A simple $O(n \log n)$ algorithm for finding the maximum distance between two finite planar sets

Godfried T. Toussaint and Jim A. McAlear
School of Computer Science
McGill University
Montreal, Quebec, Canada

Published in *Pattern Recognition Letters*, October 1982, pp. 21-24.

Abstract

A simple $O(n \log n)$ algorithm is presented for computing the maximum Euclidean distance between two finite planar sets of n points. When the n points form the vertices of simple polygons, then the complexity reduces to $O(n)$.

1 Introduction

Let $S_1 = p_1, p_2, \ldots, p_n$ and $S_2 = q_1, q_2, \ldots, q_n$ be two planar sets of n points each and let $S = S_1 \cup S_2$. The sets need not have equal cardinality but such an assumption simplifies notation. A point p_i is given in terms of the Cartesian coordinates x_i and y_i. The maximum distance between S_1 and S_2, denoted by $d_{\max}(S_1, S_2)$, is defined as

$$d_{\max}(S_1, S_2) = \max_{i,j}\{d(p_i, q_j)\}, \quad i, j = 1, 2, \ldots, n,$$

where $d(p_i, q_j)$ is the Euclidean distance between p_i and q_j.

The computation of distances between sets arises frequently in pattern recognition problems [3], where clustering is a prime example. In agglomerative (bottom-up) clustering procedures one starts with a set of N clusters each containing only one of N points to be clustered. The two most *similar* clusters are then merged to form $N-1$ new clusters. This procedure is continued by successively merging clusters. What distinguishes many clustering algorithms is the measure of *similarity* used to determine which pair of clusters gets merged at a given step. When $d_{\max}$ is used the resulting algorithm is known as the *furthest neighbor clustering algorithm* [3]. Another frequently used distance between sets is given by:

$$d_{\text{mean}}(S_1, S_2) = d(m_1, m_2),$$

where

$$m_1 = \frac{1}{n}\sum_{i=1}^{n} p_i \text{ and } m_2 = \frac{1}{n}\sum_{i=1}^{n} q_i.$$

Clearly $d_{\text{mean}}(S_1, S_2)$ can always be computed in $O(n)$ time. Duda and Hart [3] point out that d_{mean} is computationally more attractive than $d_{\max}$ by claiming that $d_{\max}$ requires the computation of all n^2 distances, resulting in an $O(n^2)$ algorithm.

In [2] it was shown that $d_{\max}(S_1, S_2)$ can be computed in $O(n \log n)$ time in the worst case. The algorithm in [2] is quite complicated and is based on first partitioning each set S_i, $i = l, 2$ into nine subsets S_{ij}, $j = 1, \ldots, 9$ and converting the $d_{\max}(S_1, S_2)$ problem into 81 diameter problems on the sets of the form $(S_{1u} \cup S_{2v})$. In this note we present a very simple algorithm based on searching a generalization of the notion of *antipodal pairs* of points. We assume that the reader is familiar with the $O(n)$ diameter algorithm of Shamos [10].

2 Preliminary Results

Let $CH(S_i)$, $i = 1, 2$ denote the set of points of S_i which are vertices of the convex hull of S_i. Denote the diameter of S_i by $\text{diam}(S_i)$, i.e.,

$$\text{diam}(S_1) = \max_{i,j}\{d(p_i, p_j)\}, \quad i, j = 1, 2. \ldots, n.$$

We now review and establish some results which will form the theoretical foundation for the algorithm of Section 3. The following results have been proved in [2].

Lemma 2.1 $d_{\max}(S_1, S_2) = d_{\max}(CH(S_1), CH(S_2))$.

Lemma 2.2 $d_{\max}(S_1, S_2) \leq \text{diam}(S)$.

It should be noted here that the claim made by several authors (such as Duda and Hart [3] and Johnson [6]) that $d_{\max}(S_1, S_2) = \text{diam}(S)$ is not always true. Were this so the diameter of $S_1 \cup S_2$ could be found in $O(n \log n)$ time with either the convex hull approach of Shamos [10, 11] of the furthest-point Voronoi diagram methods of [13] and [7].

We now define some new terms and establish some additional results. Let $LS_i(a)$ denote the directed line of support of set S_i through point $a \in S_i$ such that no points of S_i lie to the left of $LS_i(a)$.

Definition 2.1 *Given two sets of points* S_1 *and* S_2 *an* antipodal pair *of points between the sets is a pair* $p_i \in S_1$, $q_j \in S_2$ *such that* S_1 *and* S_2 *admit parallel directed lines of support* $LS_1(p_i)$ *and* $LS_2(q_j)$ *which have opposite directions.*

Consider two sets S_1 and S_2 whose convex hulls are illustrated in Fig. 1. A specified direction, such as the x-axis, determines up to four lines of support, two for each set. In Fig. 1, (a, b) and (c, d) are two *antipodal pairs* between the sets according to the above definition. Note that (a, d) is not an antipodal pair even though a and d lie on different sets and admit parallel lines of support.

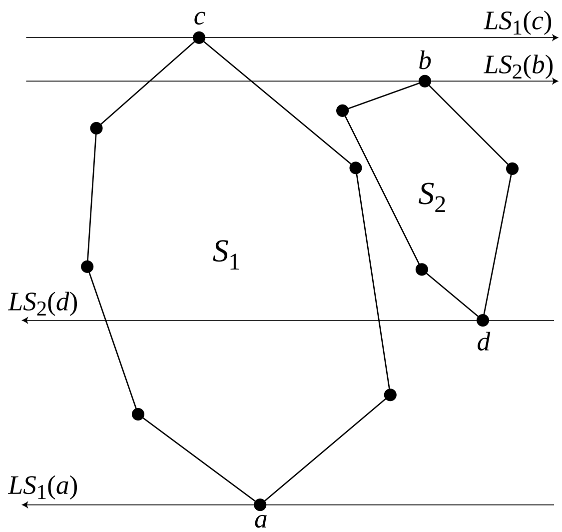

Figure 1: Illustrating antipodal pairs between sets.

Theorem 2.3 *The pair of points $a \in S_1$ and $b \in S_2$ that realize $d_{\max}(S_1, S_2)$ constitutes an antipodal pair between S_1 and S_2.*

Proof: Let $a \in S_1$ and $b \in S_2$ determine $d_{\max}(S_1, S_2)$ and refer to Fig. 2. It follows that b is the furthest point among S_2 from a. Therefore S_2 must lie in the disk determined by the circle centered at a with radius equal to $d(a, b)$. It follows that we can construct a directed line of support $LS_2(b)$ which is tangent to the circle at b. We can similarly construct $LS_1(a)$. Since $LS_1(a)$ and $LS_2(b)$ are each orthogonal to line segment $[a, b]$ they must be parallel, thus proving the theorem. ∎

Note: Although S_1 and S_2 are linearly separable in Fig. 1 the proof holds for non-linearly separable situations as well.

3 The Algorithm

Algorithm MAXDIST

Input: Two sets of points on the plane, S_1, S_2.
Output: $d_{\max}(S_1, S_2)$.
Step 1: Compute $CH(S_1)$ and $CH(S_2)$.
Step 2: Determine all *antipodal pairs* between $CH(S_1)$ and $CH(S_2)$.
Step 3: Exit with the largest distance encountered in step 2.

Theorem 3.1 *Algorithm MAXDIST computes $d_{\max}(S_1, S_2)$ in $O(n \log n)$ time.*

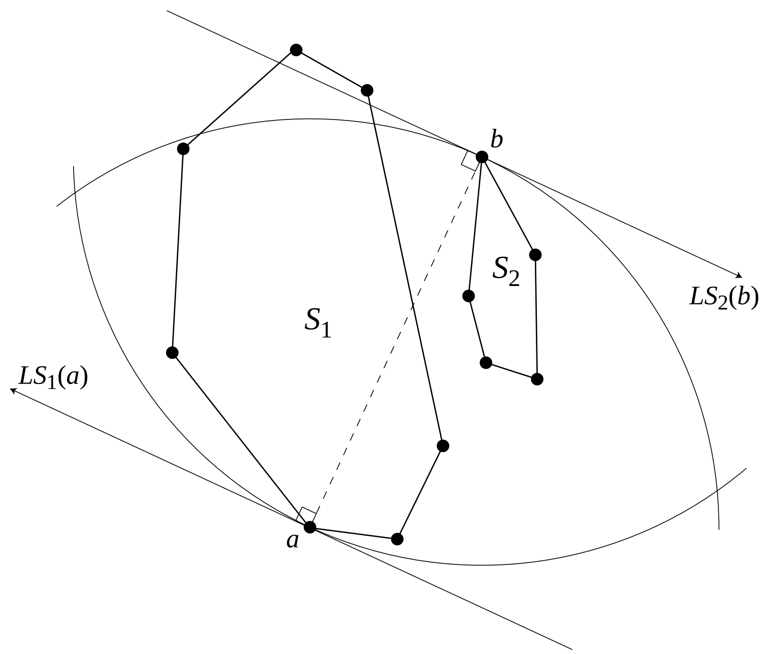

Figure 2: Illustrating the proof of Theorem 2.3.

Proof: The correctness of the algorithm follows from Lemma 2.1 and Theorem 2.3. We turn thus to the complexity. Step 1 can be done in $O(n \log n)$ time with a variety of algorithm [12, 5]. For step 2 we can use the "rotating caliper" algorithm of Shamos [10, 11]. This algorithm generates in $O(n)$ time the $O(n)$ *antipodal pairs* of a convex polygon. In algorithm MAXDIST we can use two "rotating calipers" one on each convex hull determined in step 1. Initially we pick an arbitrary direction and in $O(n)$ time we can determine the starting *antipodal pairs* between the sets. Both 'calipers' are now 'rotated' until one of the four support lines advances to a new vertex. This new vertex determines a new *antipodal pair* between the sets. This procedure is continued until the starting pair is reached again. The algorithm is essentially the same as Shamos' except for the fact that we have four, rather than two, lines of support to contend with. Thus it also generates all the *antipodal pairs* in $O(n)$ time. Hence, the complexity of MAXDIST is dominated by step 1. ■

Corollary 3.2 *Given two simple polygons P_1 and P_2, MAXDIST computes $d_{\max}(P_1, P_2)$ in $O(n)$ time.*

Proof: This result follows from the fact that the convex hull of a simple polygon can be computed in $O(n)$ time [9, 8, 4]. ■

4 Concluding Remarks

We have shown that $d_{\max}(S_1, S_2)$ can be computed with a much simpler $O(n \log n)$ algorithm than that proposed in [2]. The algorithm runs in $O(n)$ expected time under the same conditions considered in [2], to which the reader is referred for details. The optimality of the algorithm is still an open problem as is the d-dimensional case. For the analogous minimum-distance-between-sets problem Avis [1] proved an $O(n \log n)$ lower bound and Toussaint and Bhattacharya [14] exhibit several algorithms that achieve this complexity. Since no such lower bounds have been established for the $d_{\max}$ problem we cannot say that algorithm MAXDIST is optimal.

References

[1] D. Avis. Lower bounds for geometric problems. In *Proc. Allerton Conference*, pages 35–40, Urbana, IL, October 1980.

[2] B.K. Bhattacharya and G.T. Toussaint. Efficient algorithms for computing the maximum distance between two finite planar sets. *Journal of Algorithms*, 4:121–136, 1983.

[3] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis.* John Wiley, N.Y., 1973. p. 235.

[4] H. ElGindy, D. Avis, and G.T. Toussaint. Applications of a two-dimensional hidden-line algorithm to other geometric problems. Technical Report SOCS-81.13, School of Computer Science, McGill University, April 1981.

[5] R. Graham. An efficient algorithm for determining the convex hull of a planar set. *Information Processing Letters*, 1:132–133, 1972.

[6] S.C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32:241–254, September 1967.

[7] D.T. Lee. Farthest neighbor voronoi diagrams and applications. Technical Report 80-11-FC-04, Northwestern University, November 1980.

[8] D.T. Lee. On finding the convex hull of a simple polygon. Technical Report 80-03-FC-Ol, Dept. Elec. Engr. and Computer Science, Northwestern University, 1980.

[9] D. McCallum and D. Avis. A linear algorithm for finding the convex hull of a simple polygon. *Information Processing Letters*, 9(5):201–206, December 1979.

[10] M.I. Shamos. Geometric complexity. In *Proc. 7th ACM Symposium on the Theory of Computing*, pages 224–233, May 1975.

[11] M.I. Shamos. *Computational geometry.* PhD thesis, Yale University, May 1978.

[12] G.T. Toussaint. Pattern recognition and geometrical complexity. In *Proc. Fifth International Conf. on Pattern Recognition*, pages 1324–1347, Miami Beach, December 1980.

[13] G.T. Toussaint and B.K. Bhattacharya. On geometric algorithms that use the furthest-point voronoi diagram. Technical Report SOCS-81.3, School of Computer Science, McGill University, January 1981.

[14] G.T. Toussaint and B.K. Bhattacharya. Optimal algorithms for computing the minimum distance between two finite planar sets. In *Proc. Fifth International Congress of Cybernetics and Systems*, Mexico City, August 1981.