Calculating the Hausdorff Distance Between Curves

by

E. Belogay
School of Mathematics
Georgia Institute of Technology
Atlanta, GA 30332

C. Cabrelli, U. Molter
School of Mathematics
Georgia Institute of Technology
and
Department of Mathematics
Univ. of Buenos Aires

and

R. Shonkwiler
School of Mathematics
Georgia Institute of Technology
e-mail: shonkwiler@math.gatech.edu

1. Introduction

Given two bounded sets A and B in Euclidean space E^m, we define $d_B(A)$ to be

$$d_B(A) = \sup_{a \in A} d(a, B) \tag{1}$$

where the distance from point a to set B is given by

$$d(a, B) = \inf_{b \in B} d(a, b). \tag{2}$$

The point distance $d(a, b)$ may be any distance in E^m, such as for example an ℓ_p distance. We refer to (2) as the *minB calculation*. In general $d_B(A) \neq d_A(B)$. Letting $\text{Ball}_b(\epsilon)$ denote the closed ball of radius ϵ centered at point b, the *Minkowski ϵ-sausage* of B, denoted by B_ϵ, is the set

$$B_\epsilon = \bigcup_{b \in B} \text{Ball}_b(\epsilon).$$

So $d_B(A)$ is equal to the smallest ϵ such that A is contained in the ϵ-sausage of B.

Sets A and B could be the graphs of curves. Then, in 2-dimensional space, $A = \{(x(t), y(t)) : t \in [0, 1]\}$ for some parameterized curve $\gamma : t \in [0, 1] \to (x(t), y(t))$. Similarly for B.

The Hausdorff distance between A and B will be taken as

$$h(A, B) = d_B(A) + d_A(B).$$

This distance between 2-dimensional sets is important in image processing in which the sets are pixelized objects residing in a grid of $M \times N$ pixels or cells. Two objects A and B in a black and white image are identical iff the Hausdorff distance between them is 0. Further, if one object is the translate of the other by a distance t, $B = A + t$, then $h(A, B) = 2t$. Letting $|A|$ denote the number of pixels, or cardinality, of A, a straightforward computational implementation of (1) calculates $d_B(A)$, and hence $h(A, B)$ also, in $O(|A||B|)$ time. We refer to this as the *Direct* algorithm. However, it is possible to calculate $h(A, B)$ for discretized binary sets in time proportional to the frame size, MN, in two dimensions (see (Shonkwiler, 1990) for the case of ℓ_1 point distances and (Shonkwiler, 1991) for the general ℓ_p case). We refer to these (collectively) as the *Field* algorithm, since the main idea is to step from one cell to the next over the pixel grid.

Now suppose sets A and B are (discretized) curves. Their Hausdorff distance may be computed by the above mentioned algorithms, but neither is efficient. In particular the Field algorithm is inefficient since a curve is generally a sparse subset of the complete pixel grid. In fact, if both A and B have Hausdorff dimension 1, then $O(|A||B|) = O(MN)$. But for such 1-dimensional curves, one would think that an algorithm linear in their arc-lengths $|A| + |B|$ should be possible.

In this paper, we give a new algorithm for calculating (1) with an average complexity of

$$\log(\max(M, N))(|A| + |B|) \tag{3}$$

when an ℓ_p point metric with $p \neq 1$ and $p \neq \infty$ is used. (There are special configurations of the sets for which (3) is violated when the point metric is ℓ_1 or ℓ_∞.) We refer to it as the *Scaling algorithm*, for its main idea is to refine an approximation of the distance $h(A, B)$ by rescaling the resolution and doing the direct calculation for only a small subset of pairs $a \in A$ and $b \in B$, which we call *bridges*. Unlike the Field algorithm, whose running time is independent of A and B, the Scaling algorithm's running time varies with the number of points in the sets, as estimate (3) dictates. For curves, or more generally, sparse subsets, and certain other subsets with widely separated points, the Scaling algorithm is faster. The algorithm works, with only minor changes, for the various ℓ_p point metrics. The Scaling algorithm may be adapted to any space dimension; its advantage for sparse sets increases with dimension. Finally, besides calculating the distance (1), the algorithm also produces the points on the curves at which this distance is achieved.

In the next section, we present the Scaling algorithm in detail for two dimensions, but here we outline the main ideas. For the purposes of computation, we assume the discretized curves A and B live in an $N \times N$ square of cells, where N is a power of 2, $N = 2^R$. This may entail embedding the given frame in a larger one. The algorithm proceeds in stages $r = 0, 1, \ldots, R$ and begins with the entire space containing A and B as one large block. Processing at a given stage consists of these steps:

1 curve rescaling;
2 bridge updating;
3 minB (local) pruning, i.e. rescaling the $d(a, B)$ candidates;
4 maxd (global) pruning, i.e. rescaling the $d_B(A)$ candidates.

Beginning with the crudest resolution possible, a single 1×1 block, processing works in stages toward the highest resolution. In this way, many portions of both curves are identified early on as non-contenders in figuring into the calculation of $d_B(A)$. The remaining contenders are kept for further processing, the bridges, consisting of candidate pairs of blocks, or *bridgeheads*, from the representatives of A and B at the given resolution. To carry out the updating, the blocks serving as bridgeheads are refined according to their respective curves; this is step 1. Next, step 2: from the refined bridgeheads, all possibilities for new bridges are considered, but many of them can be eliminated as contenders. In step 3, the list is pruned by invoking equation (2) for each A bridgehead; this is a local comparison. In step 4, a global pruning is carried out based on equation (1), and every bridge length is compared with the current maximum.

Following the detailed explanation of these steps, we give some intermediate results of the algorithm and the number of bridges at each stage, in a typical application.

2. Algorithm for 2D sets

In the following discussion, we show how to calculate $d_B(A)$. The calculation of $d_A(B)$ is similar. To simplify the distance calculations, we use the ℓ_1 point metric but not in any essential way; any ℓ_p may be substituted. As noted above, if A and B are properly engineered, the work will be excessive but the algorithm functions nonetheless. We assume that curve A is discretized and given as a sequence $\alpha_1, \alpha_2, \ldots, \alpha_{|A|}$, of ordered pairs $\alpha_i = (x_i^{(a)}, y_i^{(a)})$, $i = 1, \ldots, |A|$, identifying cells of a uniform spaced $2^R \times 2^R$ grid. The number

$|A|$ of points we take as the *arc-length* of A. Curve B is likewise given. For the purpose of calculating the distance between curves, the order of their pixels is irrelevant, so they may be regarded as sets if desired. Unlike sets, a curve may cross itself and accordingly, some cells may be repeated in the sequence. But subsequent processing of the algorithm removes these duplications.

As stated above, the algorithm proceeds in stages $r = 0, 1, \ldots, R$. Stage $r = 0$ consists of the entire space as a single 1×1 block whose coordinates are $(0,0)_0$ and a single bridge whose A bridgehead is $(0,0)_0$ and whose B bridgehead is the same. Processing now proceeds to the next stage.

Curve rescaling

Each stage begins with a rescaling in which the block size is halved in each direction thereby increasing the number of blocks by a factor of 4 but decreasing the number of pixels per block by a factor of $1/4$. Discretized versions A_r and B_r of A and B are calculated at the new resolution. A block at stage r is a point a_r of A_r iff some cell of the block contains a point of A.

The rescaling entails a doubling of coordinates, adding one more binary digit of precision. More exactly, the block whose coordinates were $(x,y)_r$ becomes 4 smaller blocks whose various quadrants become: SouthWest $(2x, 2y)_{r+1}$, SouthEast $(2x+1, 2y)_{r+1}$, NorthWest $(2x, 2y+1)_{r+1}$, and NorthEast $(2x+1, 2y+1)_{r+1}$.

Although this calculation may proceed conceptually at the start of each new stage, it is more convenient to do the calculations of the A_r and B_r, $r = 0, \ldots, R$, in their entirety beforehand. The calculation is organized as, and the intermediate sets kept, in a quadtree. The root of the A-quadtree contains the data $(0,0)_0$, the initial block, and four pointers labeled SW, SE, NW, and NE. In general, a node at stage r corresponding to an occupied block has non-null quadrant pointers only for those sub-quadrants that are themselves occupied at stage $r+1$. Clearly, at least one such sub-quadrant will be occupied (except at stage R). The time for the calculation of both A and B quadtrees is easily seen to be $O(\log(N)(|A|+|B|))$. The quadtree representation will also prove useful for the temporary storage of minB values corresponding to each a_r.

Bridge updating

With A_r and B_r in hand, the next step at stage r is the updating of the previous bridges in such a way as to keep those that will potentially give the distance $d_B(A)$.

The original bridge, at stage $r = 0$, is $A : (0,0)_0$ to $B : (0,0)_0$. Bridges are implemented as a doubly linked list. In addition to the bridgeheads, each node of this list also holds references to the A and B quadtrees corresponding to these bridgeheads; this facilitates bridge updating. From the reference to the A quadtree, each of the occupied sub-quadrants of the A bridgehead will potentially be an A bridgehead at the next stage. The same holds for the subsequent B-bridgeheads. Initially a new bridge is created for every such pairs so up to 16 are possible; we refer to this as *bridge splitting*. But eventually it may turn out that a bridge no longer has the potential for giving $d_B(A)$ and can safely be deleted from the list. This is the purpose of the *minB* and *maxd* pruning steps.

MinB pruning

Suppose at resolution r a bridge exists between A-block $(0,0)_r$ (without loss of generality) and B-block $(x,y)_r$, $x,y > 0$, i.e. $(x,y)_r$ is a contender for $d(0,B)$ at stage r. Then the ℓ_p distance at this resolution is $\|(x,y)\|_p$. Advancing to the next resolution, assume that only the SW quadrant of the $(0,0)_{r+1}$ block is occupied by a point of A, while all the subquadrants of the $(x,y)_r$ block contain an occupied cell of B at the new resolution. These blocks are now, respectively, $(2x,2y)_{r+1}$, $(2x+1,2y)_{r+1}$, $(2x,2y+1)_{r+1}$, and $(2x+1,2y+1)_{r+1}$. The ℓ_p "circle" of radius $s = \|(2x,2y)\|_p$ centered at $(0,0)_{r+1}$ passes diagonally (in the ℓ_1 case) through the SW corner of the original B block, the circle of radius $s+1$ passes through the NW and SE blocks, and the circle of radius $s+2$ passes through the NE block. If this is the final resolution, then under the "minimum B" rule, equation (2), only the SW sub-quadrant is relevant. But otherwise, note that the upper triangle of the SW sub-quadrant and both the lower triangles of the NW and SE sub-quadrants lie between the s and $s+1$ circles. Therefore the points of those quadrants cannot be automatically eliminated at this step. and new bridges must be created to handle them in subsequent steps. However, at most 3 new bridges suffice.

On the other hand, the bridge from $A : (0,0)_{r+1}$ to the point in the NE quadrant, $(2x+1,2y+1)_{r+1}$, can be eliminated from further consideration since that block lies entirely outside the circle of radius $s+1$. Thus, a difference in distance by 2 for all bridges having the same A bridgehead is sufficient for elimination. These considerations prove the following.

Proposition. *For all bridges having the same A bridgehead, those whose lengths exceed the local minimum by more than 1 will not figure in the calculation of* $d_B(A)$.

To obtain the minimum B distance for a given A_r block, the temporary value can be maintained in the A-quadtree whose pointer is already available in the bridge node. While minB pruning is carried out, the stage r value of $d_B(A)$ can be calculated. This is computed by equation (1) and is the value that would be reported if r were the last stage. We refer to this value as *maxd*.

Maxd pruning

In addition to minB pruning, a bridge at stage r may have a length sufficiently smaller than the current maximum distance, maxd_r, that it too can be eliminated from further consideration. A bridge is eliminated in this step if its length is less than $\text{maxd}_r - 3$.

Proposition. *Let* $d = |a_r - b_r|$ *be the* ℓ_p *length of the bridge* (a_r, b_r) *at stage* r. *Then the length* d' *of every bridge that may be formed at stage* $r+1$ *from the subquadrants of* a_r *and* b_r *satisfies*

$$2d - 2 \le d' \le 2d + 2.$$

Proof. Assume without loss of generality that $a_r = (0,0)$ and $b_r = (x,y)_r$, $x,y > 0$. Then $d = \|(x,y)_r\|_p$. With the above notation, it is easy to see that the *shortest* bridge at step $r+1$ is the one joining the points $(1,1)$ and $(2x,2y)$, and its length is

$$\|(2x,2y) - (1,1)\| \ge 2\|(x,y)\|_p - \|(1,1)\|_p \ge 2d - 2,$$

since $\|(1,1)\|_p \leq 2$.

On the other hand the *longest* bridge is the one joining $(0,0)$ with $(2x+1, 2y+1)$; its length is

$$\|(2x+1, 2y+1)\| \leq 2\|(x,y)\|_p + \|(1,1)\|_p \leq 2d + 2,$$

which yields the desired result.

Corollary. *If $d(a_r, B_r) \leq \text{maxd}_r - 4$ for $a_r \in A_r$, then no bridge derived from a_r will figure in the calculation of $d_B(A)$.*

Proof. Consider a bridge whose length d is at most $\text{maxd}_r - 4$. In the worst case, the bridge corresponding to maxd_r rescales minimally, to $2\text{maxd}_r - 2$, while the one corresponding to d rescales maximally, to $2d + 2 \leq 2(\text{maxd}_r - 4) + 2 = 2\text{maxd}_r - 6$. Thus, at the next stage their separation is still 4 or more.

Complexity

As already seen, the quadtree calculation requires $O(\log(N)(|A| + |B|))$ time. The minimal execution time occurs when both A and B are singletons with points in opposite corners of the frame. Then, only $O(1)$ time need be expended at each stage for the other processing steps. We now turn to the configuration of A and B for maximal execution time.

Processing at every stage is proportional to the number of bridges, and we show that the number of bridges is at most $2(|A| + |B|)$. In order to do this, we analyze the problem from a graph-theoretical viewpoint: Let G be the bipartite graph with vertex set $V = A \cup B$ and edge set $E = \{(a,b) : a \in A, b \in B, \text{ and } (a,b) \text{ is a bridge }\}$. Assume, without loss of generality, that (a,b) is a bridge iff $d(a,b) = d(a,B)$ or $d(a,b) = d(b,A)$ which we take to be 1. We therefore want to show that the number of edges (bridges) is at most 2 times the number of vertices. We first need the following definitions, cf. (Jackson and Thoro, 1990).

Definitions. *A region associated with a planar graph G is a connected component of the complement $E^2 \backslash G$. The degree of such a region is the number of edges on its boundary.*

If both sides of an edge are on the boundary of the same region, the edge should be counted twice, e.g. if $V = \{a, b\}$ and $E = \{(a,b)\}$, then $\deg(E^2 \backslash E) = 2$.

Theorem 1. *$|E| \leq 2|V| - 3$ for ℓ_p distance in E^2, where $1 < p < \infty$.*

The proof consists of several lemmas. First we show that no two edges intersect, so G is planar. Then we observe that each region has an even number of edges. Finally we use the Euler's formula to achieve the inequality $|E| \leq 2|V| - 3$ for each connected component of G and add the results. The theorem doesn't hold for ℓ_1 and ℓ_∞ distance in E^2.

Lemma 1. *No two edges of the graph G intersect.*

Proof. Suppose that the edges between a_1 and b_1 and between a_2 and b_2 intersect in the (non-vertex) point c, where $a_i \in A$, $b_i \in B$, $i = 1, 2$. Then $d(a_1, c) + d(c, b_1) = 1 = d(a_2, c) + d(c, b_2)$. Adding up the two equalities and combining the result with the triangle inequalities for the triangles $\{a_1, c, b_2\}$ and $\{a_2, c, b_1\}$, we obtain

$$d(a_1, b_2) + d(a_2, b_1) \leq (d(a_1, c) + d(c, b_2)) + (d(a_2, c) + d(c, b_1)) = 1 + 1 = 2.$$

Since $d(a_i, b_j) \geq 1$ for all i, j, it must be that $d(a_1, b_2) = d(a_2, b_1) = 1$, thus forcing equalities in the triangle inequalities and hence the co-linearity of a_1, a_2, b_1, b_2 and c. The latter and the fact that $d(a_i, b_j) = 1$ for $i, j = 1, 2$ implies that $a_1 = a_2$ or $b_1 = b_2$, which is a contradiction.

Remark 1. *Since for any ℓ_p metric in E^2, $1 < p < \infty$, the triangle equality implies co-linearity, the lemma is true for all those metrics. Note again that this fails for $p = 1$ or $p = \infty$.*

Remark 2. *The lemma still holds if we replace E^2 with a finite grid. (In this case the point c is also on the grid.)*

Corollary. *The graph G is planar.*

Consider the connected components of a graph G. Since the sizes of their edge sets add up to the size of the edge set of G, and the same holds true for the vertex sets, it is enough to prove the inequality above for each connected component of G and add up the resulting inequalities. Therefore, without loss of generality we assume from now on that G is a connected component of the graph in the conjecture. Moreover, we can assume that G has at least 2 edges, for if G contains one or no edges the inequality is trivially satisfied.

Thus, G is now a connected planar graph with at least two edges.

Lemma 2. *The degree of each region in G is at least 4.*

Proof. Since G is bipartite, the vertices on the boundary of any region alternate between A and B. Therefore, the number of edges is even and bigger than 2.

Lemma 3. *For a connected bipartite graph with at least 2 edges $|E| \leq 2|V| - 4$.*

Proof. Denote the number of regions by $|R|$ and sum the degrees of all regions. Since each region has degree at least 4 and each edge is counted twice, we obtain that $2|E| =$ sum of the degrees of all regions $\geq 4|R|$, i.e. $|R| \leq |E|/2$. Now plug that inequality into Euler's formula to obtain

$$2 = |V| - |E| + |R| \leq |V| - |E| + |E|/2 = |V| - |E|/2,$$

i.e. $|E| \leq 2|V| - 4$. This proves the theorem.

Remark. *The estimate in the theorem is sharp. It can be achieved exactly and asymptotically. The example is the "chessboard" grid $V = \{(x, y) \in Z^2 : 1 \leq x, y \leq n\}$, $A = \{(x, y) \in V : x + y$ is even$\}$, $B = V \backslash A$, and the edges connect each point with its four (North, South, East, West) neighbors. This bipartite graph has $|V| = n^2$ vertices and $|E| = 2n(n-1)$ edges. In the limit ($n \to \infty$) we have $|E|/|V| \to 2$, maintaining $|E| < 2|V|$. The trivial example of two vertices joined by an edge satisfies $|E| = 2|V| - 3$.*

Counterexample for ℓ_1.

Consider A and B to be the points on two parallel line segments at a 45 degree angle to the x-axis, e.g. $A = \{(x,y) : x+y = 0, -2 < x < 2\}$, $B = \{(x,y) : x+y = 2, -1 < x < 3\}$. The ℓ_1 distance between A and B is exactly 2. Now consider the points in the middle two quarters of each segment: $A' = \{(x,y) \in A : -1 < x < 1\}$, $B' = \{(x,y) \in B : 0 < x < 2\}$. Assume that at a given resolution each of these sets contains n points (about 2^{-r}). Notice that for each point $a \in A'$, exactly n points of B are distance 2 from it. Therefore, there are at least n^2 bridges of length 2 and a total of $4n$ points in $A \cup B$. Thus, the inequality in the conjecture is violated for $n > 8$.

Note: A similar argument (take two line segments parallel to the x-axis) shows that the conjecture fails in the ℓ_∞ case. What distinguishes these two cases from the rest is that the "unit circle" in these metrics is not strictly convex and has flat regions.

An Example

The following table pertains to the calculation of the Hausdorff distance between a circle and a cosine curve at resolution 256×256. There are 8 stages, covering resolutions 1×1 to full resolution. The number of A and B points at each stage is given, followed by the number of bridges and the fractions of points that are bridgeheads.

Table 1 per stage point and bridge data for Cosine to Circle calculation

stage:	1	2	3	4	5	6	7	8
A pts	4	9	21	45	92	180	344	603
B pts	4	12	28	60	124	234	440	762
bridges	12	39	93	193	302	380	427	82
frac A	.85	.68	.45	.30	.19	.11	.06	.015
frac B	.85	.52	.34	.23	.14	.08	.05	.012

References

[1] Jackson and Thoro. Applied Combinatorics with Problem Solving. *Addison–Wesley* (1991), 159-163.

[2] Shonkwiler, R. An image algorithm for computing the hausdorff distance efficiently in linear time, *Inf. Proc. Letters* **30** (1989), 87-89.

[3] Shonkwiler, R. Computing the Hausdorff set distance in linear time for any L(p) point distance. *Info. Proc. Letters* 38 (1991) 201-207.