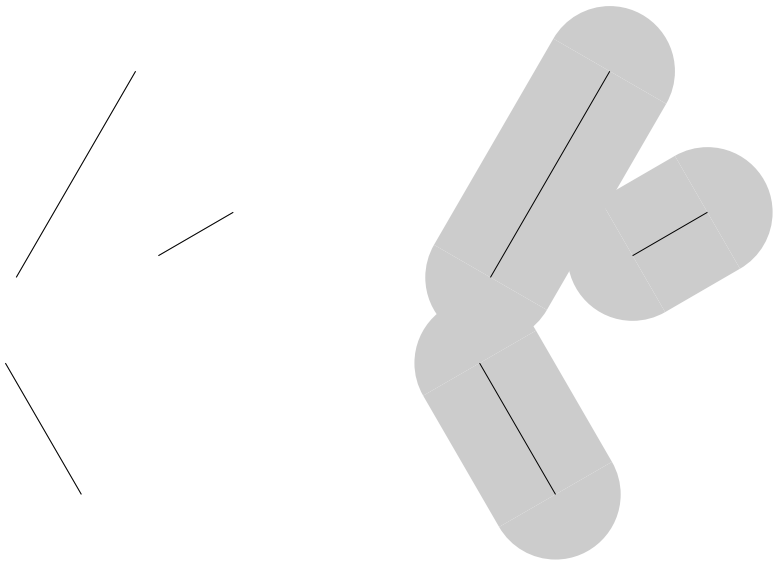

Figure 2: B and B^ε for the line segment case.

where E_2 is the group of planar Euclidean motions and $g(A)$ is the motion applied to A. By simple arguments similar to those in [21], one can show that $D(A, B)$ is a metric.

Using this notation, the *decision problem* for the one-way Hausdorff distance can be restated as, "Given A and B in the plane and an error bound $\varepsilon > 0$, is there a transformation $g \in E_2$ such that $h(g(A), B) \leq \varepsilon$?" In the remainder of this section, we develop the tools necessary to treat this problem as a geometric intersection problem.

Define

$$B^\varepsilon = \bigcup_{i=1}^{n} (b_i \oplus C_\varepsilon),$$

where C_ε is a circle of radius ε, centered at the origin, and $\oplus$ represents the Minkowski sum (i.e., $P \oplus Q = \{p + q | p \in P, q \in Q\}$); in other words B^ε is the set B expanded by ε (see Figure 2).

Note that the directed Hausdorff distance under Euclidean motion, $h(g(A), B)$, is at most ε if and only if there exists a map $g \in E_2$ such that $g(A)$ falls within B^ε. Each map $g \in E_2$ can be considered as a translation (a 2D vector) followed by a rotation about the origin. Thus, each map corresponds to a point in *transformation space* (x, y, θ). Ignoring θ for the moment (assume $\theta = 0$), the set of all maps that cause a point $\alpha \in A$ to fall within B^ε can be represented as $B^\varepsilon \oplus (-\alpha)$, a simple translate of B^ε in transformation space.

For a more complicated object, say a segment or triangle, s, each point of s must fall within B^ε. Thus the set of maps that take s within B^ε is an intersection of infinitely many translates of B^ε. We can represent this succinctly by considering *forbidden maps*, maps that fail to take s within B^ε. The forbidden maps for s correspond to the union of infinitely many translates of $\overline{B^\varepsilon}$, where $\overline{B^\varepsilon}$ is the complement of B^ε. This union can be written as a Minkowski sum: $\overline{B^\varepsilon} \oplus (-s)$ corresponding to the forbidden maps for s. Thus for a given object a_i (i.e., a point, segment, or triangle), the set of maps that take a_i within B^ε is given

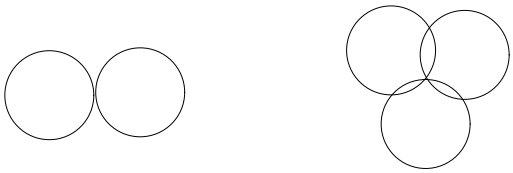

Figure 3: Double and triple events.

by

$$B_i^\varepsilon = \overline{\overline{B^\varepsilon} \oplus (-a_i)}.$$

The sets B_i^ε can be parameterized by θ. If we denote by $r_\theta(p)$ the rotation of p by θ, then

$$B_i^\varepsilon(\theta) = \overline{\overline{B^\varepsilon} \oplus (-r_\theta(a_i))},$$

i.e., $B_i^\varepsilon(\theta)$ is a two-dimensional "slice" of B_i^ε. The one-way minimum Hausdorff distance under Euclidean motion for A and B is at most ε if and only if there is a value of θ such that

$$S(\theta) = \bigcap_{i=1}^{m} B_i^\varepsilon(\theta)$$

is non-empty. In other words, we must search for a value of θ at which all the $B_i^\varepsilon(\theta)$'s have a non-empty intersection. This corresponds to a Euclidean map that brings each component of A within B^ε.

3. Finite Point Sets

When A and B are finite point sets, B^ε is a union of discs. For a fixed θ, $S(\theta)$ is therefore simply an intersection of a finite number of unions of discs of radius ε. Thus, $S(\theta)$ is either $\emptyset$ or the boundary of $S(\theta)$ consists of arcs of circles of radius ε.

Define $\mathcal{A}(\theta)$ to be the arrangement [16] formed by overlaying all the boundaries of the sets $B_i^\varepsilon(\theta)$, $1 \leq i \leq m$ on each other. In addition, define the *depth* of a point p to be the number of $B_i^\varepsilon(\theta)$ sets that contain p (where containment includes points on the boundary). We can determine whether $S(\theta)$ is non-empty for some θ by forming $\mathcal{A}(\theta)$ and determining the *depth* of each of its vertices. Clearly, $S(\theta)$ is non-empty if and only if there is a vertex in $\mathcal{A}(\theta)$ whose depth is m. Indeed, if $S(\theta)$ is non-empty, then it consists of faces of $\mathcal{A}(\theta)$. To determine if there is a good Euclidean transformation, we "sweep" the transformation space from $\theta = 0$ to 2π while maintaining the depth of the vertices in $\mathcal{A}(\theta)$.

Consider, then, how $\mathcal{A}(\theta)$ changes as θ grows. Note that, as far as maintaining depth is concerned, only the boundaries of each $B_i^\varepsilon(\theta)$ are important. This is due to the fact that an inner boundary (a portion of a circle in the interior of the union of discs that makes up B^ε) does not affect depth. We define a *vertex* of $B_i^\varepsilon(\theta)$ to be the intersection point of two circular arcs on its boundary. A vertex in $\mathcal{A}(\theta)$ is either a vertex of some $B_i^\varepsilon(\theta)$ or the intersection of two outer-boundary arcs, one from $B_k^\varepsilon(\theta)$ and one from $B_l^\varepsilon(\theta)$, for some k

and l. A *combinatorial change* to $\mathcal{A}(\theta)$ occurs in one of two cases (see Figure 3): (i) when two arcs in $\mathcal{A}(\theta)$ become tangent at some θ, or (ii) when three arcs in $\mathcal{A}(\theta)$ intersect at a point. We refer to the first kind of change as a *double event*, and the second kind as a *triple event*. The combinatorial complexity of the dynamically changing arrangement $\mathcal{A}(\theta)$ is thus defined to be the total number of combinatorial changes — double and triple events — for θ changing from 0 to 2π.

Lemma 1 *The number of double events is $O(m^2n^2)$.*

Proof. A double event is a tangency of two discs. Since each $B_i^\varepsilon(\theta)$ translates along a circular path as θ changes, any two discs from, say, $B_k^\varepsilon(\theta)$ and $B_l^\varepsilon(\theta)$, have at most two tangencies. Thus, the total number of double events is bounded by a constant multiple of the total number of pairs of the mn discs, which is $O(m^2n^2)$. ∎

Lemma 2 *The number of triple events is $O(m^3n^2)$.*

Proof. A triple event is an intersection point of three arcs centered at, say, p, q, and r, where each arc is on the outer boundary of its $B_i^\varepsilon(\theta)$. Each of p, q, and r belongs to some set $\mathcal{S}_l(\theta) = B \oplus (-r_\theta(a_{k_l}))$, for $l \in \{1,2,3\}$ respectively, the most prolific case being when each of the k_l's are distinct. The intersection point of the three arcs is by definition equidistant from p, q, and r, and on the outer boundary of $B_{k_1}^\varepsilon(\theta)$, $B_{k_2}^\varepsilon(\theta)$, and $B_{k_3}^\varepsilon(\theta)$, respectively. Thus the intersection corresponds to a vertex in the Voronoi diagram [16, 24] of the union of the three sets $\mathcal{S}_l(\theta)$ at this θ. Huttenlocher *et al.* [20] show that if one has k sets of n points moving rigidly in the plane, then there are at most $O(n^2k^2\log^* k)$ changes in the Voronoi diagram of all the points. In our case, $k = 3$; hence, we get that the total number of triple events for the three sets is $O(n^2)$. There are $O(m^3)$ different ways to choose three such sets, so the total number of triple events is $O(m^3n^2)$. Since the results of Huttenlocher *et al.* assume that the points are in general position, we need to make the same assumption here. ∎

Now we turn to our method for solving the one-way Hausdorff distance decision problem.

Theorem 3 *Given ε and planar point sets A and B as above, the one-way Hausdorff distance decision problem under Euclidean motion can be solved in time $O(m^3n^2\log mn)$.*

Proof. In order to solve this decision problem we must determine whether or not the set $S(\theta)$ is uniformly empty or not for all $\theta \in [0, 2\pi]$; that is, we are looking for a θ at which all the sets $B_i^\varepsilon(\theta)$ intersect, if such a θ exists. It is possible that $S(0)$ is non-empty, but this can be determined by the cubic-time algorithm of [21] for computing the minimum Hausdorff distance under translation. Otherwise, if $S(\theta)$ is ever to become non-empty, it will be at a combinatorial change to the arrangement $\mathcal{A}(\theta)$. Thus, by Lemmas 1 and 2, it is enough to examine all $O(m^3n^2)$ combinatorial changes to $\mathcal{A}(\theta)$, for $\theta = 0$ to 2π, and determine whether they result in S becoming non-empty.

We first compute a superset of all the double and triple events: we compute the $\binom{m}{2}$ dynamic Voronoi diagrams of the sets $\mathcal{S}_i(\theta) \cup \mathcal{S}_j(\theta)$ (defined as in the previous proof) and

determine all double events. Similarly, by computing all $\binom{m}{3}$ triple dynamic Voronoi diagrams, we can determine all triple events. The difficulty is in deciding which of these combinatorial changes serves to bring about the creation of the non-empty intersection $S(\theta)$ for some θ.

As mentioned above, this can be found by keeping track of the depth of vertices in the dynamically-changing arrangement $\mathcal{A}(\theta)$. We therefore will store all the vertices of the current $\mathcal{A}(\theta)$ in a dynamic dictionary V, implemented, say, using a red-black tree [15, 18, 26]. Each vertex $v \in V$ is labeled with the pair (i, j) such that v is defined by the intersection of circle i and j. In fact, we store such a v twice, once labeled with (i, j) and once with (j, i), and we order vertices in V by the first index of their labels. If there is more than one vertex with first index i, then we break the ties by ordering these vertices by their cyclic orientation around circle i.

Recall that the *depth* of a point $\in \mathcal{R}^2$, at a given θ, is the number of sets $B_i^{\varepsilon}(\theta)$ which contain it. S is non-empty if and only if some point is covered to depth m at some θ. So it is enough to maintain the depths of the vertices of the dynamically changing arrangement, and halt when any of them reach the value m. If this does not happen then $S(\theta)$ is uniformly empty for all $\theta \in [0, 2\pi]$. We maintain the depths as follows. Initially, we compute the depth information for all vertices of all the sets $B_i^{\varepsilon}(\theta)$ at $\theta = 0$. We then sort all double and triple events by θ, in time $O(m^3 n^2 \log mn)$, and consider them in succession. At each triple event up to three vertices are each crossing the boundary of a set $B_i^{\varepsilon}(\theta)$. In constant time, we can determine whether each such vertex is entering or leaving the set it's crossing; if it is entering, we increment its depth, and if it is leaving, we decrement its depth. We also need to swap the positions of $O(1)$ vertices in V, as some cyclic orderings have now changed.

At a double event two vertices are either being created or deleted. If they are being deleted, then we remove them from V, and if they are being created we insert them into V. In the insertion case we must also initialize their depth, which we can do by examining their cyclic neighbors in V (if they have no cyclic neighbors, then their depth is 2).

As noted above, we stop and output "yes" if any vertex v in V has depth m at any point θ in the sweep (indeed, at that point we can output (v, θ) as a representative valid transformation). Otherwise, we will finish and output "no." The time required for the entire process is $O(m^3 n^2 \log mn)$. ∎

For the bidirectional Hausdorff decision problem, we must determine a single Euclidean transformation g such that $h(g(A), B)$ and $h(B, g(A))$ are both less than or equal to ε. For $h(B, g(A))$, we construct and maintain a similar arrangement of unions of discs, and overlay the two families of arrangements on top of each other. We define depth of coverage analogously; in this case there will be a non-empty intersection if and only if the depth of the coverage equals $m + n$. Thus,

Corollary 4 *Given ε and planar point sets A and B as above, the Hausdorff decision problem under Euclidean motion can be solved in time $O((m + n)^5 \log mn)$.*

4. Sets of Line Segments

Now, we turn to the case in which $A = \{a_1, \ldots, a_m\}$ (resp., $B = \{b_1, \ldots, b_n\}$) is a set of non-intersecting (except possibly at endpoints) line segments. Our approach will be similar to that of the previous section. Recall that our goal is to determine if there is a Euclidean motion g such that $g(A)$ lies entirely within B^ε.

The well-versed reader may suspect that we could use a result of Avnaim and Boissonnat [8] to solve this problem, but this does not seem possible. They solve the problem of placing a general polygonal shape P with m sides and corners, in another general polygonal shape Q, with n sides and corners, when P is allowed to translate and rotate. Let us call Q the *environment*. Avnaim and Boissonnat show that all placements of P where P does not intersect the environment can be found in time $O(m^3n^3 \log mn)$. (See [8] for more details.) There are two main differences between our problem and the problem solved in [8], however. First, our environment B^ε consists not only of straight line segments but also of circular arcs. Nevertheless, an extension of the work in [8] to include circular arcs does not appear too difficult. The more important difference is that the placement problem solved in [8] is placement that simply avoids intersection with the environment, while we need to ensure that A is really *within* B^ε. (To be precise, [8] also show how to determine whether P is within Q, for the case when both P and Q are simple polygons.)

Conceptually, we would like to maintain the arrangement of $B_i^\varepsilon(\theta)$'s as θ changes, as in the previous section. The only boundaries maintained would be outer boundaries of $B_i^\varepsilon(\theta)$ for each i. Our goal is to determine if there is a θ for which some region of this arrangement is covered by all m of the $B_i^\varepsilon(\theta)$'s. It may help to think of each $B_i^\varepsilon(\theta)$ as corresponding to a distinct color; our goal is to find a region covered by all the colors. This straightforward idea leads to some difficulties in maintaining coverage information (see Figure 4): as θ changes, (1) parallel boundaries can intersect and (2) islands can form as part of a single $B_i^\varepsilon(\theta)$. Intersecting parallel boundaries seem to require a large number of updates all along their intersection, while islands seem to require some kind of point location to maintain coverage information.

We avoid these difficulties by maintaining more information in our arrangement. Instead of simply maintaining the outer boundaries of the $B_i^\varepsilon(\theta)$'s, we maintain all the boundaries of all the objects (circles and line segments) used to build each $B_i^\varepsilon(\theta)$ (see Figure 5). Consider the set consisting of every arc or segment that ever appears on an outer boundary of some $B_i^\varepsilon(\theta)$. Each such arc is a portion of a circle; each circle is generated by a pair of endpoints, one from A and one from B. We keep track of all $O(mn)$ such circles over all θ. Each segment that appears as part of some $B_i^\varepsilon(\theta)$ is generated by a segment or endpoint of A and a segment or endpoint of B^ε. Again, there are $O(mn)$ such segments and we keep track of all of them over all θ. Let W be the set of all $O(mn)$ such circles and segments.

Some portions of these segments and circles of W are not useful boundaries in the sense that they are not outer boundaries of $B_i^\varepsilon(\theta)$ for some i. Note though that W includes all of the useful boundaries. The useful boundary portions are labeled to indicate i (i.e., the color, say green) and which side of the boundary is inside (colored green) and which side is outside (no color or plain) of $B_i^\varepsilon(\theta)$. We sketch how this labeling is maintained below.

First we build the arrangement of W for $\theta = 0$ and use a standard line-sweep to determine coverage for each region of this arrangement in time $O(m^2n^2 \log mn)$ (see [1]). We assume

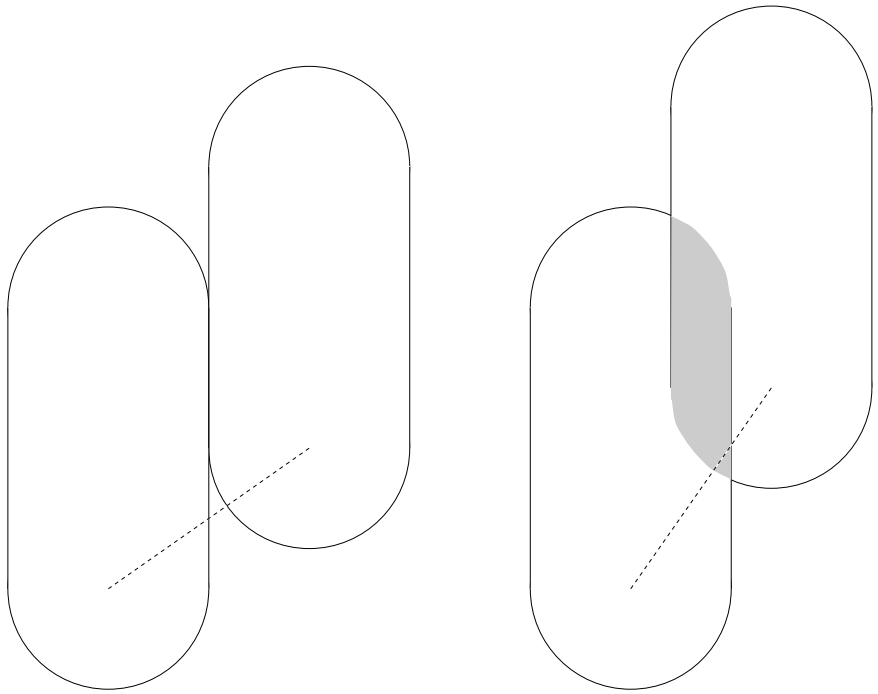

Figure 4: As θ changes, parallel boundaries intersect creating an island.

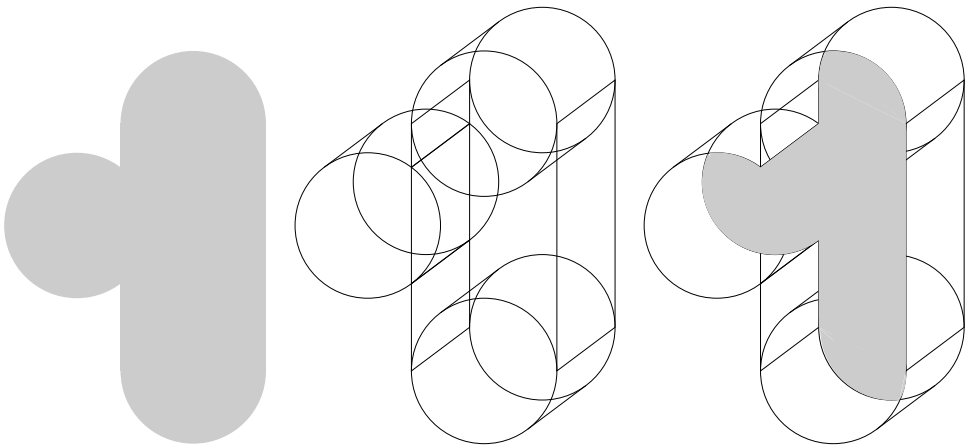

Figure 5: A simple B^ε, all the boundaries used to build $B_i^\varepsilon(\theta)$ and $B_i^\varepsilon(\theta)$ itself (shown shaded).

that each region of the arrangement has a counter indicating its depth of coverage. As θ changes, the arrangement changes. Updates on coverage information are needed whenever a region in the arrangement is created or destroyed. Just as for the point-case, there are two types of events that create or destroy regions: (1) double events and (2) triple events. Because W consists of $O(mn)$ objects with each object moving on a circular path, there are at most $\binom{mn}{2}$ double events and at most $\binom{mn}{3}$ triple events, or $O(m^3n^3)$ events all together. We can preprocess our set W to determine all the possible events and their corresponding θ; then we can sort the events by θ. Now we know the order in which our events occur as θ changes in the arrangement of W. We claim that coverage information can be maintained with only logarithmic search time followed by constant time update per event.

To locate events within W, we maintain a balanced tree for each segment and circle. Given a new event, we assume that adjacent intersections in the arrangement W can be found in $O(\log mn)$ time.

We make use of the labeling of useful boundaries mentioned above. When a new region is created due to a double event or a triple event, its coverage counter can be initialized by examining an adjacent region and the boundary between the new region and the adjacent region. If the boundary is labeled not-useful (i.e., it is not part of an outer boundary of some $B_i^\varepsilon(\theta)$ for some i), then the coverage counter for the new region has the same value as the one in the adjacent region. If the boundary is labeled useful (say its color is green) then the new counter is either one more or one less than the counter in the adjacent region depending on which side of the boundary is labeled green and which side is labeled plain.

We still must show how the labeling of useful boundaries is maintained. Consider just the portion of W that is associated with $B_i^\varepsilon(\theta)$; call this portion W_i. We can use W_i to track changes in $B_i^\varepsilon(\theta)$ as θ changes. Note that W_i consists of just $O(n)$ circles and segments. It is easy to see that there are at most $O(n^3)$ combinatorial changes (double events or triple events) in the arrangement of W_i as θ changes. By tracking these events as θ changes we can easily keep track within an individual W_i of the useful and not-useful boundaries and the additional color labeling.

Now each portion of a boundary (outer and inner, useful and not-useful) in the entire arrangement of all of W corresponds to a (usually larger) boundary in some W_i. So our solution is to link each (small) boundary in the arrangement of W to its corresponding (larger) boundary in W_i for the appropriate i. The labels "useful" and "not-useful" are maintained in all of the W_i arrangements, requiring a total of $O(mn^3)$ updates. The necessary linking information is updated whenever double or triple events occur in the W arrangement.

We maintain $m+1$ arrangements, one for W and one for each W_i. We have $O(m^3n^3)$ update events in the W arrangement and $O(mn^3)$ update events among all the W_i arrangements where a single update takes $O(\log mn)$ time. Thus, we have the following theorem.

Theorem 5 *Given ε and sets of segments A and B as above, the one-way Hausdorff decision problem under Euclidean motion can be solved in time $O(m^3n^3 \log mn)$.*

To solve the bidirectional Hausdorff decision problem, we must maintain a single large arrangement consisting of two overlapping smaller arrangements (one for $h(g(A), B)$ and one for $h(B, g(A))$) to determine if there is a θ at which coverage reaches $m + n$. The above techniques can be used to decide this, giving the following corollary.

Corollary 6 *Given ε and sets of segments A and B as above, the Hausdorff decision problem under Euclidean motion can be solved in time $O((m+n)^6 \log mn)$.*

For sets A and B consisting of filled-in triangles (and segments and points), time bounds for the various problems are the same as for segments. To see this, consider the sets A' and B' constructed by throwing out the triangle interiors and converting each triangle into three segments. A' and B' are used to construct the set W of circles and segments that we used in developing the bounds for segments. Note that these same circles and segments include all the boundaries for the original problem involving filled-in triangles. The only thing that changes is the definition of what is inside and outside. Moreover, once we can solve the problem for filled-in triangles, we can solve it for arbitrary polygons by simply triangulating each polygon in A and B as a preprocessing step, by say the method of Chazelle [9] or Garey *et al.* [17].

5. The Minimization Problem

Having provided methods for solving Hausdorff decision problems under Euclidean motion, we now return to the minimization problem. Here, we apply the parametric searching technique [23, 12, 13, 14, 1, 10], a powerful tool for efficiently solving a variety of optimization problems. This tool has been applied successfully to several problems in computational geometry [1, 10]. See [23, 1, 10] for a discussion of parametric search and the details of how it is applied. Parametric search applies to problems that are parameterized by some real parameter ε. We assume we have a sequential algorithm A_s that decides our particular parameterized problem (for a specific ε) in T_s steps. For us, A_s is a decision problem for the one-way Hausdorff distance as discussed in the previous sections. We wish to find the smallest value ε^* such that A_s still has a positive solution. Suppose, in addition to A_s, we have a parallel comparison-based sorting method A_p that is parameterized by ε and runs in T_p steps using P processors [13]. In our case, since we wish to sort all the double and triple events, a "comparison" involves testing if a given (double or triple) event (that is parametrized by ε) comes before or after (in terms of the θ sweep) another event. The important property here is that ε^* is a critical value for both the sorting algorithm and our sequential algorithm A_s, ensuring that it will be discovered during the parametric search. To resolve the comparisons in any given step of A_p we use A_s and perform a binary search on the set of critical ε's needed to determine the value of the comparisons in this step. This requires $O(P + T_s \log P)$ time and allows us to then proceed with the next step of A_p. Upon completing our simulation of A_p we will have a value for ε^* (by taking the left endpoint of the final interval constraint), and the total time required will be $O(T_p P + T_s T_p \log P)$.

In our case, however, we can do even better than this. In particular, if the parallel algorithm A_p can be designed for the weaker EREW PRAM[6] so as to run in T_p steps using P processors and $O(P)$ memory cells, then Cole [12, 13] shows how one can implement this simulation in $O(T_p P + T_s(T_p + \log P))$ time. Since we can use Cole's sorting algorithm to order the double and triple events in the EREW PRAM model [13], we therefore derive the following:

[6]This is the synchronous shared memory model where all memory accesses, whether reads or writes, are exclusive.

Theorem 7 *Given point sets A and B as above, the minimum one-way Hausdorff problem under Euclidean motion can be solved in time $O(m^3n^2\log^2 mn)$; the minimum (bidirectional) Hausdorff problem under Euclidean motion can be solved in time $O((m+n)^5\log^2 mn)$. For segments, the corresponding one-way and bidirectional problems take time $O(m^3n^3\log^2 mn)$ and time $O((m+n)^6\log^2 mn)$, respectively.*

6. Conclusion

We conclude with an open problem. When we let $m = O(n)$ there is (ignoring log factors) an n^3, n^4, n^5, n^6 progression on the time bounds for the various problems: points under translation, segments under translation, points under rigid motion, and segments under rigid motion, respectively. There are lower bound constructions showing that the number of combinatorially distinct possible matches for these problems are $\Omega(n^3)$, $\Omega(n^4)$, $\Omega(n^5)$, and $\Omega(n^6)$, repectively (the first lower bound was shown originally in [21]; constructions for all the lower bounds appear in [25]). Note that even though the number of combinatorially distinct possible matches for the problem of points under translation is $\Omega(n^3)$ with the L_1 metric, Chew and Kedem [11] show a way to get around this bound, solving this problem in time $O(n^2\log^2 n)$. Their method uses a technique that avoids examining the entire set of combinatorially distinct possible matches, however it appears to depend critically on the L_1 metric. Thus it remains open whether there is a way to achieve a similar improvement for points under translation with the more-standard L_2 metric. It also is open whether there are similar techniques for rigid motion, and for sets of line segments.

References

[1] P.K. Agarwal, M. Sharir, and S. Toledo, "Applications of parametric searching in geometric optimization," *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, 1992, 72–82.

[2] H. Alt, B. Behrends, and J. Blömer, "Measuring the resemblance of polygonal shapes," *Proc. 7th ACM Symp. on Comp. Geom.*, 1991, 186–193.

[3] H. Alt and M. Godau, "Measuring the resemblance of polygonal curves," *Proc. 8th ACM Symp. on Comp. Geom.*, 1992, 102–109.

[4] H. Alt, K. Mehlhorn, H. Wagener and E. Welzl, "Congruence, similarity and symmetries of geometric objects," *Disc. and Comp. Geom.*, **3**, 1988, 237–256.

[5] H. Aonuma, H. Imai, K. Imai and T. Tokuyama, "Maximin location of convex objects in a polygon and related dynamic Voronoi diagrams," *Proc. 6th ACM Symp. on Comp. Geom.*, 1990, 225–234.

[6] E.M. Arkin, L.P. Chew, D.P. Huttenlocher, K. Kedem, and J.S.B. Mitchell, "An efficiently computable metric for comparing polygonal shapes," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 13, no. 3, 1991, pp. 209–216.

[7] E.M. Arkin, K. Kedem, J.S.B. Mitchell, J. Sprinzak and M. Werman, "Matching points into noise regions: combinatorial bounds and algorithms," *ORSA Journal of Computing*, 4(4)(1992), pp. 375–386.

[8] F. Avnaim and J.-D. Boissonnat, "Polygon placement under translation and rotation," *Proc. 5th Symp. Theoret. Aspects Comput. Sci.*, Lecture Notes in Computer Science, **294**, Springer-Verlag, 1988, 322–333.

[9] B. Chazelle, "Triangulating a simple polygon in linear time," *Disc. and Comp. Geom.*, **6**, 1991, 485–524.

[10] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir, "Diameter, width, closest line pair, and parametric searching," *Proc. 8th ACM Symp. on Comp. Geom.*, 1992, 120–129.

[11] L.P. Chew and K. Kedem, "Improvements on approximate pattern matching problems," *3rd Scandinavian Workshop on Algorithm Theory*, Lecture Notes in Computer Science, **621**, Springer-Verlag, 1992, 318–325.

[12] R. Cole, "Slowing down sorting networks to obtain faster sorting algorithms," *J. ACM*, **34**(1), 1987, 200–208.

[13] R. Cole, "Parallel merge sort," *SIAM J. Comput.*, **17**(4), 1988, 770–785.

[14] R. Cole, J. Salowe, W. Steiger, and E. Szemerédi, "An Optimal-Time Algorithm for Slope Selection," *SIAM J. Comput.*, **18**, 1989, 792–810.

[15] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, MIT Press (Cambridge, Mass.: 1990).

[16] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, NY, 1987.

[17] M.R. Garey, D.S. Johnson, F.P. Preparata, and R.E. Tarjan, "Triangulating a simple polygon," *Information Processing Letters*, **7**(4), 1978, 175–179.

[18] L.J. Guibas and R. Sedgewick, "A dichromatic framework for balanced trees," *Proc. 19th IEEE Symp. on Foundations of Computer Science*, 1978, 8–21.

[19] P.J. Heffernan and S. Schirra, "Approximate decision algorithms for point set congruence," *Proc. 8th ACM Symp. on Comp. Geom.*, 1992, 93–101. Unpublished manuscript, 1991.

[20] D.P. Huttenlocher, K. Kedem, and J. Kleinberg, "On dynamic Voronoi diagrams and the minimum Hausdorff distance for point sets under Euclidean motion in the plane" *Proc. 8th ACM Symp. on Comp. Geom.*, 1992, 110–119.

[21] D.P. Huttenlocher, K. Kedem, and M. Sharir, "The upper envelope of Voronoi surfaces and its applications," *Disc. and Comp. Geom.*, vol. 9, no. 3, pp. 267–291.

[22] K. Imai, S. Sumino, and H. Imai, "Minimax geometric fitting of two corresponding sets of points," *Proc. 5th ACM Symp. on Comp. Geom.*, 1989, 266–275.

[23] N. Megiddo, "Applying parallel computation algorithms in the design of serial algorithms," *J. ACM*, **30**(4), 1983, 852–866.

[24] F.P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, NY, 1985.

[25] W.J. Rucklidge, "Lower bounds for the complexity of the Hausdorff distance," to appear in *Proc. 5th Canadian Conf. on Comp. Geom.*, 1993.

[26] R.E. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, PA 1983.