LINEAR-TIME APPROXIMATE HAUSDORFF DISTANCE

OLIVER CHUBET, PARTH PARIKH, DONALD R. SHEEHY, AND SIDDHARTH SHETH

North Carolina State University

1. Introduction

The Hausdorff distance is a metric on compact subsets of a metric space. Let $(X, \mathbf{d})$ be a metric space and let A and B compact subsets of X. The distance from a point $x \in X$ to the set B is $\mathbf{d}(x, B) := \min_{b \in B} \mathbf{d}(x, b)$. The directed Hausdorff distance is $\mathbf{d}_h(A, B) := \max_{a \in A} \mathbf{d}(a, B)$, and the (undirected) Hausdorff distance is $\mathbf{d}_H(A, B) := \max\{\mathbf{d}_h(A, B), \mathbf{d}_h(B, A)\}$. This definition leads directly to a quadratic time algorithm for finite sets. If one only has access to distances (i.e., no ability to hash points), there is a corresponding quadratic-time lower bound.

It is not easy to get an asymptotic improvement on the naive Hausdorff distance algorithm without using an efficient data structure in higher dimensions. Alt et al. [1] give an $O(n \log n)$ time algorithm to compute exact Hausdorff distance in the plane using Voronoi diagrams. Many heuristics are used in practice to speed up the naive algorithm [8, 7, 2]. Another popular technique is to use a geometric tree data structure [9, 5].

We present an algorithm that computes a $(1 + \varepsilon)$-approximation to the Hausdorff distance between two sets of total size n in $\left(\frac{1+\varepsilon}{\varepsilon}\right)^{O(d)} n$ time after time after preprocessing the input sets individually into linear-size ball trees. The preprocessing takes $\left(\frac{1}{\varepsilon}\right)^{O(d)} n \log \Delta$ time per set.Throughout, we assume that distance computations take constant time. Although we explicitly consider the spread of the input in the analysis of our algorithms, there are several cases where we assume that the spread is at most $2^{O(n)}$.

2. Background

2.1. **Doubling Metrics.** Let $(X, \mathbf{d})$ be a metric space. A **metric ball** in X of radius r centered at c is defined as $\mathbf{ball}(c, r) := \{x \in X \mid \mathbf{d}(x, c) \leq r\}$. The **spread** Δ of $A \subseteq X$ is the ratio of the largest to smallest pairwise distance of points in A.

The set A is **λ-packed** if $d(a, b) \geq \lambda$ for any distinct $a, b \in A$. A collection of sets $\mathcal{C}$ **covers** A if $A \subseteq \bigcup_{S \in \mathcal{C}} S$. The **doubling constant** of X is the minimum number ρ such that any ball in X can be covered by at most ρ balls of half the radius. The **doubling dimension** is $\dim(X) := \log_2 \rho$. If $\dim(X)$ is bounded then X is a **doubling metric**. The following lemma [4] is true for packed and bounded sets.

Lemma 1 (Standard Packing Lemma)**.** *If X is a metric space with $\dim(X) = d$ and $Z \subset \mathbf{ball}(x, r)$ for some $x \in X$ is λ-packed then $|Z| \leq \left(\frac{4r}{\lambda}\right)^d$.*

2.2. **Greedy Permutations.** Let $P = (p_0, ..., p_{n-1})$ be an ordering of n points. The **i^{th}-prefix** of P is the set P_i containing points $p_0, \ldots, p_{i-1}$. We say P is a **greedy permutation** if $\mathbf{d}(p_i, P_i) = \mathbf{d}_H(P_i, P)$ for all $i > 0$. Let $\alpha > 1$. We say P is an α-approximate greedy permutation if $\mathbf{d}_H(P_i, P) \leq \alpha \mathbf{d}(p_i, P_i)$ for all $i > 0$. If $\mathbf{d}(p_i, q) \leq \alpha \mathbf{d}(p_i, P_i)$, then q is an **approximate nearest predecessor**. We denote the distance to the

(approximate) nearest predecessor of p_i by ε_{p_i}. Then there is a $\frac{1}{\alpha}\varepsilon_{p_i}$-packing of the points in the prefix P_i. The greedy permutation of a set can be computed in $O(n \log \Delta)$ time in low dimensions.

2.3. **Greedy Trees.** A balltree [6] is defined by recursively partitioning a metric space and representing the parts in a binary tree. Each node of the tree is a metric ball that covers the points in its subtree.

A **greedy tree** G is a balltree that uses the greedy permutation P to guide the partition. For each node $\mathbf{ball}(p, r)$, the center p is a point from P. The radius r is the maximum distance to a point of P in the ball. Any node $\mathbf{ball}(p, r)$ with $r > 0$ has two children, $\mathbf{ball}(p, r_L)$ and $\mathbf{ball}(c, r_R)$, where p is the (approximate) nearest predecessor of c.[1] The nodes with radius 0 are leaves; there is a node $\mathbf{ball}(p, 0)$ for each element of P. The greedy tree has $2n - 1$ nodes, where $|P| = n$, and the radii are non-increasing from parent to child. Given a greedy permutation and the approximate nearest predecessors, the corresponding greedy tree can be computed in $O(n \log \Delta)$ time ($O(n)$ time to build the tree and $O(n \log \Delta)$ time to compute radii).

3. Approximate Hausdorff Distance

In this section, we describe an algorithm that computes $\mathbf{d}_h(A, B)$, given sets A and B. As in prior work [3], we assume that the input is preprocessed into a tree structure; in our case, we use greedy trees. We assume that nodes of the greedy trees are listed in sorted order by radii. The algorithm proceeds by iterating over the nodes. Each node gets added as a vertex in the **neighbor graph** which maintains the following invariants:

- **Neighbor Invariant**: If $\mathbf{d}(a, B) = \mathbf{d}(a, b)$ then there is an edge between the nodes storing a and b.
- **Partition of Input**: Any point in A or B will be stored by some node in the neighbor graph.
- **Lower Bounds**: A node p in the neighbor graph stores a lower bound $l(p)$ to the distance $\mathbf{d}(p, B)$. This is called a **local lower bound**. The greatest of these is the **global lower bound**, denoted L.

The algorithm then proceeds, updating the neighbor graph and pruning edges that are too long to impact the neighbor invariant. It stops when it reaches a node whose radius is sufficiently small compared to the global lower bound. At that point the lower bound is a good approximation, and we return it. We update

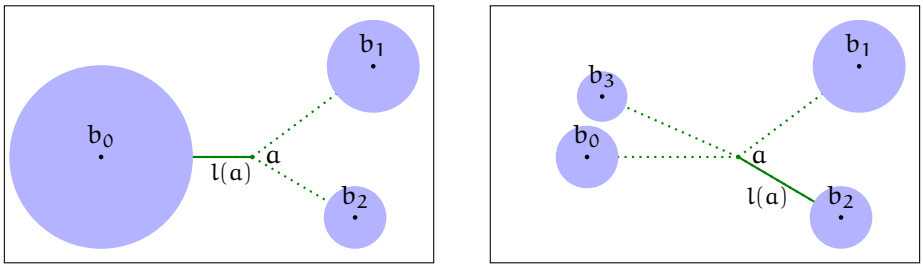

FIGURE 1. This figure depicts an update of $l(a)$ after replacing a node at b_0 with its children.

the local lower bound for a point p as shown in Figure 1 by computing,

$$l(p) = \min_{q \in N(p)} \{\mathbf{d}(p, q) - r_q\}.$$

We update L each time we update a local lower bound by comparing the two values (see Figure 2). We prune nodes that are too far to contain any nearest neighbors (see Section 3.1) and stop when L is a $(1 + \varepsilon)$-approximation of the exact distance (see Section 3.2). These improvements bound degrees in the neighbor graph by a constant and so each node can be processed in constant time.

[1]The (approximate) nearest predecessors need not be unique, however for the sake of construction we assume we have chosen one.

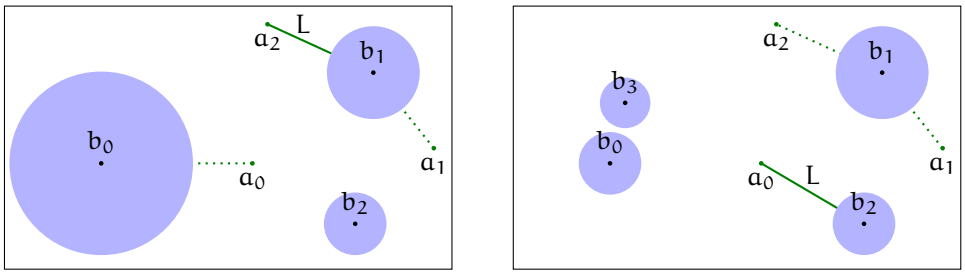

FIGURE 2. This figure depicts an update of L after replacing the node at b_0 with its children.

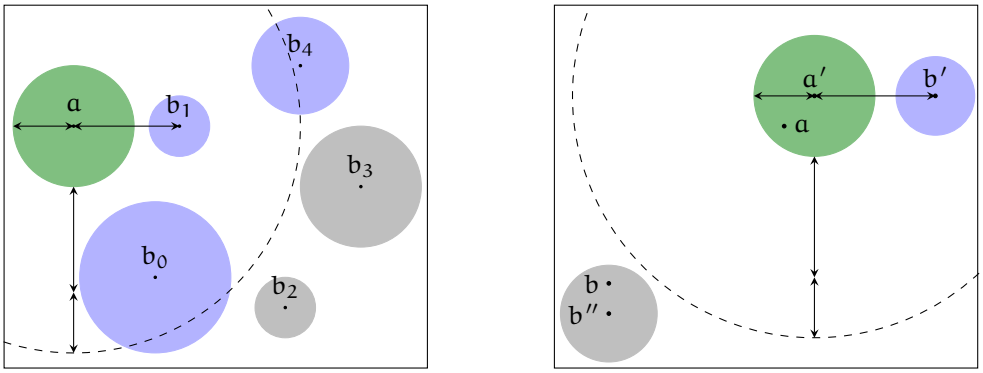

FIGURE 3. This figure shows the role of the pruning condition. On the left, the balls b_2 and b_3 are too far away to contain the nearest neighbor of a, so edges (a, b_2) and (a, b_3) will be pruned from the neighbor graph. The pruning condition respects the neighbor invariant and does not prune edge (a, b_4). In the right image let $a' \in A^r$. For any point $a \in B(a', r_{a'})$, by the triangle inequality, $d(a, B) \leq r_{a'} + \mathbf{d}(a', b')$. If b'' is a pruned center, then no point $b \in B(b'', r_{b''})$ can be the nearest neighbor of a, because $\mathbf{d}(a, b') \leq \mathbf{d}(a, b)$ for any such b.

Algorithm: HAUSDORFF
Input: Greedy trees G_A, G_B and approximation factor ε
Output: $(1+\varepsilon)$-approximation of $\mathbf{d}_h(A, B)$

(1) Initialize the neighbor graph N with a node for each root and an edge between them.
(2) Iterate over nodes **ball**(p, r) of $G_A \cup G_B$ in non-increasing order of radii while $r > (\frac{\varepsilon}{2})L$.
  (a) Let the children of **ball**(p, r) be called **ball**(p, r_L) and **ball**(c, r_R).
  (b) Add c to N, and add edges so that $N(c) = N(p)$.
  (c) Update the radii, $r_p = r_L$ and $r_c = r_R$.
  (d) If $p \in A$, prune $N(c)$ and $N(p)$, and update $l(c)$ and $l(p)$.
  (e) If $p \in B$, for each $a \in N(p)$, prune $N(a)$ and update $l(a)$.
(3) Return L.

To compute exact $\mathbf{d}_h(A, B)$, let $\varepsilon = 0$. However our running time guarantees do not hold in that case.

3.1. **Pruning Condition.** Consider an iteration of the main loop where the node **ball**(p, r) is being processed. Let A^r and B^r be the nodes in N from G_A and G_B respectively. We prune an edge $(a, b) \in N$ if $\mathbf{d}(a, b') + r_a < \mathbf{d}(a, b) - r_a - r_b$ for some $b' \in B^r$, as shown in Figure 3. The following lemma proves that the pruning condition does not trim any edge between nodes that contain nearest neighbors. A proof of correctness for the pruning condition is presented in Appendix **??**.

3.2. **Stopping Condition.** The running time of an iteration depends on the degree of a vertex in the neighbor graph. The key to achieving this bound is to stop the algorithm early so that the number of neighbors can be bounded by a packing argument. Let r be the radius of the current node, and L the global lower bound. Then, $L \leq \mathbf{d}_h(A, B) \leq L + 2r$. Moreover, if $r \leq (\frac{\varepsilon}{2})L$, then L is a $(1+\varepsilon)$-approximation to $\mathbf{d}_h(A, B)$ (see Appendix **??**).

3.3. **Analysis.**

Theorem 2. *Given two approximate greedy trees for sets A and B of total cardinality n,* HAUSDORFF *computes a $(1+\varepsilon)$-approximation of $\mathbf{d}_h(A, B)$ in $\left(\frac{1+\varepsilon}{\varepsilon}\right)^{O(d)} n$ time.*

Proof. In order to bound the degrees of the neighbor graph N, we first establish that the points associated with the neighbors of a vertex in N are packed. By construction, any node $p \in N$ is the center of the left- or right-child of a greedy tree node with radius at least r. Then by Lemma **??**, N is $\frac{(\alpha-1)r}{(2\alpha-1)\alpha}$-packed. Thus,

$$|N(a)| \leq \left(\frac{2\alpha(2\alpha-1)(L+2r)}{(\alpha-1)r}\right)^d \leq \left(\frac{8\alpha^2(1+\varepsilon)}{(\alpha-1)\varepsilon}\right)^d,$$

for all $r \geq (\frac{\varepsilon}{2})L$, by Lemma 1. Therefore, the number of edges incident to any given node in N is $\left(\frac{1+\varepsilon}{\varepsilon}\right)^{O(d)}$. So we spend $\left(\frac{1+\varepsilon}{\varepsilon}\right)^{O(d)}$ time for each iteration of the algorithm. This gives a running time of $\left(\frac{1+\varepsilon}{\varepsilon}\right)^{O(d)} n$. □

4. Conclusion

We have given a new algorithm for computing the Hausdorff distance and its relatives. After pre-processing the point sets, individual distance computations take only linear time.

References

[1] Helmut Alt, Bernd Behrends, and Johannes Blömer. Approximate matching of polygonal shapes. *Annals of Mathematics and Artificial Intelligence*, 13(3-4):251–265, September 1995.

[2] Yilin Chen, Fazhi He, Yiqi Wu, and Neng Hou. A local start search algorithm to compute exact Hausdorff Distance for arbitrary point sets. *Pattern Recognition*, 67:139–148, July 2017.

[3] Daniel P. Huttenlocher, Klara Kedem, and Jon M. Kleinberg. On dynamic voronoi diagrams and the minimum hausdorff distance for point sets under euclidean motion in the plane. In *Proceedings of the Eighth Annual Symposium on Computational Geometry*, SCG '92, pages 110–119, New York, NY, USA, 1992. Association for Computing Machinery.

[4] Robert Krauthgamer and James R. Lee. Navigating nets: Simple algorithms for proximity search. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 798–807, USA, 2004. Society for Industrial and Applied Mathematics.

[5] Sarana Nutanong, Edwin H. Jacox, and Hanan Samet. An incremental Hausdorff distance calculation algorithm. *Proceedings of the VLDB Endowment*, 4(8):506–517, May 2011.

[6] Stephen M. Omohundro. Five balltree construction algorithms. Technical Report 562, ICSI Berkeley, 1989.

[7] Jegoon Ryu and Sei-ichiro Kamata. An efficient computational algorithm for Hausdorff distance based on points-ruling-out and systematic random sampling. *Pattern Recognition*, 114:107857, June 2021.

[8] Abdel Aziz Taha and Allan Hanbury. An Efficient Algorithm for Calculating the Exact Hausdorff Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(11):2153–2163, November 2015.

[9] Dejun Zhang, Fazhi He, Soonhung Han, Lu Zou, Yiqi Wu, and Yilin Chen. An efficient approach to directly compute the exact Hausdorff distance for 3D point sets. *Integrated Computer-Aided Engineering*, 24(3):261–277, July 2017.