Illinois, pp. 35-40, October 1980.

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

[23] J. L. Bentley, M. G. Faust, and F. P. Preparata, "Approximation algorithms for convex hulls," *Comm*. *ACM* **25** (January 1982), 64-68.

[24] S. G. Akl and G. T. Toussaint, "A fast algorithm for the planar convex hull problem," internal manuscript, School of Computer Science, McGill University, 1978.

[25] B. K. Bhattacharya and G. T. Toussaint. "A time-and-storage efficient implementation of an optimal planar convex hull algorithm," Technical Report No. SOCS 81.40, School of Computer Science, McGill University, December 1981.

[4] M. I. Shamos, *Computational Geometry*, Ph.D. thesis, Yale University, 1978.

[5] G. T. Toussaint and B. K. Bhattacharya, "On geometric algorithms that use the furthest-point Voronoi diagram," School of Computer Science, McGill University, Tech. Rept. No. 81.3, January 1981.

[6] G. T. Toussaint, "Pattern recognition and geometrical complexity," *Proc. Fifth International Conf. on Pattern Recognition*, Miami Beach, pp. 1324-1347, December 1980.

[7] A. Renyi and R. Sulanke, "Uber die konvexe Hülle von n zufällig gewählten Punkten, I", *Z. Wahrsch. Verw. Gebiete* **2** (1963), 75-84.

[8] A. Renyi and R. Sulanke, "Uber die konvexe Hülle von n zufällig gewählten Punkten, II", *Z. Wahrsch. Verw. Gebiete* **3** (1964), 138-147.

[9] A. Renyi and R. Sulanke, "Zufällige konvexe Polygone in einem Ringgebeit", *Z. Wahrsch. Verw. Gebiete* **9** (1968), 146-157.

[10] L. Devroye, "Recent results on the average time behavior of some algorithms in computational geometry," *Computer Science and Statistics: Proc. 13th Symposium on the Interface*, Springer-Verlag, New York, pp. 76-82, 1981.

[11] J. L. Bentley and M. I. Shamos, "Divide and conquer for linear expected time," *Inform. Process. Lett.* **7** (1978), 87-91.

[12] L. Devroye and G. T. Toussaint, "A note on linear expected time algorithms for finding convex hulls," *Computing*, (1981), **26**, 361-366.

[13] G. T. Toussaint, "Computational geometric problems in pattern recognition," *Pattern Recognition Theory and Applications*, (J. Kittler, Ed.), NATO ASI, Oxford University, pp. 73-91, April 1981.

[14] D. McCallum and D. Avis, "A linear algorithm for finding the convex hull of a simple polygon," *Inform. Process. Lett.* **9**, no.5 (December 1979), 201-206.

[15] D. T. Lee, "On Finding the Convex Hull of a Simple Polygon," Tech. Rept. No. 80-03-FC-01, Dept. Elec. Engr., and Computer Science, Northwestern University, 1980.

[16] H. ElGindy, D. Avis, and G. T. Toussaint, "Applications of a two-dimensional hidden-line algorithm to other geometric problems," Tech. Rept. No. SOCS-81.13, School of Computer Science, McGill University, April 1981.

[17] J. L. Bentley, *et al*., "On the average number of maxima in a set of vectors and applications," *J. ACM* **25** (1978), 536-543.

[18] L. Devroye, "A note on finding convex hulls via maximal vectors," *Inform. Process. Lett.* **11** (August 1980), 53-56.

[19] S. G. Akl and G. T. Toussaint, "Efficient convex hull algorithms for pattern recognition applications," *Proc. Fourth Internat. Joint Conf. on Pattern Recognition*, Kyoto, Japan, pp. 483-487, November 1978.

[20] R. Graham, "An efficient algorithm for determining the convex hull of a planar set," *Inform., Process. Lett.* **1** (1972), 132-133.

[21] D. Avis, "Lower bounds for geometric problems," *Proc. Allerton Conference*, Urbana,

It should be noted that if metrics other than the euclidean are used, then the O(n log n) complexity can be reduced for the set-of-points problem. In [13] it is shown that with the L_1 and L_∞ metrics, the maximum distance between two arbitrary sets of points can be computed in O(n) worst-case time in two dimensions. Furthermore, in d dimensions the L_∞ maximum distance can be computed in O(nd) worst-case time.

Finally, we note that for the euclidean metric we can obtain O(n) expected running time for any fixed dimensions d with a slight modification of the algorithm MAXDIST-1. Lemma 2.1 generalizes to higher dimensions. Furthermore, the 2^d sets of maximal vectors [17] of S_i are a superset of the vertices of the convex hull of S_i. Thus in the modified versions of MAXDIST-1 we first find the maximal vector sets S_{mi} of S_i, $i = 1, 2$ and then we use BRUTE-FORCE to compute $d_{max}(S_{m1}, S_{m2})$. Bentley et al. [17] have shown that the maximal vectors can be found in O(n) expected time whenever the underlying density f can be written as a d-fold product of densities:

$$f(x_1,..., x_d) = \prod_1^d f_i(x_i).$$

These are very general conditions which are satisfied for example for the normal density. Furthermore, Devroye [18] has shown that, under the above condition, if an $O(n^p)$ ($p \geq 1$) algorithm is used on the resulting set of maximal vectors the algorithm has average complexity O(n). Since computing the maximum distance by BRUTE-FORCE is an $O(n^2)$ operation, $p = 2$ in Devroye's theorem and the result follows.

One open problem in this area is to find algorithms for computing d_{max} in high dimensions in sub-quadratic worst-case time. Another is to establish a super-linear lower bound on the complexity of the problem. For the analogous *minimum* distance between sets problem, Avis [21] shows that Ω(n log n) is a lower bound, and Toussaint and Bhattacharya [22] exhibit algorithms that achieve this complexity. Since it is not known whether Ω(n log n) is a lower bound for the d_{max} problem, it is an open question whether algorithm MAXDIST-2 is optimal.

7. Acknowledgments

The authors are grateful to Luc Devroye for helpful discussions on the expected running time analysis of the algorithms in this paper. They also thank an anonymous reviewer for many helpful suggestions in revising the paper, and for pointing out that the ε-approximate diameter algorithm of Bentley, Faust, and Preparata [23] could be used to obtain a similar result for the maximum distance problem.

8. References

[1] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley, New York, 1973, p. 235.

[2] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika* **32**, September 1967, pp. 241-254.

[3] M. I. Shamos, "Geometric complexity," *Proc. 7th ACM Symposium on the Theory of Computing*, pp. 224-233, May 1975.

It is obvious from the analysis of MAXDIST-2 that if we replace the *exact* O(n log n) convex hull computations in Step 1 with *approximate* O(n) convex hull algorithms, we will have met our goal. Recently, Bentley, Faust and Preperata [23] proposed an algorithm that computes an ε-approximate convex hull of n points on the plane in $O(n + 1/\varepsilon)$ worst-case time. As pointed out in [23], this algorithm yields an ε-approximate diameter algorithm that runs in $O(n + 1/\varepsilon)$ time. Clearly we can obtain an approximate d_{max} algorithm by using the algorithm of Bentley et al. [23] in Step 1 of the MAXDIST-2. However, all the diameter computations in Step 4 are being made on *approximate* convex polygons which have themselves been obtained in Step 3 from the *approximate* hulls of Step 1, thus making the error-analysis somewhat involved. We can make sure that the error-analysis of [23] carries over exactly by simply doing away with Step 1 altogether, and by modifying Step 3 to compute the convex hulls of the unions of subsets. More precisely, we obtain the following algorithm.

Algorithm APPROXDIST

begin

Step 1: Partition S_i, $i = 1, 2$ into nine subsets S_{ij}, $j = 1, 2,..., 9$ as described in Lemma 2.6

Step 2: Find the ε-approximate convex hull of the union of each pair of subsets S_{1i} and S_{2j} for $i,j = 1, 2,..., 9$ using the algorithm of Bentley et al. [23].

Step 3: Find the diameter of each convex hull determined in Step 2 as in algorithm MAXDIST-2

Step 4: Find the maximum diameter computed in Step 3 and output it as $d_{max}(S_1, S_2)$.

end

It is clear from the above algorithm that when the algorithm exits with the final value of $d_{max}(S_1, S_2)$, say d*, then d* is the true diameter of a convex polygon which is itself the approximate convex hull of a set of points $S_{1i} \cup S_{2j}$ for some i and j. Since d* is the ε-approximate diameter obtained from the algorithm of Bentley et al. [23], and since the complexity of APPROXDIST is obtained by Step 2, it follows that algorithm APPROXDIST computes the ε-approximate maximum distance between two sets in $O(n + 1/\varepsilon)$ worst-case time.

6. Concluding Remarks

We have shown in this paper that given two planar point sets, S_1 and S_2, with a total of n points, the maximum euclidean distance between S_1 and S_2 can be computed in O(n log n) time in the worst case. We have also shown that if S_1 and S_2 are simple polygons, then the maximum euclidean distance between S_1 and S_2 can be computed in O(n) worst-case time.

TABLE 1

Average Run Times (msec) for Computing the Maximum Distance between Two Sets, Each Containing n Points Uniformly Distributed in Adjacent Non-overlapping Unit Squares[a]

Algorithm	n = 10(125)		n = 100(50)		n = 1000(15)		5000(10)	
	Time	Std. dev.	Time	Std. dev.	Time	Std. dev.	Time	Std. dev.
BRUTE-FORCE	0.0389	0.0157	25.9	0.0452	2570	4.69	---	---
MAXDIST-1	1.41	0.0969	5.73	0.406	39.2	2.01	178	4.89
MAXDIST-2	2.22	0.348	7.13	0.437	40.7	1.97	180	4.81

[a]In all tables, the numbers in parentheses indicate the number of times the experiment was repeated.

TABLE 2

Average Run Times (msec) for Computing the Maximum Distance between Two Sets, Each Set Uniformly Distributed in the Same Unit Square.

Algorithm	n = 10(125)		n = 100(50)		n = 1000(15)		5000(10)	
	Time	Std. dev.	Time	Std. dev.	Time	Std. dev.	Time	Std. dev.
MAXDIST-1	1.41	0.098	5.73	0.397	39.2	2.01	178	4.77
MAXDIST-2	2.01	0.201	7.51	0.625	41.8	2.05	182	4.85

TABLE 3

Average Run Times (msec) for Computing the Maximum Distance in a Worst-Case Situation, Each Set Containing n Points Uniformly Distributed on a Circle[b]

Algorithm	n = 10(125)		n = 100(50)		n = 1000(15)		5000(10)	
	Time	Std. dev.	Time	Std. dev.	Time	Std. dev.	Time	Std. dev.
MAXDIST-2	2.83	0.287	51.1	0.286	575	17.7	2914	85.1

[b]Both circles have unit radius, one is centered at (-1, 1), while the other is centered at (-1, 1.5).

as linearly unseparable sets, and the third to observe the algorithms' worst-case behavior. For simplicity, the number of points in each set was kept equal to n. The experiments were performed with n = 10, 100, 1000 and 5000. Each experiment was repeated a number of times. The average running times along with their standard deviations are given in Tables 1, 2, and 3. These times were obtained for programs compiled with a FORTRAN G1 compiler (with source optimization level 0), and executed on an AMDAHL V-7 computer at McGill University's Computing Center.

In the first set of experiments, points of both sets S_1 and S_2 were generated such that each set was distributed uniformly in adjacent side-to-side non-overlapping unit squares. The results are shown in Table 1. It is clear from the table that for very small n, (n = 10), the BRUTE-FORCE method is the fastest. For medium sized data sets (n = 100, 1000), MAXDIST-1 is the fastest, and for large sets, MAXDIST-1 and MAXDIST-2 are equally efficient. Comparing the MAXDIST algorithms to BRUTE-FORCE, we see that for n = 100 they run four times faster than BRUTE-FORCE, and for n = 1000, sixty times faster.

In the second set of experiments, both sets S_1 and S_2 were generated such that each set was distributed uniformly in the same unit square. These experiments were performed in order to determine whether linear separability between S_1 and S_2 had any appreciable effects on the algorithms' running time. The results shown in Table 2 indicate that the differences are insignificant. No times are given to for the BRUTE-FORCE algorithm, since in this case linear separability in no way affects the algorithm.

In the third set of experiments, both sets S_1 and S_2 were generated such that each set was distributed uniformly on intersecting circles. Thus $N_1 = N_2 = n$. The results are shown in Table 3. In this situation, the MAXDIST algorithms are at their worst. MAXDIST-1 is clearly always worse than BRUTE-FORCE, since in addition to BRUTE-FORCE it computes convex hulls in vain. Thus no times are shown for MAXDIST-1. For MAXDIST-2, we notice that BRUTE-FORCE is faster for $n \leq 100$. However, by the time n =1000, MAXDIST-2 runs four times faster than BRUTE-FORCE.

The above results clearly indicate the desirability of MAXDIST-2 in terms of running time. However, to more fully appreciate the algorithms, a practitioner may be interested in the lengths of the codes. The BRUTE-FORCE algorithm requires about 35 lines of source code. The MAXDIST algorithms depend on a convex hull program. The convex hull code, CONHUL, used in these experiments [24] requires about 390 lines, resulting in a total of 425 and 580 lines, respectively, for MAXDIST-1 and MAXDIST-2. However, these programs can be reduced by approximately 180 lines by using the more recent convex hull program, HULL, [25] which runs only marginally slower than CONHUL.

5. Approximate Algorithms

In some applications, obtaining an approximate solution quickly is more important than obtaining an exact answer at the cost of additional computation. In this section, we point out that an approximation to the maximum distance between sets can be computed in worst-case time *linear* in the number of input points, and that this approximation can be made as close to the true distance as desired.

Worst-Case Analysis

Step 1 can be computed in O(n log n) time [6]. Step 2 is clearly an O(n) operation. In Step 3 we need to find the convex hull of pairs of possibly overlapping convex polygons. Shamos [4] and Toussaint [13] exhibit simple O(n) algorithms for computing the convex hull of two convex polygons with a total of n vertices. Since the total number of vertices in all the convex polygons in $CH(S_i)$, $i = 1, 2$ is $N_i \leq n$, Step 3 can be done in O(n) time using the algorithms in [4] or [13]. The diameter of a convex n-gon can also be found in O(n) time using the "antipodal-pair" algorithm of Shamos [3]. Since the total number of vertices of all the convex polygons computed in Step 4 is O(n), it follows that Step 4 can be computed in linear time. Note that although Steps 3 and 4 involve 81 diameter computations, the "big-oh's" are not "hiding" constant factors of 81, as might appear at first glance. It is easy to see that the factor is in fact only 18. Finally, Step 5 can be done in time O(1), since it involves finding the maximum of at most 81 diameters. Thus, the entire algorithm is dominated by Step 1 and runs in O(n log n) time.

Expected Analysis

Since all steps other than Step 1 run in linear or sub-linear time, we need only be concerned with finding the convex hull in O(n) expected time. Again, many algorithms that run in linear expected time for some distributions are given in [12]. The use of any of these CH algorithms will yield a similar complexity for the MAXDIST-2 algorithm.

The Maximum Distance between Polygons

It is known that the convex hull of a simple n-vertex polygon can be computed in O(n) time [14-16]. Thus if S_1 and S_2 are simple polygons, Step 1 of algorithm MAXDIST-2 runs in O(n) time and we have the following theorem.

Theorem 3.3: The maximum distance between two simple polygons with a total of n vertices can be computed in O(n) time in the worst case.

Note that if in addition S_1 and S_2 are convex, then we can dispense with Step 1 altogether.

4. Experimental Results

In the implementation of both algorithms, the convex hulls in Step 1 were computed using the algorithm of Akl and Toussaint [19]. This algorithm is a modification of Graham's algorithm [20], which first discards most points from consideration and subsequently sorts the remaining points along the x-coordinate. It runs in O(n log n) time in the worst-case and O(n) expected time under the conditions described in [12]. Step 3 in MAXDIST-2 was implemented using the algorithm of Toussaint [13], which runs in time linear with the total number of vertices of the two convex polygons being merged. Finally, the computation of diameters in Step 4 was done using Shamos' "antipodal-pair" algorithm [13].

The BRUTE-FORCE method, MAXDIST-1, and MAXDIST-2 were compared experimentally by performing a Monte-Carlo simulation. Three sets of experiments were performed: the first two to compare the expected performance of the algorithms operating on linearly separable as well

where C(CH) is the complexity of finding the convex hulls in Step 1, and the k_i are positive constants. Taking the expected value of Eq. (9) yields

$$E\{C\} = E\{C(CH)\} + k_1E\{N_1\} + k_2E\{N_2\} + k_3E\{N_1N_2\} + k_4$$

$$\leq E\{C(CH)\} + k_1E\{N_1\} + k_2E\{N_2\} + k_3E\{N_1^2\} + k_3E\{N_2^2\} + k_4. \quad (10)$$

Devroye [10] has shown for the above conditions that $E\{N_i^p\} = O((\sqrt{n})^p)$ for any $p \geq 1$, $i = 1, 2$. Thus with $p = 2$ in Devroye's theorem, (10) becomes

$$E\{C\} \leq E\{C(CH)\} + O(n). \quad (11)$$

Now $E\{N_i\} \leq \sqrt{n}$ for many distributions. A list of distributions satisfying this condition appears in [11]. For example, when the points in S_i are uniformly distributed in a convex polygon of k sides, where k is fixed, then $E\{N_i\} = O(\log n)$ [7-9], where it is also established that for normal distributions $E\{N_i\} = O(\sqrt{\log n})$; see also [10]. Many convex hull algorithms, such as that of Bentley and Shamos [11], run in O(n) expected time under these conditions. Therefore $E\{C\} = O(n)$. For a survey of linear expected time CH algorithms, the reader is referred to [12].

Algorithm MAXDIST-2

begin

Step 1: Compute $CH(S_1)$ and $CH(S_2)$.

Step 2: Partition $CH(S_i)$ $i = 1, 2$ into nine subsets $CH(S_{ij})$, $j = 1, 2,..., 9$ as described in the previous section.

Step 3: Find the convex hull of the union of each pair of subsets

$$CH[CH(S_{1i}) \cup CH(S_{2j})]$$

for $i, j = 1, 2,..., 9$.

Step 4: Find the diameter of each convex polygon determined in Step3.

Step 5: Find the maximum diameter computed in Step 4.

$$\max_{i,\,j} \{\text{diam}[CH(S_{1i}) \cup CH(S_{2j})]\},$$

and output it as $d_{max}(S_1, S_2)$.

end

Theorem 3.2: Algorithm MAXDIST-2 computes the maximum distance between S_1 and S_2.

Proof: The correctness of Step 1 follows from Lemma 2.1. Because the diameter of a set is equal to the diameter of the convex hull of the set (see [3-4]), Steps 3 and 4 correctly compute the diameter of $CH(S_{1i}) \cup CH(S_{2j})$. Finally, the correctness of Steps 2 and 5 follows directly from Theorem 2.1. Q.E.D.

for j = 1 to n

begin

if $d(p_i, q_j) > d_{max}(S_1, S_2)$

then $d_{max}(S_1, S_2) \leftarrow d(p_i, q_j)$

end

end

Clearly, the above algorithm always runs on $O(n^2)$ time.

Algorithm MAXDIST-1

begin

Step 1: Compute $CH(S_1)$ and $CH(S_2)$.

Step 2: Compute $d_{max}(CH(S_1), CH(S_2))$ using the brute-force method, and output it as $d_{max}(S_1, S_2)$.

end

Theorem 3.1: Algorithm MAXDIST-1 computes the maximum distance between two sets S_1, S_2.

Proof: The proof follows directly from Lemma 2.1.

Worst-Case Analysis

In the worst case, Step 1 requires O(n log n) time. For a survey of O(n log n) convex hull algorithms, the reader is referred to [6]. In the worst case, all points of S_i will be vertices of $CH(S_i)$, i = 1, 2, and thus Step 2 will run in $O(n^2)$ time. Thus, the worst-case complexity of MAXDIST-1 is dominated by Step 2 and is $O(n^2)$.

Expected Analysis

Here we assume that $S_1 = \{p_1, p_2, ..., p_{n_1}\}$ consists of n_1 points drawn independently at random from some common underlying distribution f_1. A similar assumption is made concerning $S_2 = \{q_1, q_2, ..., q_{n_2}\}$ with respect to f_2. Note that S_1 and S_2 may be dependent, and f_1 need not be identical to f_2. Let N_1 and N_2 be the number of points in $CH(S_1)$ and $CH(S_2)$, respectively. N_1 and N_2 are possibly dependent random variables. We further assume that the expected value of N_i, $E\{N_i\} \leq \sqrt{n}$ and $n_1 + n_2 \leq n$. Then the complexity C of MAXDIST-1 is given by

$$C = C(CH) + k_1N_1 + k_2N_2 + k_3N_1N_2 + k_4, \quad (9)$$

Applying Lemma 2.4 yields:

$$d_{max}(S_{1k}, S_{2k'}) = \text{diam}(S_{1k} \cup S_{2k'}), \tag{6}$$

and from (1)

$$d_{max}(S_1, S_2) = \text{diam}(S_{1k} \cup S_{2k'}). \tag{7}$$

Now we show that the diameter of the union of the subsets of points containing the pair that realizes the maximum distance between the two sets is greater than the diameter if the union if any other pair of subsets, i.e.,

$$\text{diam}(S_{1k} \cup S_{2k'}) > \text{diam}(S_{1i} \cup S_{2j})$$

$$\text{for } i, j = 1,...,9,\ i \neq k, \text{ and } j \neq k'. \tag{8}$$

Assume that there exists a pair of subsets S_{1r}, S_{2s} such that $\text{diam}(S_{1r} \cup S_{2s}) > \text{diam}(S_{1k} \cup S_{2k'})$. Two cases arise: (A) the two points realizing $\text{diam}(S_{1r} \cup S_{2s})$ both belong to either S_{1r} or S_{2s}, or (B) one point belongs to S_{1r} and the other to S_{2s}. In case (A), it follows from Lemma 2.6 that

$$\text{diam}(S_{1r}) < d_{max}(S_1, S_2),$$

$$\text{diam}(S_{2s}) < d_{max}(S_1, S_2).$$

Furthermore, from Eq. (7) we have

$$\text{diam}(S_{1r}) < \text{diam}(S_{1k} \cup S_{2k'}),$$

which is a contradiction. In case B, if one point of $\text{diam}(S_{1r} \cup S_{2s})$ belongs to S_{1r} and the other to S_{2s}, then we have a distance between S_1 and S_2 that is greater than $\text{diam}(S_{1k} \cup S_{2k'}) = d_{max}(S_1, S_2)$ by Eq. (7), thus also creating a contradiction and establishing Eq. (8). Substituting Eq. (8) into Eq. (7) yields

$$d_{max}(S_1, S_2) = \max_{i,\, j} \{\text{diam}(S_{1i} \cup S_{2j})\}. \qquad \text{Q.E.D.}$$

3. The Algorithms

In this section we describe two algorithms for computing $d_{max}(S_1, S_2)$, prove their correctness and analyze their worst-case and expected running times. Since these algorithms are also compared with a brute-force algorithm, we also describe the latter.

Algorithm BRUTE-FORCE

$d_{max}(S_1, S_2)$ <-- 0

for i = 1 to n

begin

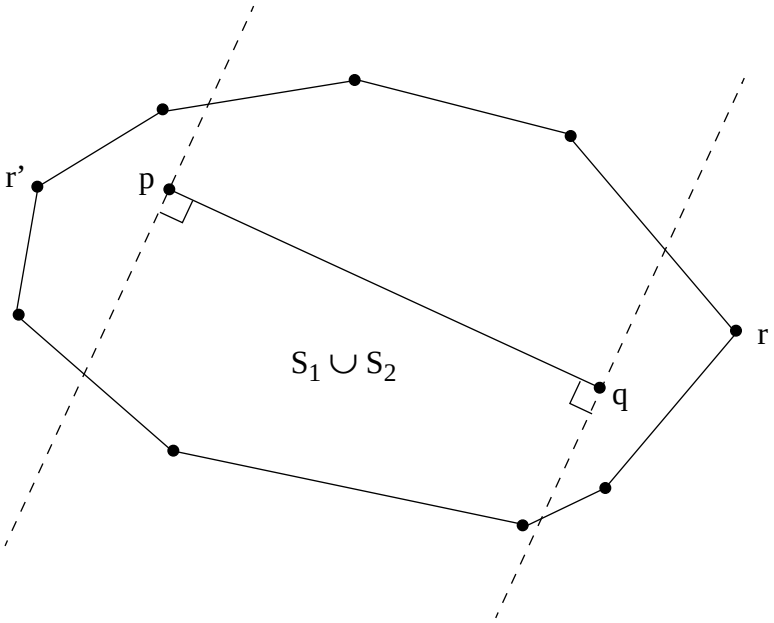

Fig. 5. Both points realizing $d_{max}(S_1, S_2)$ cannot be nonconvex hull points of $S_1 \cup S_2$.

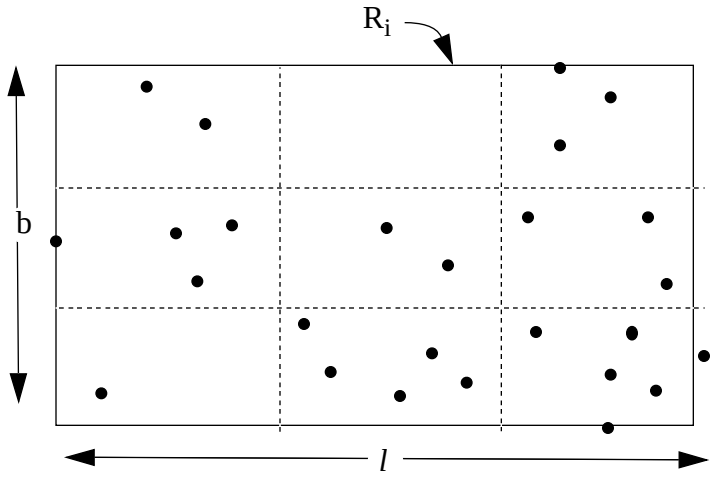

Fig. 6. Rectangle R_i is an encasing rectangle of S_i and it is partitioned into nine equal sub-rectangles.

From Lemma 2.6 it follows that

$$\text{diam}(S_{1k}) < d_{max}(S_1, S_2) \tag{2}$$

and

$$\text{diam}(S_{2k'}) < d_{max}(S_1, S_2). \tag{3}$$

Substituting (2) and (3) in (1) yields

$$\text{diam}(S_{1k}) < d_{max}(S_{1k}, S_{2k'}) \tag{4}$$

and

$$\text{diam}(S_{2k'}) < d_{max}(S_{1k}, S_{2k'}), \tag{5}$$

which implies that $d_{max}(S_{1k}, S_{2k'}) > \max [\text{diam}(S_{1k}), \text{diam}(S_{2k'})]$.

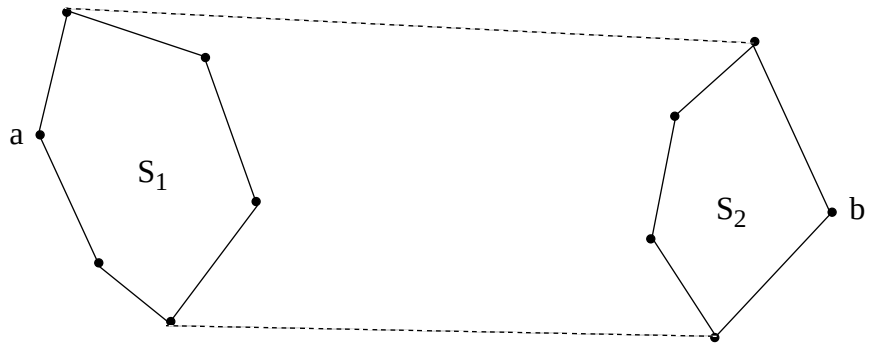

Fig. 2. An example where $d_{max}(S_1, S_2) = diam(S_1 \cup S_2) = d(a, b)$.

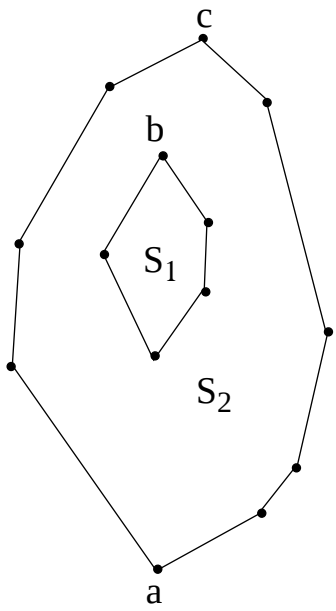

Fig 3. An example where $d_{max}(S_1, S_2) = d(a, b) \neq diam(S_1 \cup S_2) = d(a, c)$.

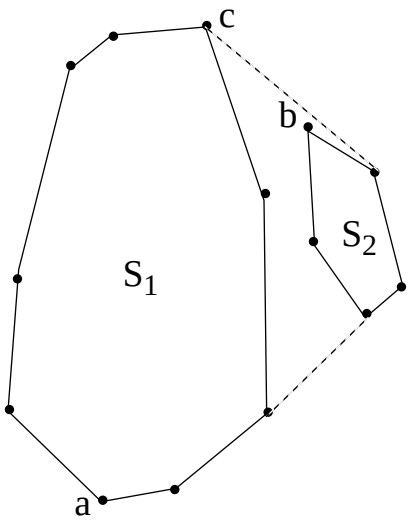

Fig. 4. Even if S_1 and S_2 are linearly separable both points realizing $d_{max}(S_1, S_2)$ need not belong to the convex hull of $S_1 \cup S_2$. Here $d_{max}(S_1, S_2) = d(a, b) \neq diam(S_1 \cup S_2) = d(a, c)$, and $b \notin CH(S_1 \cup S_2)$.

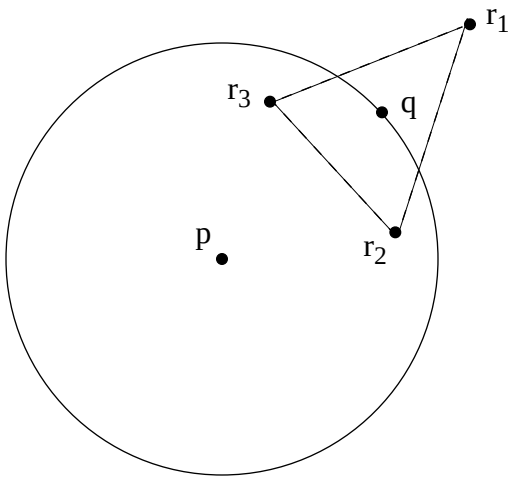

Fig. 1. Given a point $p \in S_1$, for every $q \in S_2$ and $q \notin CH(S2)$ there exists a convex hull point $r_1 \in S_2$ such that $d(p, r_1) > d(p, q)$.

Proof: Consider only the set S_1; similar arguments hold for S_2. First enclose the set S_1 in a rectangle R_1 with sides parallel to the x and y axes determined by the extreme points in the x and y directions. Let $p_{x\,min}$ be a point in S_1 with minimum x coordinate. Define $p_{x\,max}$, $p_{y\,min}$, $p_{y\,max}$ similarly. Let $l = x_{x\,max} - x_{x\,min}$, $b = y_{y\,max} - y_{y\,min}$, and $l > b$. Thus $diam(S_1) \geq l$.

The rectangle R_1 is now partitioned into nine sub-rectangles R_{1i}, $i = 1, 2, ..., 9$, each of length $l/3$ and breadth $b/3$, as illustrated in Fig. 6. Let S_{li} denote the set of points of S lying in R_{li}; thus $S_l = \bigcup_{1}^{9} S_{li}$. It follows that for each i, $i = 1, 2, ..., 9$ we have:

$diam(S_{li}) \leq$ diagonal of R_{li}

$= (l/3)\sqrt{l^2 + b^2}$

$< 0.5l$ since $l > b$ and $\sqrt{2}/3 < 0.5$

$< 0.5\ diam(S_1)$ since $diam(S_1) \geq l$

$< 0.5 \max [diam(S_1), diam(S_2)]$

$< d_{max}(S_1, S_2)$ by lemma 2.5

Q. E. D.

Theorem 2.1: $d_{max}(S_1, S_2) = \max_{i,\,j} \{diam(S_{li} \cup S_{2j})\}$.

Proof: Let $d_{max}(S_1, S_2) = d(p_i, q_j)$, where $p_i \in S_1$ and $q_j \in S_2$, and assume without loss of generality that $p_i \in S_{1k}$ and $q_j \in S_{2k'}$. Then we have

$$d_{max}(S_1, S_2) = d_{max}(S_{1k}, S_{2k'}) \qquad (1)$$

It should be noted that the claim made by several authors (such as Duda and Hart [1] and Johnson [2]) that $d_{max}(S_1, S_2) = diam(S)$ is not always true. Were this so, there would be no need for this paper since 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 [3, 4], or the furthest-point Voronoi diagram method of [5]. Although the idea of using the furthest-point Voronoi diagram to find the diameter of a set is originally due to Shamos [4], his algorithm is based on another invalid claim that the diameter is an edge in the dual of the Voronoi diagram. A counter-example to this claim and a modified diameter algorithm are given in [5]. Figure 2 illustrates an example where $d_{max}(S_1, S_2) = diam(S)$. Figure 3 shows an example where $d_{max}(S_1, S_2) \neq diam(S)$. Note that in Fig. 3 one of the points realizing $d_{max}(S_1, S_2)$ is not part of CH(S). This can remain true even if S_1 and S_2 are linearly separable as illustrated in Fig. 4. However, in the following lemma we establish that of the points realizing $d_{max}(S_1, S_2)$, at least one of them must belong to CH(S).

Lemma 2.3: One of the points realizing $d_{max}(S_1, S_2)$ must be a vertex of CH(S).

Proof: Let $p \in S_1$, and $q \in S_2$ realize $d_{max}(S_1, S_2)$ with $p, q \notin CH(S)$, and refer to Fig. 5. Construct lines through p and q perpendicular to the line segment joining p and q. Let $H(p, \overline{q})$ denote the closed half-plane, determined by the perpendicular through p, which does not contain q. Since p and q are not vertices of CH(S), there exists at least one vertex of CH(S) in $H(p, \overline{q})$. The same holds for $H(q, \overline{p})$. Let r and r' be two such points. Since $d(p, q) = d_{max}(S_1, S_2)$, $d(p,q) < d(p, r)$, and $p \in S_1$, it follows that $r \in S_1$. Similarly, we can conclude that $r' \in S_2$. Thus there exist two points $r, r' \in H(p, \overline{q}) \cup H(q, \overline{p})$ such that $d(r, r') > d(p, q)$, which is a contradiction. Therefore p and q cannot both be nonconvex hull points of S. Q.E.D.

Lemma 2.4: If $d_{max}(S_1, S_2) > \max[diam(S_1), diam(S_2)]$, then we have $d_{max}(S_1, S_2) = diam(S)$.

Proof: By partitioning the complete set of pairs of points of S into three subsets we can write:

$$diam(S) = \max\{ \max_{i,j} [d(p_i, q_j)], \max_{i,j} [d(p_i, p_j)], \max_{i,j} [d(q_i, q_j)]\}$$
$$= \max \{d_{max}(S_1, S_2), diam(S_1), diam(S_2)\}$$
$$= d_{max}(S_1, S_2) \quad \text{by assumption.}$$

Q.E.D.

Lemma 2.5: $d_{max}(S_1, S_2) \geq 0.5 \max [diam(S_1), diam(S_2)]$.

Proof: Without loss of generality assume that $diam(S_1) \geq diam(S_2)$ and let p_i, p_j realize $diam(S_1)$. Let q be a point of S_2. Then, by the triangle inequality,

$$d(p_i, q) + d(q, p_j) \geq d(p_i, p_j).$$

Furthermore, without loss of generality assume $d(p_i, q) \leq d(p_j, q)$. Then we have $0.5\, d(p_i, p_j) \leq d(p_j, q) \leq d_{max}(S_1, S_2)$. Since $d(p_i, p_j) = diam(S_1) \geq diam(S_2)$, the lemma follows. Q. E. D.

Lemma 2.6: Each set S_1 and S_2 can be partitioned into nine subsets such that the diameter of each subset is less than $d_{max}(S_1, S_2)$.

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

where

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

Clearly $d_{mean}(S_1, S_2)$ can be computed in O(n) time. Duda and Hart [1] 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 this paper we show that $d_{max}(S_1, S_2)$ can be computed in O(n log n) time in the worst case. A second algorithm that first computes the convex hulls of S_1 and S_2 runs in $O(n^2)$ worst-case time. Both algorithms achieve O(n) expected running time for many underlying distributions of the points. Furthermore, when S_1 and S_2 are simple polygons, $d_{max}(S_1, S_2)$ can be computed in O(n) time in the worst case.

After establishing some preliminary results in Section 2, the algorithms and their complexity analyses are presented in Section 3. Section 4 describes timing experiments comparing the above algorithms with each other and the brute-force method. An ε-approximation algorithm that runs in $O(n + 1/\varepsilon)$ time is presented in Section 5. Some concluding remarks are made in Section 4.

2. Preliminary Results

Let $CH(S_i)$, $i = 1,2$ denote the set of points of S_i that are vertices of the convex hull of S_i. Denote the diameter of S_i by $diam(S_i)$. For example, for the set S_1 we have

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

We now establish a series of results which will form the theoretical foundation for the algorithms of Section 3.

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

Proof: Let $p \in S_1$, and let $q \in S_2$ such that $q \notin CH(S_2)$. Construct a circle C centered at p with radius d(p, q), and refer to Fig. 1. Since $q \notin CH(S_2)$, there exists at least three vertices r_1, r_2, $r_3 \in CH(S_2)$ such that q lies in the triangle formed by r_1, r_2, and r_3. Therefore, at least one point, say r_1, must lie outside the circle C. Thus $d(p, q) < d(p, r_1)$ and d(p, q) cannot be $d_{max}(S_1, S_2)$. Since the argument holds whether p does or does not belong to $CH(S_1)$, it follows that if $p \notin CH(S_1)$ and/or $q \notin CH(S_2)$, then p, q cannot realize $d_{max}(S_1,S_2)$. Therefore, if p, q realize $d_{max}(S_1, S_2)$, then $p \in CH(S_1)$ and $q \in CH(S_2)$. Q.E.D.

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

Proof: By definition, $diam(S) = diam(S_1 \cup S_2)$ is the maximum over the complete set of distances G of S. On the other hand, $d_{max}(S_1, S_2)$ is the maximum over a set of distances G'. Since $G' \subseteq G$, the lemma follows. Q.E.D.

Efficient Algorithms for Computing the Maximum Distance Between Two Finite Planar Sets

Binay K. Bhattacharya
and
Godfried T. Toussaint

McGill University
School of Computer Science
Montreal, Quebec, Canada

ABSTRACT

An 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 this complexity can be reduced to O(n). The algorithm is empirically compared to the brute-force method as well as an alternate $O(n^2)$ algorithm. Both the O(n log n) and $O(n^2)$ algorithms run in O(n) expected time for many underlying distributions of the points. An ε-approximate algorithm can be obtained that runs in $O(n + 1/\varepsilon)$ worst-case time.

1. Introduction

Let $S_1 = p_1, p_2,..., p_n$ and $S_2 = q_1, q_2,..., q_n$ be two planar sets of n points, and let $S = S_1 \cup S_2$. (The sets need not have equal cardinality, but this assumption simplifies notation.) A point p_i is given by 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

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

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

The computation of distances between sets arises in pattern recognition problems [1], where clustering is a prime example. In agglomerative (bottom-up) clustering procedures, one starts with a set of *N* clusters each containing 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* [1]. Another frequently used distance between sets is

* Received by *Journal of Algorithms* on June 8, 1981; revised May 15, 1982.