Commun. Korean Math. Soc. **28** (2013), No. 4, pp. 833–850
http://dx.doi.org/10.4134/CKMS.2013.28.4.833

COMPUTING THE HAUSDORFF DISTANCE BETWEEN TWO SETS OF PARAMETRIC CURVES

Ik-Sung Kim and William McLean

Abstract. We present an algorithm for computing the Hausdorff distance between two parametric curves in $\mathbb{R}^n$, or more generally between two sets of parametric curves in $\mathbb{R}^n$. During repeated subdivision of the parameter space, we prune subintervals that cannot contain an optimal point. Typically, our algorithm costs $O(\log M)$ operations, compared with $O(M)$ operations for a direct, brute-force method, to achieve an accuracy of $O(M^{-1})$.

1. Introduction

Let (X, d) be a metric space, so that $d(x, y)$ is the distance between two points x and $y \in X$. We denote the distance from a point $x \in X$ to a nonempty subset $B \subseteq X$ by

$$d(x, B) = \inf_{y \in B} d(x, y). \tag{1}$$

Given a second, nonempty subset $A \subseteq X$, the *directed Hausdorff distance from A to B* is defined by

$$h(A, B) = \sup_{x \in A} d(x, B),$$

and the *Hausdorff distance between A and B* by

$$H(A, B) = \max\bigl(h(A, B), h(B, A)\bigr).$$

If $x^* \in A$ is such that $h(A, B) = d(x^*, B)$, then we call x^* an *optimal point.* We remark that the Hausdorff distance defines a metric on the set of closed, bounded subsets of X.

In pattern recognition and computer vision, it is very important to compare shapes and patterns, and to give a numerical value indicating their similarity. The Hausdorff distance is a well known similarity measure: the smaller the Hausdorff distance between two shapes the greater is their degree of resemblance. In practice, one often wants to find $\min_{g \in \mathcal{G}} h\bigl(A, g(B)\bigr)$ for some transformation group $\mathcal{G}$, such as the set of rigid body motions.

Received September 26, 2011.
2010 *Mathematics Subject Classification.* Primary 65D18, 68U05.
Key words and phrases. Hausdorff distance, pruning technique.

Several authors have considered methods for computing $h(A,B)$. Alt et al. [1] discussed polynomial-time algorithms when A and B are finite collections of simplices (e.g., line segments, triangles, or tetrahedrons). Belogay et al. [2] considered discretized curves, lying on an $M \times N$ grid of pixels, and devised an algorithm with an average cost of $(|A| + |B|) \log \max(M,N)$, where $|A|$ denotes the number of pixels in A. Bouts [3] considered the problem of computing the minimum Hausdorff distance, $\min_{g \in \mathcal{G}} h\bigl(A, g(B)\bigr)$, where A and B are finite planar sets and $\mathcal{G}$ is the group of translations. Rote [5] discussed a one-dimensional version of the same problem, and Scharf [6] developed an algorithm for computing the Hausdorff distance between planar sets of curves having rational parametric representations.

In this paper, we focus our attention on the problem of computing the Hausdorff distance $H(A,B)$ between two sets of parametric curves, A and B, in n-dimensional Euclidean space. Thus,

$$X = \mathbb{R}^n \quad \text{and} \quad d(x,y) = |x-y| = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

for $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$.

To begin with, we suppose that each of A and B is a single, continuous parametric curve. In Section 2, we consider a direct method for approximating the directed Hausdorff distance $h(A,B)$ using M points in A and N points in B, leading to a computational cost of $O(MN)$ operations.

Section 3 addresses the problem of computing the distance $d(x,B)$ from a point x to a curve B. Subsequently, we treat this problem as solved, and assume that $d(x,B)$ is known exactly.

We proceed in Section 4 to explain an adaptive algorithm for estimating $h(A,B)$, that selectively samples points on A, far fewer than M in total, to reduce the computational cost. The key idea is to reduce the necessary work by "pruning" subintervals of the parameter interval $[a,b]$. In typical cases, our algorithm requires a runtime of only $O(\log M)$, far less than the $O(M)$ needed by the direct method, to achieve a comparable accuracy. In this context, each computation of $d(x,B)$ counts as a single operation.

In Section 5 we extend our algorithm to handle the case when A and B consist of multiple parametric curves.

Finally, Section 6 presents several numerical examples illustrating the performance of our algorithm for different configurations of A and B.

2. Direct method

Suppose that A and B are individual curves, parameterized by some continuous functions

$$f : [a,b] \to \mathbb{R}^n \quad \text{and} \quad g : [c,d] \to \mathbb{R}^n,$$

respectively, so that

$$A = \{\, f(t) : a \le t \le b\} \quad \text{and} \quad B = \{\, g(u) : c \le u \le d\}. \tag{2}$$

By selecting parameter values

$$a \le t_1 < t_2 < \cdots < t_M \le b \quad \text{and} \quad c \le u_1 < u_2 < \cdots < u_N \le d, \tag{3}$$

and setting $x_i = f(t_i)$ and $y_j = g(u_j)$, we obtain two sets of sample points

$$S = \{x_1, x_2, \dots, x_M\} \subseteq A \quad \text{and} \quad T = \{y_1, y_2, \dots, y_N\} \subseteq B.$$

We can then compute an obvious, brute-force, discrete approximation to the directed Hausdorff distance,

$$h(A,B) \approx h(S,T) = \max_{1\le i\le M} \min_{1\le j\le N} |x_i - y_j|, \tag{4}$$

at a cost of $O(MN)$ operations. To estimate the error in the approximation (4), we adopt the convention that $t_0 = a$, $t_{M+1} = b$, $u_0 = c$ and $u_{N+1} = d$, and define the mesh widths

$$\delta_S = \max_{1\le i\le M+1} (t_i - t_{i-1}) \quad \text{and} \quad \delta_T = \max_{1\le j\le N+1} (u_j - u_{j-1}). \tag{5}$$

The proof of the error bound makes use of two technical lemmas. We denote the modulus of continuity of f by

$$\omega_f(\sigma) = \sup\{\, |f(t) - f(t')| : t,\, t' \in [a,b] \text{ with } |t - t'| \le \sigma \,\},$$

and note that $\omega_f(\sigma) \searrow 0$ as $\sigma \searrow 0$ because f is continuous.

Lemma 2.1. *$h(A,S) \le \omega_f(\frac{1}{2}\delta_S)$ and $h(B,T) \le \omega_g(\frac{1}{2}\delta_T)$.*

Proof. Given $x = f(t) \in A$, we can find $t_i \in S$ such that $|t - t_i| \le \frac{1}{2}\delta_S$, so

$$d(x,S) \le |x - x_i| = |f(t) - f(t_i)| \le \omega_f(\tfrac{1}{2}\delta_S),$$

and taking the supremum over $x \in A$ gives the estimate for $h(A,S)$. The estimate for $h(B,T)$ follows in the same way. □

Lemma 2.2. *$d(x,B) \le d(x,T) \le d(x,B) + h(B,T)$ for $x \in \mathbb{R}^n$.*

Proof. The inclusion $T \subseteq B$ immediately implies the left-hand inequality:

$$d(x,B) = \inf_{y\in B} |x - y| \le \inf_{y\in T} |x - y| = d(x,T).$$

To prove the right-hand inequality, let $\epsilon > 0$. First choose $y^* \in B$ such that

$$|x - y^*| < d(x,B) + \epsilon,$$

and then choose $y_j \in T$ such that $|y_j - y^*| = \min_{1\le k\le N} |y_k - y^*| = d(y^*,T)$. In this way,

$$\begin{aligned} d(x,T) &\le |x - y_j| \le |x - y^*| + |y^* - y_j| \\ &< d(x,B) + \epsilon + d(y^*,T) \le d(x,B) + \epsilon + h(B,T). \end{aligned}$$

□

Theorem 2.3. *For the approximation* (4) *we have the error bound*

$$\begin{aligned}\bigl|h(S,T)-h(A,B)\bigr| &\le \max\bigl(h(A,S),h(B,T)\bigr)\\ &\le \max\bigl(\omega_f(\tfrac12\delta_S),\omega_g(\tfrac12\delta_T)\bigr).\end{aligned}$$

Proof. In view of Lemma 2.1, it suffices to show that

$$h(S,T)\le h(A,B)+h(B,T)\quad\text{and}\quad h(A,B)\le h(S,T)+h(A,S).$$

Lemma 2.2 implies

$$d(x_i,T)\le d(x_i,B)+h(B,T)\le h(S,B)+h(B,T),$$

and the first inequality follows by taking the maximum over $x_i\in S$. For the second inequality, let $\epsilon>0$ and note that $h(A,B)\le h(A,T)$ by Lemma 2.2. Choose $x^*\in A$ such that $h(A,T)<d(x^*,T)+\epsilon$, and then choose $x_i\in S$ such that $|x^*-x_i|=d(x^*,S)$. In this way,

$$|x^*-y_j|\le|x^*-x_i|+|x_i-y_j|=d(x^*,S)+|x_i-y_j|$$

and so

$$\begin{aligned}d(x^*,T)=\min_{1\le j\le N}|x^*-y_j| &\le d(x^*,S)+\min_{1\le j\le N}|x_i-y_j|\\ &= d(x^*,S)+d(x_i,T)\le h(A,S)+h(S,T),\end{aligned}$$

implying that $h(A,B)\le h(A,T)<h(A,S)+h(S,T)+\epsilon$. □

3. Distance from a point to a curve

We seek a method for approximating $h(A,B)$ that requires fewer operations than the direct method (4). To do so, we first require an efficient method to compute (1), the distance from a point x to the curve B.

Since $d(x,B)=\min_{c\le u\le d}\phi(u)$, where $\phi(u)=|x-g(u)|$, we can use any standard method for computing the minimum value of a continuous function over a closed, bounded interval. A typical minimization algorithm [4] combines golden-section search with quadratic interpolation to give a reliable method that achieves superlinear convergence if the objective function is C^2 in a neighbourhood of the limit of the iterates. Unfortunately, this limit might be only a *local* minimum. Convergence to a *global* minimum u^* on an interval $[c,d]$ is guaranteed if the objective function ϕ is *unimodal*: strictly decreasing for $c\le u\le u^*$ and strictly increasing for $u^*\le u\le d$.

In our case, we must expect that, in general, ϕ is not unimodal on $[c,d]$. Given a subdivision of the interval $[c,d]$ as in (3), Procedure 3.1 returns a list $\mathcal{I}$ of non-overlapping subintervals such that

$$\min_{u\in\bigcup\mathcal{I}}\phi(u)\le\min_{0\le j\le N}\phi_j,\qquad\text{where }\phi_j=\phi(u_j);$$

see Figure 1. Applying a standard minimization algorithm over each subinterval in $\mathcal{I}$, and taking the least of these computed local minima, we have a candidate for the global minimum.

Procedure 3.1 *SeekUnimodal*(u, ϕ)

```
$\mathcal{I} \leftarrow \emptyset$
**if** $\phi_0 < \phi_1$ **then**
  $\mathcal{I} \leftarrow \mathcal{I} \cup [u_0, u_1]$
**end if**
**for** $j = 1, N-1$ **do**
  **if** $\phi_j < \phi_{j-1}$ and $\phi_j < \phi_{j+1}$ **then**
    $\mathcal{I} \leftarrow \mathcal{I} \cup [u_{j-1}, u_{j+1}]$
  **end if**
**end for**
**if** $\phi_N < \phi_{N-1}$ **then**
  $\mathcal{I} \leftarrow \mathcal{I} \cup [u_{N-1}, u_N]$
**end if**
**return** $\mathcal{I}$
```

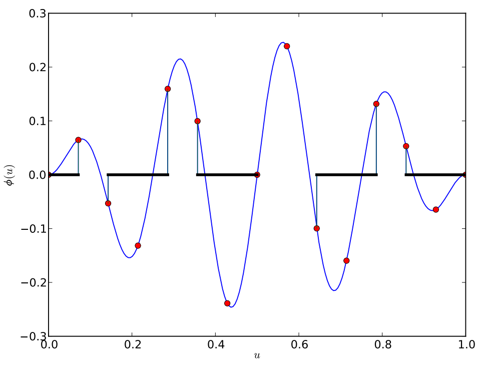


FIGURE 1. The intervals $\mathcal{I}$ found by *SeekUnimodal*.

Heuristically, we expect that if the chosen subdivision of $[c, d]$ is sufficiently fine, in other words if δ_T in (5) is sufficiently small, then

(1) ϕ is unimodal on each subinterval in $\mathcal{I}$,
(2) a global minimum of ϕ occurs in at least one subinterval in $\mathcal{I}$.

However, for any choice of sample parameter values t_j we can easily construct an x and B for which neither 1 nor 2 hold.

Indeed, no finite set of samples can reveal with certainty if ϕ is unimodal on a specified interval unless we know some additional information. For instance, if ϕ is strictly convex, then it must be unimodal.

In some cases, the following result provides bounds on the global minimum of ϕ regardless of whether this function is unimodal.

Theorem 3.1. *If $\phi : [c,d] \to \mathbb{R}$ is a C^1 function and if there is a constant $\Lambda > 0$ such that*

$$\phi'(v) - \phi'(u) \le \Lambda(v-u) \quad \text{for } c \le u < v \le d, \tag{6}$$

then for $\delta = \delta_T$ given by (5) *we have*

$$\min_{c\le u\le d} \phi(u) \le \min_{0\le j\le N+1} \phi(u_j) \le \min_{c\le u\le d} \phi(u) + \tfrac{1}{4}\Lambda\delta^2.$$

Proof. The bound is an easy consequence of Lemma 2.4.1 and Theorem 6.2.1 of Brent [4]. □

The condition (6) holds if, for instance, ϕ is C^2 with $\phi''(u) \le \Lambda$ for $c \le u \le d$.

Unfortunately, we do not know of a reliable automatic way to choose a suitable value for N in Procedure 3.1. In practice, we can make a visual assessment of the length scale of oscillations in the graph of ϕ.

Even if we known an interval that contains a point u^* at which ϕ attains a global minimum and on which ϕ is unimodal, we cannot expect to compute u^* exactly using a system of floating-point arithmetic with fixed relative precision $\epsilon > 0$. If we are willing to perform a sufficient number of steps of the minimization algorithm, then we should be able to compute $\phi(u^*)$ with essentially the full relative accuracy ϵ. However, for u^* itself we have to be content with much lower accuracy, since an uncertainty of order ϵ in the value of $\phi(u^*)$ corresponds to a much larger uncertainty of order $\sqrt{\epsilon}$ when ϕ is C^2 and $\phi'(u^*) = 0$.

We could attempt to compute the directed Hausdorff distance $h(A,B)$ by applying the minimization algorithm, in combination with Procedure 3.1, to the objective function $\varphi(t) = -d\big(f(t),B\big)$, so that $h(A,B) = -\min_{a\le t\le b} \varphi(t)$. However, the problem of choosing N is even worse than in the basic calculation of $d(x,B)$, since the behaviour of φ is difficult to infer from looking at the curves A and B. Instead, in the next section, we prefer an alternative approach.

4. Algorithm based on a pruning technique

Suppose that A and B are continuous parametric curves, and that we have an algorithm for computing $d(x,B)$ exactly for $x \in A$.

We may approximate the directed Hausdorff distance from A to B as follows,

$$h(A,B) \approx h(S,B) = \max_{1\le i\le M} d(x_i,B). \tag{7}$$

The cost of computing $h(S,B)$ will be roughly M times the cost of computing $d(x,B)$ for a point x. By taking $T = B$ in Theorem 2.3, we see that

$$|h(S,B) - h(A,B)| \le \omega_f(\tfrac{1}{2}\delta_S),$$

so we can improve the accuracy of the approximation (7) by adding more sample points to reduce the value of δ_S. However, such a direct computation costs $O(M)$ operations, and achieving high accuracy will be expensive because $\delta_S \ge (b-a)/M$. We therefore propose a new, cheaper algorithm.

When adding sample points, the following simple observation provides a criterion for ignoring parts of A, a process referred to as "pruning", based on a current estimate m for $h(A, B)$.

Lemma 4.1. *Suppose that $x^* \in A$ and $m \geq d(x^*, B)$. If we define*

$$w = m - d(x^*, B),$$

then for $x \in A$,

$$d(x, B) \leq m \quad \textit{whenever} \quad |x - x^*| \leq w.$$

Proof. If $|x - x^*| \leq w$, then for all $y \in B$,

$$|x - y| \leq |x - x^*| + |x^* - y| \leq w + d(x^*, B) = m. \qquad \square$$

From the lemma above, we can prune all points x such that $|x - x^*| \leq w$.

Consider a subinterval $[\alpha, \beta]$ of the parameter interval $[a, b]$, and suppose that $\mathit{MaxDist} \leq h(A, B)$. In Procedure 4.1, we put

$$\mu = \tfrac{1}{2}\alpha + \tfrac{1}{2}\beta, \quad \mathit{Midpt} = f(\mu), \quad \mathit{Dist} = d(\mathit{Midpt}, B),$$

and assume that $\mathit{Dist} \leq \mathit{MaxDist}$. Applying Lemma 4.1 with $x^* = \mathit{Midpt}$, $m = \mathit{MaxDist}$ and $w = \mathit{MaxDist} - \mathit{Dist}$, we see that

$$d(x, B) \leq \mathit{MaxDist} \quad \text{whenever} \quad |x - \mathit{Midpt}| \leq w. \tag{8}$$

Since f is continuous, the set

$$R_{w,\alpha,\beta} = \{\alpha, \beta\} \cup \{\, t \in [\alpha, \beta] : |f(t) - \mathit{Midpt}| = w \,\}$$

is closed in $[\alpha, \beta]$, and we may define

$$\begin{aligned} t_\alpha &= \max\{\, t \in R_{w,\alpha,\beta} : t \leq \mu \,\}, \\ t_\beta &= \min\{\, t \in R_{w,\alpha,\beta} : t \geq \mu \,\}. \end{aligned} \tag{9}$$

If $w = 0$, then $t_\alpha = t_\beta = \mu$. Otherwise,

$$|f(t) - \mathit{Midpt}| < w \quad \text{for } t_\alpha < t < t_\beta,$$

and therefore by (8),

$$d\big(f(t), B\big) \leq \mathit{MaxDist} \quad \text{for } t_\alpha \leq t \leq t_\beta,$$

which means that, when adding new sample points, we may ignore $f(t)$ for $t \in [t_\alpha, t_\beta]$, since such a point is never further from B than the best of the sample points we have already used.

This analysis leads to Procedure 4.1, a pruning algorithm that returns a set $\mathcal{I}$ of 0, 1 or 2 subintervals of $[\alpha, \beta]$, whose union $\bigcup \mathcal{I}$ contains all values of $t \in [\alpha, \beta]$ that might satisfy $d\big(f(t), B\big) > \mathit{MaxDist}$. In other words, if $f(t)$ is a potentially advantageous new sample point with $t \in [\alpha, \beta]$, then t must belong to one of the intervals in $\mathcal{I}$.

Procedure 4.2 starts from the whole interval $[a, b]$ and repeatedly applies the pruning procedure until ariving at a collection of subintervals $\mathcal{I}$, each of which has length smaller than a specified tolerance ϵ.

Procedure 4.1 *Prune*$([\alpha,\beta], \mathit{Midpt}, \mathit{Dist}, \mathit{MaxDist})$

```
Require: $[\alpha,\beta] \subseteq [a,b]$ and $\mathit{Midpt} = f(\frac{1}{2}\alpha + \frac{1}{2}\beta)$
Require: $\mathit{Dist} = d(\mathit{Midpt}, B) \le \mathit{MaxDist}$
Ensure: If $t \in [\alpha,\beta]$ and $d\big(f(t), B\big) > \mathit{MaxDist}$ then $t \in \bigcup \mathcal{I}$
  $w \leftarrow \mathit{MaxDist} - \mathit{Dist}$
  $t_\alpha \leftarrow \max\{\, t \in R_{w,\alpha,\beta} : t \le \frac{1}{2}\alpha + \frac{1}{2}\beta \,\}$
  $t_\beta \leftarrow \min\{\, t \in R_{w,\alpha,\beta} : t \ge \frac{1}{2}\alpha + \frac{1}{2}\beta \,\}$
  $\mathcal{I} \leftarrow \emptyset$
  if $t_\alpha > \alpha$ then
    $\mathcal{I} \leftarrow \mathcal{I} \cup \{[\alpha, t_\alpha]\}$
  end if
  if $t_\beta < \beta$ then
    $\mathcal{I} \leftarrow \mathcal{I} \cup \{[t_\beta, \beta]\}$
  end if
  return $\mathcal{I}$
```

As we will see in the numerical examples of Section 6, Procedure 4.2 exhibits two kinds of convergence behaviour, depending on the nature of the optimal point $x^* = f(t^*) \in A$ where $d(x^*, B) = h(A, B)$. The number of retained subintervals grows like $\sqrt{M}$ if t^* is a stationary point of the function $t \mapsto d\big(f(t), B\big)$, but remains bounded otherwise. In the former case, the CPU time also grows like $\sqrt{M}$, but fortunately because *MaxDist* is then less sensitive to the location of x^* we do not need to reduce *MaxSize* to such a small value; see Example 4 in Section 6.

Theorem 4.2. *The value of MaxDist returned by Procedure 4.2 satisfies the error bound*

$$0 \le h(A, B) - \mathit{MaxDist} \le \omega_f(\tfrac{1}{2}\epsilon).$$

Proof. If $\mathcal{I} = \mathit{Prune}([\alpha,\beta], \mathit{Midpt}, \mathit{Dist}, \mathit{MaxDist})$, then the length of each subinterval in $\mathcal{I}$ is at most $(\beta-\alpha)/2$. Thus, after l iterations of the repeat loop in Procedure 4.2 we have $\mathit{MaxSize} \le 2^{-l}(b-a)$, showing that the termination condition $\mathit{MaxDist} < \epsilon$ must eventually be satisfied. Let $\mathcal{I} = \{[\alpha_j, \beta_j]\}_{j=1}^N$ denote the final family of subintervals, and let $S = \{\mu_j\}_{j=1}^N$ denote the corresponding set of midpoints, so that $\delta_S = \max_{1\le j\le N}(\beta_j - \alpha_j)$ satisfies $\delta_S < \epsilon$. The pruning technique ensures that the set $A_1 = \bigcup \mathcal{I}$ satisfies $h(A_1, B) = h(A, B)$, and by taking $T = B$ in Theorem 2.3 we see that

$$0 \le h(A, B) - \mathit{MaxDist} = |h(A, B) - h(S, B)| \le \omega_f(\tfrac{1}{2}\delta_S) \le \omega_f(\tfrac{1}{2}\epsilon),$$

where we used the fact that the modulus of continuity $\omega_f(\sigma)$ is a monotone increasing function of σ. □

It remains to discuss the calculation of t_α and t_β. Setting

$$\phi(t) = |f(t) - \mathit{Midpt}|^2 - w^2$$

Procedure 4.2 *DirHdfDist*(A, B, ϵ)

```
Require: $A$ and $B$ are curves given by (2)
Require: $\epsilon > 0$
Ensure: $\mathit{MaxDist} \le h(A,B) \le \mathit{MaxDist} + \omega_f(\epsilon/2)$
  $[\alpha_1, \beta_1] \leftarrow [a, b]$; $N \leftarrow 1$; $\mathit{MaxSize} \leftarrow b - a$; $\mathit{Midpt}_1 \leftarrow f(\frac{1}{2}\alpha_1 + \frac{1}{2}\beta_1)$
  $\mathit{Dist}_1 \leftarrow d(\mathit{Midpt}_1, B)$; $\mathit{MaxDist} \leftarrow \mathit{Dist}_1$
  repeat
    $\mathcal{I} \leftarrow \emptyset$
    for $j = 1, N$ do
      $\mathcal{I} \leftarrow \mathcal{I} \cup \mathit{Prune}([\alpha_j, \beta_j], \mathit{Midpt}_j, \mathit{Dist}_j, \mathit{MaxDist})$
    end for
    Redefine $\alpha_j$, $\beta_j$, $N$ so that $\mathcal{I} = \{\, [\alpha_j, \beta_j] : 1 \le j \le N \,\}$
    for $j = 1, N$ do
      $\mathit{Midpt}_j \leftarrow f(\frac{1}{2}\alpha_j + \frac{1}{2}\beta_j)$
      $\mathit{Dist}_j \leftarrow d(\mathit{Midpt}_j, B)$
      $\mathit{MaxDist} \leftarrow \max(\mathit{Dist}_j, \mathit{MaxDist})$
    end for
    $\mathit{MaxSize} \leftarrow \max\{\, \beta_j - \alpha_j : j = 1, 2, \ldots, N \,\}$
  until $\mathit{MaxSize} < \epsilon$
  return $\mathit{MaxDist}$
```

the problem is to find the solutions of $\phi(t) = 0$ nearest to μ and within the interval $[\alpha, \beta]$, if they exist. It suffices to consider the case $w > 0$, for which $\phi(\mu) < 0$.

Given $\gamma \in [\mu, \beta]$ such that $\phi(\gamma) \ge 0$, we may compute t_β, with any desired accuracy up to essentially the full relative precision ϵ, using a standard rootfinding procedure. For instance, the Brent–Dekker algorithm [4] uses a combination of the bisection, secant and inverse quadratic interpolation methods, and achieves superlinear convergence provided ϕ has a nonzero derivative at the root. The chief risk is that the interval $(\mu, \gamma]$ contains more than one root, because in this case the rootfinding iteration may converge to the wrong one. Procedure 4.3 uses a very simple, linear search to obtain $\gamma = \mathit{FirstCross}(\phi, \mu, \beta, n)$; the larger the value of n, the less the chance of skipping over a sign change. If no γ is found, then $t_\beta = \beta$.

Similarly, to compute t_α we solve $\phi(t) = 0$ for t in the interval $[\gamma, \mu]$, with $\gamma = \mathit{FirstCross}(\phi, \mu, \alpha, n)$, unless no such γ is found, in which case we put $t_\alpha = \alpha$.

5. Sets of parametric curves

We now consider the general case when A and B consist of multiple parametric curves, that is,

$$A = A^1 \cup A^2 \cup \cdots \cup A^P \quad \text{and} \quad B = B^1 \cup B^2 \cup \cdots \cup B^Q, \tag{10}$$

Procedure 4.3 *FirstCross*$(\phi, \mathit{Start}, \mathit{Finish}, n)$

```
Require: $\phi(\mathit{Start}) < 0$
  $\Delta t = (\mathit{Finish} - \mathit{Start})/n$
  $\mathit{SignChange} =$ false
  for $k = 1, n$ do
    $\gamma = a + k\,\Delta t$
    if $\phi(\gamma) \ge 0$ then
      $\mathit{SignChange} =$ true
      break
    end if
  end for
  if $\mathit{SignChange}$ then
    return $\gamma$
  else
    return none
  end if
```

where

$$A^p = \{\, f^p(t) : a^p \le t \le b^p \,\} \quad \text{and} \quad B^q = \{\, g^q(u) : c^q \le u \le d^q \,\}. \tag{11}$$

To compute the distance from a point x to the set B, we compute the distance to each of the B^q and use the fact that

$$d(x, B) = \min_{1 \le q \le Q} d(x, B^q).$$

Similarly, since

$$h(A, B) = \max_{1 \le p \le P} h(A^p, B),$$

we could compute $h(A, B)$ using P calls to Procedure 4.2, but this means we effectively prune subintervals from each A^p separately. A more efficient approach is to prune subintervals from all the A^p based on a common value of *MaxDist*, as described in Procedure 5.1.

6. Examples

We present seven examples to illustrate the performance of Procedures 4.2 and 5.1 for computing the directed Hausdorff distance $h(A, B)$. In each case, we set the tolerance $\epsilon = (b - a)/M$ so that the accuracy is comparable that achieved by a semi-direct method,

$$h(A, B) \approx h(S, B) = \max_{1 \le i \le M} d(x, B);$$

cf. (4). We aim to reduce the computational cost from $O(M)$ to something like $O(\log M)$. Figure 2 shows the sets A and B for Examples 1–6, all of which are in 2D. Figure 5 shows Example 7, which is in 3D. The computations were performed using Matlab.

Procedure 5.1 *DirHdfDist*(A, B, ϵ)

```
Require: $A$ and $B$ are sets of parametric curves given by (10), (11).
Require: $\epsilon > 0$
  for $p = 1,\ P$ do
    $[\alpha_1^p, \beta_1^p] \leftarrow [a^p, b^p]$; $N^p \leftarrow 1$; $MaxSize^p \leftarrow b^p - a^p$;
    $Midpt_1^p \leftarrow f^p(\frac{1}{2}\alpha_1^p + \frac{1}{2}\beta_1^p)$ $Dist_1^p \leftarrow d(Midpt_1^p, B)$;
  end for
  $MaxSize \leftarrow \max_{1\le p\le P} MaxSize^p$; $MaxDist \leftarrow \max_{1\le p\le P} MaxDist^p$;
  repeat
    for $p = 1,\ P$ do
      $\mathcal{I}^p \leftarrow \emptyset$
      for $j = 1,\ N^p$ do
        $\mathcal{I}^p \leftarrow \mathcal{I}^p \cup Prune([\alpha_j^p, \beta_j^p], Midpt_j^p, Dist_j^p, MaxDist)$
      end for
      Redefine $\alpha_j^p$, $\beta_j^p$, $N^p$ so that $\mathcal{I}^p = \{\, [\alpha_j^p, \beta_j^p] : 1 \le j \le N^p \,\}$
      for $j = 1, N^p$ do
        $Midpt_j^p \leftarrow f^p(\frac{1}{2}\alpha_j^p + \frac{1}{2}\beta_j^p)$
        $Dist_j^p \leftarrow d(Midpt_j^p, B)$
        $MaxDist \leftarrow \max(Dist_j^p, MaxDist)$
      end for
      $MaxSize^p \leftarrow \max\{\, \beta_j^p - \alpha_j^p : j = 1, 2, \dots, N^p \,\}$
    end for
    $MaxSize \leftarrow \max_{1\le p\le P} MaxSize^p$
  until $MaxSize < \epsilon$
  return $MaxDist$
```

Figure 3 shows the running times for Examples 1–7 as a function of M. In each case, the running time is the average for 20 trials. The upper graph has a log scale on the horizontal axis, and for comparison we plot $0.05 \times \log M$. The lower graph has a log scale on both axes, and for comparison we plot $\sqrt{M}$.

Example 1. Let

$$\begin{aligned} f(t) &= (t, t+1) && \text{for } t \in [a, b] = [0, 1], \\ g(u) &= (u, u+2) && \text{for } u \in [c, d] = [-\tfrac{1}{2}, \tfrac{1}{2}], \end{aligned} \tag{12}$$

so that A and B, given by (2), are parallel line segments in $\mathbb{R}^2$. In this case, $d(x, B) = 1/\sqrt{2}$ for every point $x \in A$, so the directed Hausdorff distance is $h(A, B) = 1/\sqrt{2}$. This example actually illustrates the worst-case behaviour of our algorithm: because every point of A is a non-isolated, optimal point, the pruning technique is not able to cull any part of A, and the running time is $O(M)$; see Table 1. However, this longer running time is of no consequence because we obtain the value of $h(A, B)$ to machine precision even for a small value of M.

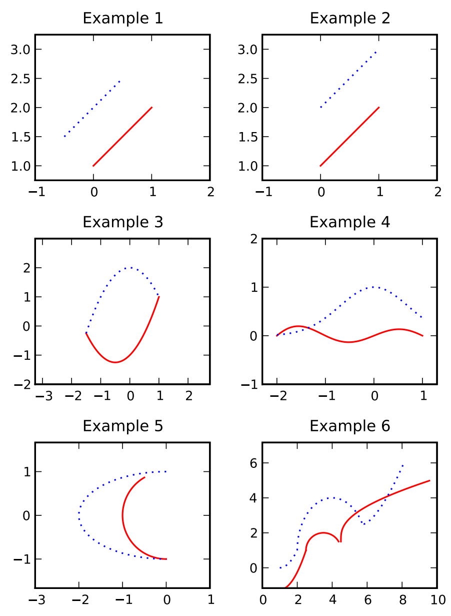


FIGURE 2. The sets A (solid line) and B (dotted line).

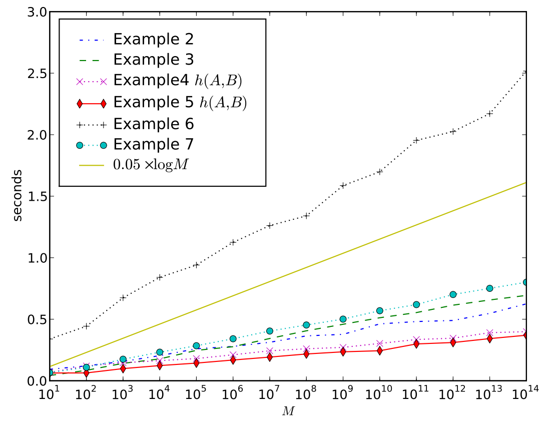
TABLE 1. Results for Example 1, where $[c,d] = [-\frac{1}{2}, \frac{1}{2}]$.

M	$\#\mathcal{I}$	*MaxSize*	*MaxDist*	seconds
10	16	6.25000e-002	0.707106781186548	0.2263
100	128	7.81250e-003	0.707106781186548	1.0379
1000	1024	9.76563e-004	0.707106781186548	7.8072
10000	16384	6.10352e-005	0.707106781186548	131.2414
100000	131072	7.62939e-006	0.707106781186548	1326.7364

Example 2. Define f and g as in (12), but change the parameter interval for g to $[c,d] = [0,1]$. Now, $h(A,B) = 1$ and the unique, isolated optimal point is $(0,1) \in A$. Table 2 shows the effectiveness of the pruning technique: at the final step of Procedure 4.2 the list $\mathcal{I}$ always contains only 2 subintervals, and we observe a dramatic reduction in the running time.

Example 3. With the choice

$$\begin{aligned} f(t) &= (t, t^2 + t - 1) \quad && \text{for } t \in [a,b] = [-1.5, 1], \\ g(u) &= (u, 2 - u^2) \quad && \text{for } u \in [c,d] = [-1.5, 1], \end{aligned}$$



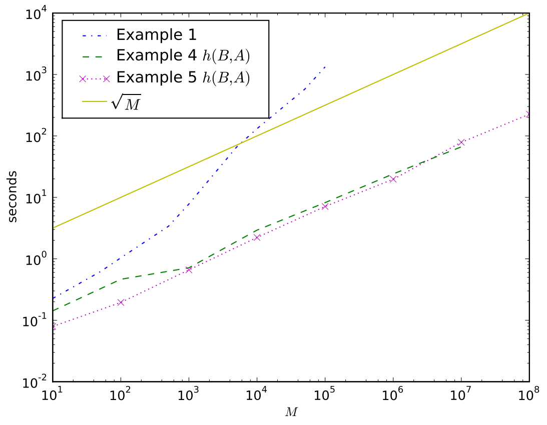


FIGURE 3. Runtimes for Examples 1–7.

the curves A and B are parabolas, and each optimal point is isolated so the runtime is $O(\log M)$.

Example 4. We define a polynomial of degree 11,

$$L(t) = t \prod_{\substack{k=-5 \\ k\neq 0}}^{5} \frac{t+k}{k},$$

and put

$$\begin{aligned} f(t) &= \big(t, \tfrac{2}{5}L(t)\big) \quad \text{for } t \in [a,b] = [-2,1], \\ g(u) &= \big(u, e^{-u^2}\big) \quad \text{for } u \in [c,d] = [-2,1]. \end{aligned}$$

TABLE 2. Results for Example 2, where $[c,d] = [0,1]$.

M	$\#\mathcal{I}$	*MaxSize*	*MaxDist*	seconds
10	2	6.250e-002	0.969253901204427	0.0932
10^2	2	7.813e-003	0.996101409284279	0.1209
10^3	2	9.766e-004	0.999511838017519	0.1570
10^4	2	6.104e-005	0.999969482887551	0.2041
10^6	2	9.537e-007	0.999999523162956	0.2788
10^8	2	7.451e-009	0.999999996274710	0.3647
10^{10}	2	5.821e-011	0.999999999970897	0.4620
10^{12}	2	9.095e-013	0.999999999999546	0.4900
10^{14}	2	7.105e-015	0.999999999999997	0.6252

TABLE 3. Results for Example 3.

M	$\#\mathcal{I}$	*MaxSize*	*MaxDist*	seconds
10	2	2.223e-001	1.80407474453161	0.0427
10^2	4	1.479e-002	1.84497126959031	0.0848
10^3	4	1.068e-003	1.84729390012603	0.1416
10^4	4	1.857e-004	1.84744139978487	0.1775
10^5	4	1.347e-005	1.84747017516372	0.2470
10^6	4	2.344e-006	1.84747203457295	0.2768
10^8	4	1.234e-008	1.84747242413561	0.4061
10^{10}	4	1.558e-010	1.84747242617187	0.5121
10^{12}	4	1.967e-012	1.84747242619758	0.6143
10^{14}	4	2.492e-014	1.84747242619790	0.6937

This time, in Table 4, we show results for computing both $h(A,B)$ and $h(B,A)$. The calculation of $h(A,B)$ is fast, because at each step $\mathcal{I}$ contains only 2 subintervals. However, the calculation of $h(B,A)$ is much slower since the number of subintervals grows like $\sqrt{M}$. Figure 4 shows that t^* is not a stationary point of $d\big(f(t),B\big)$, whereas u^* is a stationary point of $d\big(g(u),A\big)$. Notice however that we obtain 7 digits of accuracy with $M = 10^7$ in the case of $h(A,B)$, but only $M = 10^3$ in the case of $h(B,A)$. The corresponding CPU times are 0.2435 seconds and 0.7140 seconds, so the calculation of $h(B,A)$ is not as problematic as it may at first seem. For practical computations, we recommend modifying the stopping criterion in Procedure 4.2 so that we exit the repeat-loop if $\sharp\mathcal{I}$ grows beyond a few hundred.

Example 5. We take

$$\begin{aligned} f(t) &= (\cos t, \sin t) && \text{for } t \in [a,b] = [\tfrac{2}{3}\pi, \tfrac{3}{2}\pi], \\ g(u) &= (2\cos u, \sin u) && \text{for } u \in [c,d] = [\tfrac{1}{2}\pi, \tfrac{3}{2}\pi], \end{aligned}$$

TABLE 4. Results for Example 4.

$h(A,B)$

M	$\#\mathcal{I}$	*MaxSize*	*MaxDist*	seconds
10	4	1.801e-001	0.90624262085435	0.0676
10^2	2	1.552e-002	0.91884096011899	0.1202
10^3	2	1.857e-003	0.92111336842335	0.1461
10^4	2	2.224e-004	0.92138430952460	0.1635
10^5	2	2.665e-005	0.92141675342013	0.1803
10^6	2	1.105e-006	0.92142098638664	0.2110
10^7	2	1.324e-007	0.92142114759182	0.2435
10^8	2	1.587e-008	0.92142116690665	0.2598
10^{10}	2	2.278e-010	0.92142116949814	0.3034
10^{12}	2	1.132e-012	0.92142116953570	0.3460
10^{14}	2	1.621e-014	0.92142116953589	0.3990

$h(B,A)$

M	$\#\mathcal{I}$	*MaxSize*	*MaxDist*	seconds
10	4	1.591e-001	0.96637645204818	0.1422
10^2	14	1.960e-002	0.96811505246297	0.4644
10^3	38	2.446e-003	0.96811574434799	0.7140
10^4	144	1.528e-004	0.96811576879148	2.9292
10^5	406	1.911e-005	0.96811576962759	8.2115
10^6	1152	2.388e-006	0.96811576966203	24.0839
10^7	3248	2.985e-007	0.96811576966290	66.3821

so that A is an arc of the unit circle and B is the left half of the ellipse $x^2/2^2 + y^2 = 1$. For $h(A,B)$ we observe a fast, $O(\log M)$ running time, but for $h(B,A)$ the optimal point occurs at a stationary point for $u \mapsto d\,(g(u), A)$ so, as with Example 4, the running time is $O(\sqrt{M})$.

Example 6. We let A consist of three parametric curves given by

$$\begin{aligned} f^1(t) &= (t+1, t^2 - \tfrac{5}{4}) && \text{for } t \in [a^1, b^1] = [0, \tfrac{3}{2}], \\ f^2(t) &= (-\cos t + \tfrac{7}{2}, 1 + \sin t) && \text{for } t \in [a^2, b^2] = [0, \tfrac{5}{6}\pi], \\ f^3(t) &= (t^4 + \tfrac{9}{2}, e^t + \tfrac{1}{2}) && \text{for } t \in [a^3, b^3] = [0, \tfrac{3}{2}], \end{aligned}$$

and likewise define B by

$$\begin{aligned} g^1(u) &= (u+1, u^2) && \text{for } u \in [c^1, d^1] = [0, 1], \\ g^2(u) &= (4 - 2\cos u, 1 + 3\sin u) && \text{for } u \in [c^2, d^2] = [0, \tfrac{5}{6}\pi], \\ g^3(u) &= (u^2 + 5.8, u^3 + 2.5) && \text{for } u \in [a^3, b^3] = [0, \tfrac{3}{2}]. \end{aligned}$$

Table 6 shows that the algorithm of Section 5 converges rapidly in this case.

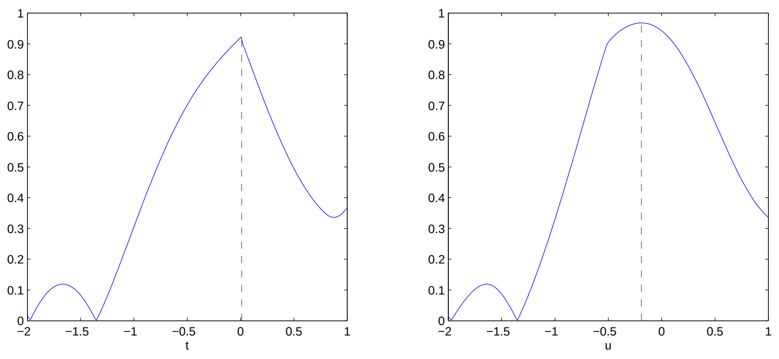


FIGURE 4. Shown on the left is $d\big(f(t), B\big)$, and on the right, $d\big(g(u), A\big)$ for Example 4.

Example 7. In our final example, A and B are curves in $\mathbb{R}^3$, parameterized by the functions

$$f(t) = (\cos t, \sin t, t) \qquad \text{for } t \in [a, b] = [\tfrac{3}{4}\pi, \tfrac{5}{4}\pi],$$
$$g(u) = (2\cos u, \sin u, 2u) \quad \text{for } u \in [c, d] = [\tfrac{1}{2}\pi, \tfrac{3}{2}\pi];$$

see Figure 5. The results in Table 7 show that we again have fast convergence.

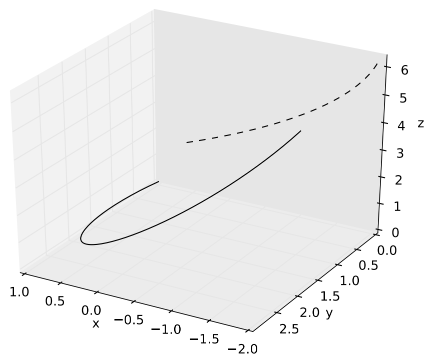


FIGURE 5. The sets A (solid line) and B (dashed line) for Example 7.

Table 5. Results for Example 5.

$h(A,B)$

M	$\#\mathcal{I}$	*MaxSize*	*MaxDist*	seconds
10	4	3.881e-002	0.811053984461385	0.0630
10^2	2	1.941e-002	0.813088499053888	0.0636
10^3	4	2.249e-003	0.816488847027757	0.0988
10^4	4	6.450e-005	0.816474874381915	0.1234
10^5	4	1.828e-005	0.816495972180698	0.1435
10^6	4	1.616e-006	0.816496452244338	0.1686
10^7	2	1.463e-007	0.816496519011455	0.1923
10^8	4	1.607e-008	0.816496578441697	0.2175
10^{10}	2	2.512e-010	0.816496580849110	0.2439
10^{12}	4	1.380e-012	0.816496580927693	0.3107
10^{14}	4	2.398e-014	0.816496580927726	0.3700

$h(B,A)$

M	$\#\mathcal{I}$	*MaxSize*	*MaxDist*	seconds
10	6	1.822e-001	2.30824867811854	0.0792
10^2	14	2.253e-002	2.30934241002361	0.1950
10^3	34	2.799e-003	2.30940105480629	0.6603
10^4	148	1.749e-004	2.30940107413654	2.2248
10^5	428	2.186e-005	2.30940107671031	7.1056
10^6	1208	2.733e-006	2.30940107675840	19.7423
10^7	4844	1.708e-007	2.30940107675850	78.6975
10^8	13688	2.135e-008	2.30940107675850	225.7033

Table 6. Results for Example 6.

M	$\#\mathcal{I}$	*MaxSize*	*MaxDist*	seconds
10	4	4.682e-002	1.8762335102915	0.3394
10^2	6	1.038e-002	1.8762335102915	0.4430
10^3	8	1.297e-003	1.8825200121058	0.6742
10^4	2	8.106e-005	1.8832818217065	0.8402
10^5	4	1.255e-005	1.8832818217065	0.9404
10^6	6	7.845e-007	1.8832854731356	1.1248
10^8	4	1.327e-008	1.8832864769426	1.3403
10^{10}	4	3.203e-011	1.8832864842719	1.6981
10^{12}	14	1.274e-012	1.8832864842760	2.0254
10^{14}	4	1.021e-014	1.8832864842770	2.5173

7. Conclusion

We have presented a new algorithm for computing the Hausdorff distance between two sets of parametric curves in $\mathbb{R}^n$. By employing a pruning technique, our method typically requires only $O(\log M)$ evaluations of the distance

TABLE 7. Results for Example 7.

M	$\#\mathcal{I}$	*MaxSize*	*MaxDist*	seconds
10	4	9.817e-002	1.58493363710854	0.0663
10^2	4	1.227e-002	1.61399049107328	0.1092
10^3	4	1.534e-003	1.61765296105915	0.1750
10^4	4	9.587e-005	1.61814396044311	0.2323
10^5	4	1.198e-005	1.61817260562654	0.2846
10^6	4	1.498e-006	1.61817618630199	0.3414
10^8	4	1.170e-008	1.61817669383126	0.4532
10^{10}	4	9.143e-011	1.61817669779633	0.5692
10^{12}	4	1.429e-012	1.61817669782706	0.7010
10^{14}	4	1.110e-014	1.61817669782755	0.8007

from a point to a curve, compared with $O(M)$ such evaluations to achieve comparable accuracy in the direct method. When the optimal point occurs at a stationary point of the distance function, the cost of our algorithm increases to $O(\sqrt{M})$, but faster convergence means that we require only a moderate value of M to attain acceptable accuracy, and the growth in the number of retained subintervals provides a convenient stopping criterion.

References

[1] H. Alt, P. Braß, M. Godau, C. Knauer, and C. Wenk, *Computing the Hausdorff distance of geometric patterns and shapes*, Discrete and computational geometry, 65–76, Algorithms Combin., 25, Springer, Berlin, 2003.

[2] E. Belogay, C. Cabrelli, U. Molter, and R. Shankwiler, *Calculating the Hausdorff distance between curves*, Inform. Process. Lett. **64** (1997), no. 1, 17–22.

[3] M. Bouts, *Comparing images using the Hausdorff distance*, unpublished manuscript, 2006.

[4] R. P. Brent, *Algorithms for Minimization without Derivatives*, Prentice–Hall, 1973.

[5] G. Rote, *Computing the minimum Hausdorff distance between two point sets on a line under translation*, Inform. Process. Lett. **38** (1991), no. 3, 123–127.

[6] L. Scharf, *Computing the Hausdorff distance between sets of curves*, Thesis, Freie Universität, Berlin, 2003.

Ik-Sung Kim
Division of Data Information
Korea Maritime University
Busan 606-791, Korea
E-mail address: ikim@hhu.ac.kr

William McLean
School of Mathematics and Statistics
The University of New South Wales
Sydney 2052, Australia
E-mail address: w.mclean@unsw.edu.au