JOURNAL OF ALGORITHMS 6, 213–224 (1985)

Computing the Extreme Distances between Two Convex Polygons

H. EDELSBRUNNER

Institutes for Information Processing, Technical University of Graz, Schiesstattgasse 4a, A-8010 Graz, Austria

Received October 21, 1982; revised December 20, 1983

A polygon in the plane is convex if it contains all line segments connecting any two of its points. Let P and Q denote two convex polygons. The computational complexity of finding the minimum and maximum distance possible between two points p in P and q in Q is studied. An algorithm is described that determines the minimum distance (together with points p and q that realize it) in $O(\log m + \log n)$ time, where m and n denote the number of vertices of P and Q, respectively. This is optimal in the worst case. For computing the maximum distance, a lower bound $\Omega(m + n)$ is proved. This bound is also shown to be best possible by establishing an upper bound of $O(m + n)$. © 1985 Academic Press, Inc.

1. INTRODUCTION

A *polygon* P is a bounded, closed, and connected subset of the real plane. The boundary of P consists of a finite number of straight line segments called *edges*. P is called *convex* if it contains all line segments that connect any two of its points. The endpoints of the edges of P are called its *vertices*.

Let P and Q denote two convex polygons with m and n vertices, respectively, and let $d(p, q)$ be the Euclidean distance between two points p and q in the plane. We call $\{d(p, q) | \ p \text{ in } P \text{ and } q \text{ in } Q\}$ the (*Euclidean*) *distance set* of P and Q. The extreme values in the distance set are the primary interest of this paper. More specifically, we study the complexity of computing the minimum and maximum distance and of finding pairs of points that realize them.

Finding the minimum distance possible between P and Q has applications in planning collision-free paths through a set of obstacles, see Schwartz [7] who gives an algorithm that computes it in $O((\log m)(\log n))$ time. We improve on this result by exhibiting an algorithm that determines the minimum distance, along with a pair of points that realize it, in time

0196-6774/85 $3.00

$O(\log m + \log n)$. This result is described in Section 3. Section 4 examines the complexity of computing the maximum value in the distance set of P and Q. A lower bound of $\Omega(m + n)$ time in the worst case is proved and an optimal algorithm that runs in $O(m + n)$ time is sketched. The geometric facts needed for an appropriate presentation of the algorithms in Sections 3 and 4 are developed in Section 2. Finally, Section 5 reviews the main contributions and briefly addresses the extension of the problems to three dimensions.

2. Geometric Preliminaries

This section develops the geometric facts needed for the algorithms in Sections 3 and 4. As in Section 1, P and Q denote two convex polygons with m and n vertices, respectively. Let $d(P, Q)$ denote the minimum value and $D(P, Q)$ the maximum value in the distance set of P and Q.

Lemma 2.1. *If $d(P, Q) > 0$, then there exist points p in P and q in Q that realize $d(P, Q)$ such that p and q are vertices or either of them is a vertex and the other lies on an edge.*

Proof. Note first that neither p nor q can be an inner point. The remaining possibility for proving Lemma 2.1 false is that p and q lie on edges but neither of them can be a vertex. Let p lie on edge e_p of P and let q lie on edge e_q of Q (see Fig. 2.1). Then e_p and e_q are necessarily parallel and we may assume that they are vertical. W.l.o.g. assume also that the upper endpoint v of e_p is lower than the upper endpoint of e_q. But then v

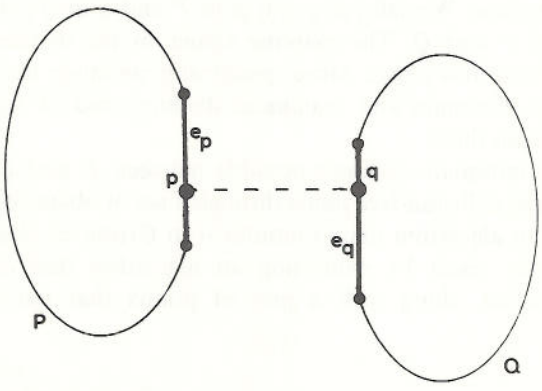

Figure 2.1

and its orthogonal projection onto e_q realize $d(P, Q)$. This completes the argument by contradiction.

The possibilities for the points that realize $D(P, Q)$ are even more restrictive.

LEMMA 2.2. *If p in P and q in Q realize $D(P, Q)$ then p and q are vertices of P and Q, respectively.*

Proof. Note again that p and q lie necessarily on the boundaries of P and Q, respectively. We assume w.l.o.g. that p is not a vertex of P and lies on some edge of P. Let q in Q maximize the distance to p (see Fig. 2.2). Then at least one of the endpoints of the edge that contains p is further away from q than p. This contradicts the assumption and completes the argument.

Intuitively, the next fact asserts that a point of P that realizes the minimum distance to Q "sees" all of Q. To simplify its formulation, let $s(p,q)$ denote the line segment that connects p and q.

LEMMA 2.3. *Let $d(P, Q) > 0$ and p in P such that there exists q in Q with $d(p, q) = d(P, Q)$. Then p is the only common point of P and $s(p, q'')$, for every point q'' of Q.*

Proof. By Lemma 2.1, p lies necessarily on the boundary of P. Assume now that there exists a point q'' in Q such that $s(p, q'')$ intersects P in at least one additional point p' (see Fig. 2.3). Let q' on $s(q, q'')$ such that $s(p, q)$ and $s(p', q')$ are parallel. By convexity, q' is contained in Q, and $d(p', q') < d(p, q)$. This contradicts the assumption and completes the argument.

A related fact holds for pairs of points that realize $D(P, Q)$.

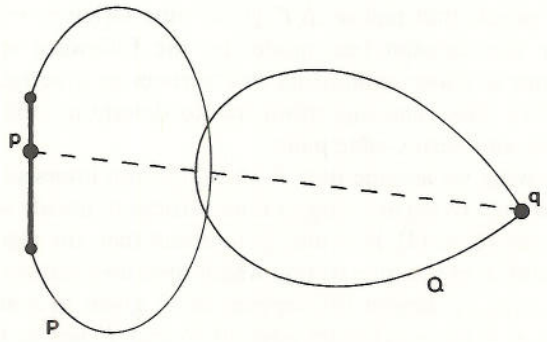

FIGURE 2.2

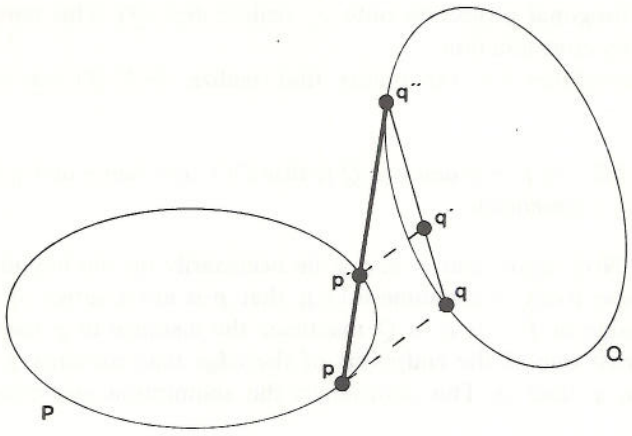

FIGURE 2.3

LEMMA 2.4. *Let p in P and q in Q realize $D(P, Q)$. Then the orthogonal projection of an arbitrary point x of P or Q onto the line supporting $s(p, q)$ lies between p and q.*

Proof. Assume the contrary, that is, there is a point x in P or Q whose orthogonal projection onto the very line does not fall in between p and q. W.l.o.g. let x project onto p or p lie between the projection of x and q. Then $d(q, x) > d(q, p)$ which contradicts the assumption and completes the argument.

3. COMPUTING THE MINIMUM DISTANCE

This section investigates the complexity of computing the minimum distance $d(P, Q)$ between two convex polygons P and Q with m and n vertices, respectively. An algorithm is outlined that computes $d(P, Q)$ along with a pair of points that realize $d(P, Q)$ in time $O(\log m + \log n)$. This is optimal under the decision tree model by the following argument: The minimal distance is either realized by two vertices or a vertex and an edge (see Lemma 2.1). Thus any algorithm has to determine one out of 3 mn pairs of vertices and vertex-edge pairs.

For the following we assume that P and Q do not intersect. When this is not guaranteed then $O(\log m + \log n)$ time suffices to decide whether or not they intersect, see [2] or [4]. If P and Q intersect then the algorithms in [2], [4] deliver a point x in the intersection which therefore realizes $d(P, Q) = 0$.

Let $v_0, v_1, \ldots, v_{m-1}$ denote the vertices of P given in counterclockwise order. That is, $e_i = (v_i, v_{i+1})$ is an edge of P and P lies to the left of the directed line that passes first through v_i and then through v_{i+1}. (The indices

are taken modulo m.) Similarly, let $w_0, w_1, \ldots, w_{n-1}$ be the vertices of Q in counterclockwise order and let $f_i = (w_i, w_{i+1})$ be the edges of Q, taking the indices modulo n. We assume that both sequences of vertices are stored in one-dimensional arrays. For convenience, we use the capital letters P and Q to denote these arrays also.

Our approach to computing $d(P, Q)$ is to perform binary search in P and Q simultaneously. At each step half of the edges of at least one remaining sequence are eliminated. To this end, we first need to identify two subsequences of the vertices and edges of P and Q that permit the simultaneous binary search strategy. We now concentrate on this initial step which determines a subsequence P' of vertices and edges of P and a subsequence Q' of vertices and edges of Q. P' and Q' will be chosen such that the following two conditions are satisfied:

(i) If p in P and q in Q realize $d(P, Q)$ then p is on P' and q is on Q', and

(ii) the edges of P' and Q' are not edges of the convex hull of the union of P and Q (see Fig. 3.1).

P' and Q' are computed by the following algorithm:

Procedure INITIAL PHASE

Let v and w denote arbitrary points in P and Q, respectively (e.g., $v = v_0$ and $w = w_0$).

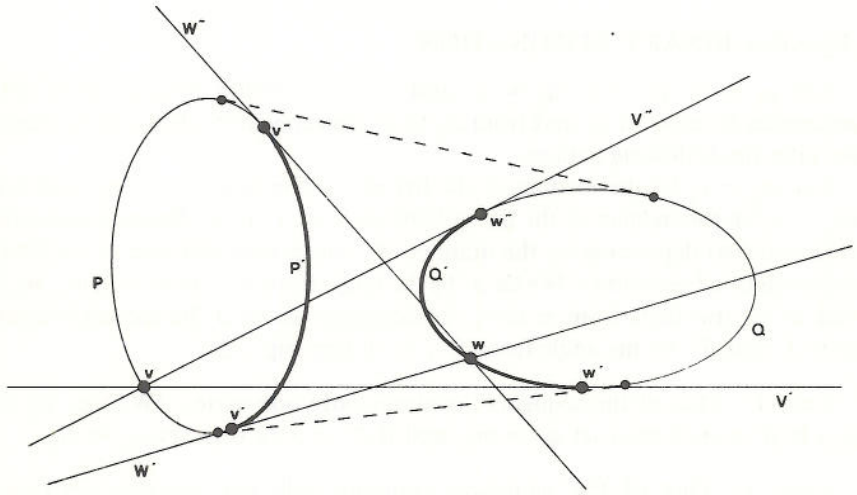

FIG. 3.1 Construction of P' and Q'.

Determine the two lines V' and V'' that pass through v and are tangent to Q such that Q is contained in the wedge from V' to V'' (see Fig. 3.1). Let w' and w'' denote the vertices closest to v where V' and V'' touch Q. Analogously, determine the two lines W' and W'' that contain w and are tangent to P such P is contained in the wedge from W'' to W'. Let v' and v'' denote the vertices closest to w where W' and W'' touch P, respectively.

We choose the sequence of vertices and edges from v' to v'' as P' and the sequence of vertices and edges from w'' to w' as Q' (see Fig. 3.1).

LEMMA 3.1. (a) *P' and Q', as computed by Procedure INITIAL PHASE, satisfy conditions* (i) *and* (ii).

(b) Procedure INITIAL PHASE can be implemented such that it computes P' and Q' in $O(\log m + \log n)$ time.

Proof. Concerning part (a): Condition (ii) is clear from the construction. By Lemma 2.3, every point p in P that realizes $d(P, Q)$ with some q in Q is contained in P'. By symmetry, the same is true for all q in Q that realize $d(P, Q)$ with some p in P.

Concerning part (b): The amount of time required by the procedure depends on the amount of time needed to find a tangent that contains a specified point outside the polygon. Chazelle and Dobkin [2] show that this task can be accomplished in time $O(\log k)$, where k is the number of vertices of the polygon. Thus, $O(\log m + \log n)$ time suffices for Procedure INITIAL PHASE. This completes the argument.

Now we are ready to present the algorithm that computes $d(P, Q)$ along with points p in P and q in Q that realize $d(P, Q)$.

Algorithm BINARY ELIMINATION

Let $p_1 := v'$, $p_2 := v''$, $q_1 := w'$, and $q_2 := w''$. While at least one of the sequences from p_1 to p_2 and from q_2 to q_1 contains more than two vertices, we take the following actions:

Let $m_p = v_i$ be the median in the list of vertices from p_1 to p_2, and let $m_q = w_j$ be the median in the list of vertices from q_2 to q_1. Several cases are distinguished depending on the number of vertices involved and on the four angles defined as follows: We let α' be the angle from e_{i-1} to $m = s(m_p, m_q)$ and α'' be the angle from m to e_i. Analogously, we let β' be the angle from m to f_j and β'' be the angle from f_{j-1} to m (see Fig. 3.2).

Case 1. One of the sequences contains only one vertex. (W.l.o.g. $p_1 = p_2$.) If $\beta' \geq \pi/2$ then set $q_1 := m_q$, and if $\beta'' \geq \pi/2$ then set $q_2 := m_q$.

Case 2. One of the sequences contains only two vertices. (W.l.o.g. $p_2 = m_p$ and p_1 precedes p_2.) Two cases are distinguished:

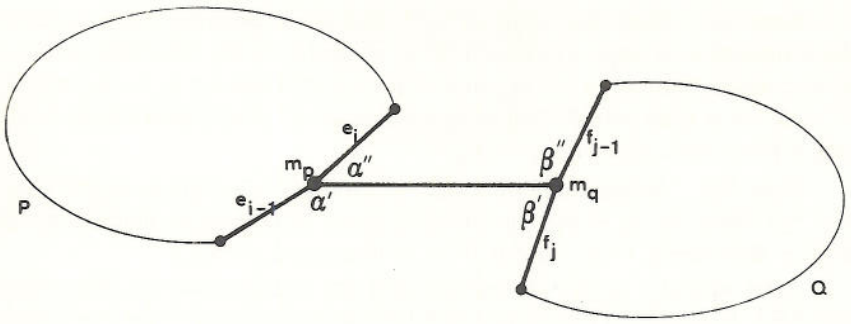

FIGURE 3.2

Case 2.1. $\alpha' > 0$. Then perform the following three steps: (1) If $\alpha' + \beta' > \pi$ then take the following actions: If $\alpha' \geqslant \pi/2$ then the set $p_1 := p_2$ and if $\beta' \geqslant \pi/2$ then set $q_1 := m_q$. (2) If $\beta'' \geqslant \pi/2$ then set $q_2 := m_q$. Finally, (3) if $\alpha' < \beta'' < \pi/2$ then take the following actions: If the orthogonal projection of m_q onto $s(p_1, p_2)$ exists then set $q_2 := m_q$. Otherwise set $p_2 := p_1$. (See Fig. 3.3 (a) for an illustration).

Case 2.2. $\alpha' \leqslant 0$. Then set $p_2 := p_1$. Additionally, we may proceed as follows: If $\beta' \geqslant \pi$ then set $q_1 := m_q$ and if $\beta'' \geqslant \pi$ then set $q_2 := m_q$. (See Fig. 3.3 (b) for an illustration.)

Case 3. Both sequences contain at least three vertices each (see Fig. 3.2). Again two cases are distinguished.

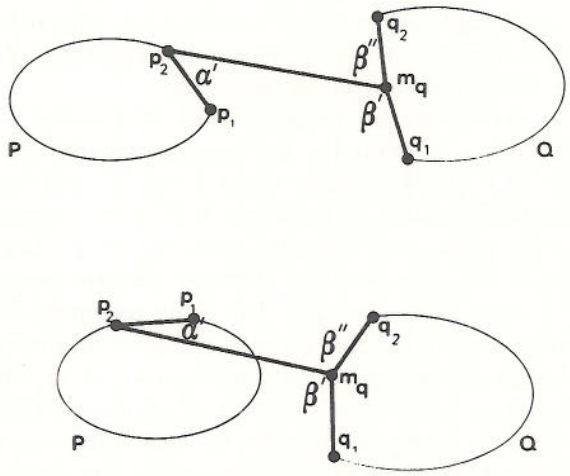

FIGURE 3.3

Case 3.1. Each one of α', α'', β', and β'' is positive. Then perform the following two steps: (1) If $\alpha' + \beta' > \pi$ then take the following actions: If $\alpha' \geq \pi/2$ then set $p_1 := m_p$ and if $\beta' \geq \pi/2$ then set $q_1 := m_q$. (2) If $\alpha'' + \beta'' > \pi$ then take the following actions: If $\alpha'' \geq \pi/2$ then set $p_2 := m_p$ and if $\beta'' \geq \pi/2$ then set $q_2 := m_q$.

Case 3.2. At least one of α', α'', β', and β'' is nonpositive. (W.l.o.g. $\alpha' \leq 0$.) Then set $p_2 := m_p$. In addition, parts of Q' can be eliminated: If $\beta' \geq \pi$ then set $q_1 := m_q$ and if $\beta'' \geq \pi$ then set $q_2 := m_q$.

A brief analysis of the correctness and the requirements of Algorithm BINARY ELIMINATION is presented before we proceed to the evaluation of its output.

LEMMA 3.2. *Algorithm BINARY ELIMINATION returns a pair of vertices, a vertex and an edge, or two edges in* $O(\log m + \log n)$ *time. In any case, the pairs of points that realize* $d(P, Q)$ *are contained in what is delivered.*

Proof. We first show that at least half of the edges of at least one sequence are eliminated at each step. By convexity of P and Q we have

(i) $\beta' \geq \pi/2$ or $\beta'' \geq \pi/2$,

(ii) $\alpha' + \beta' \leq \pi$ implies $\alpha' < \beta''$, and

(iii) $\alpha' + \beta' > \pi$ or $\alpha'' + \beta'' > \pi$.

Now, (i) guarantees that the desired amount of edges is eliminated in Case 1. By (ii), the same is true in Case 2.1, and (iii) implies the elimination of the appropriate number of edges in Case 3.1. Cases 2.2 and 3.2 lead trivially to the deletion of enough edges. As a consequence, Algorithm BINARY ELIMINATION stops in $O(\log m + \log n)$ time. Clearly, it delivers either two vertices, a vertex and an edge, or two edges.

It remains to be shown that no point p of P' (and q of Q') is ever eliminated if a point q' in Q (and p' in P) exists such that p and q' (and q and p') realize $d(P, Q)$. We discuss Cases 1 and 2.1 and leave the other cases to the interested reader. First Case 1: If $\beta' \geq \pi/2$ then $d(p_1, m_q) < d(p_1, q)$ for every point q on the edges from m_q to q_1. Thus, the action $q_1 = m_q$ taken in this case does not eliminate relevant points. The same holds for $\beta'' \geq \pi/2$. Now Case 2.1: The same reasoning as for Case 1 implies the correctness of subcases (1) and (2). Let us thus consider subcase (3) (see Fig. 3.3 (a)). If the orthogonal projection m_q' of m_q onto $s(p_1, p_2)$ exists then $d(q, s(p_1, p_2)) > d(m_q, m_q')$ for all points q different from m_q on the edges from q_2 to m_q. If m_q' does not exist then $d(q, s(p_1, p_2)) > d(m_q, p_1)$ for all of these points q. This implies the correctness of the actions taken and completes the argument.

Next let us consider the final phase that ultimately computes $d(P, Q)$. In this phase, the constant amount of information provided by Algorithm BINARY ELIMINATION is exploited.

Procedure FINAL PHASE

Case 1. Two vertices p_1 and q_1 are delivered. Then $d(P, Q) = d(p_1, q_1)$.

Case 2. A vertex p_1 and an edge $f = (q_2, q_1)$ is delivered. Let $f(p_1)$ denote the orthogonal projection of p_1 onto f (if it exists). Due to Lemma 2.1, $d(P, Q)$ is realized either by p_1 and q_1, by p_1 and q_2, or by p_1 and $f(p_1)$.

Case 3. Two edges $e = (p_1, p_2)$ and $f = (q_1, q_2)$ are delivered. Let $f(p_i)$ denote the orthogonal projection of p_i onto f, for $i = 1, 2$, if it exists, and let $e(q_i)$ denote the orthogonal projection of q_i onto e, for $i = 1, 2$, if it exists. Then $d(P, Q)$ is realized either by p_1 and q_1, p_1 and q_2, p_2 and q_1, or p_2 and q_2, or by p_1 and $f(p_1)$, p_2 and $f(p_2)$, q_1 and $e(q_1)$, or q_2 and $e(q_2)$.

Having described all computations for finding the minimum distance, we state the main result of this section.

THEOREM 3.3. *Let P and Q denote two convex polygons with m and n vertices, respectively. The minimum distance between P and Q, along with points p in P and q in Q that realize it, can be computed in $O(\log m + \log n)$ time.*

Proof. The task is accomplished in three phases. Procedure INITIAL PHASE identifies two appropriate subsequences of the boundaries of P and Q in $O(\log m + \log n)$ time (see Lemma 3.1). In additional $O(\log m + \log n)$ time, Algorithm BINARY ELIMINATION determines a pair of vertices, a vertex and an edge, or a pair of edges that contain all pairs (p, q) realizing $d(P, Q)$ (see Lemma 3.2). Procedure FINAL PHASE then computes $d(P, Q)$ along with p in P and q in Q that realize $d(P, Q)$ in constant time. This completes the argument.

At this point we note that Chin and Wang [3] independently proved $O(\log m + \log n)$ as an upper bound for finding the minimum distance. They did that by developing an essentially identical algorithm.

4. COMPUTING THE MAXIMUM DISTANCE

This section shows that computing the maximum distance between two convex polygons is harder than computing the minimum distance. In contrast to the case of the minimum distance, now convexity is of no use at

all. The lower bound established below relies on representing a polygon by the linear list of its vertices and edges. Nevertheless, we find the result significant since this seems to be the most natural representation. The existence of a representation that allows the computation of the maximum distance in sublinear time remains an open problem.

THEOREM 4.1. *Let P and Q denote convex polygons with m and n vertices stored in linear arrays, respectively. Then $\Omega(m+n)$ is a lower bound on the worst-case time complexity of any algorithm that computes the maximum distance $D(P,Q)$.*

Proof. We show the assertion by application of an adversary argument to a special choice of P and Q: P degenerates to a single point and the vertices of Q lie on a circle whose center is P (see Fig. 4.1). Thus, if any vertex of Q withdraws from P as long as Q remains convex then $D(P,Q)$ increases while only one vertex of Q changes. This shows that each vertex of Q has to be considered at least once in order to compute $D(P,Q)$ correctly. This completes the argument.

The lower bound given above is asymptotically best possible as follows from the algorithm outlined in the proof of Theorem 4.2. The same algorithm was developed independently in Toussaint and McAlear [9]. A

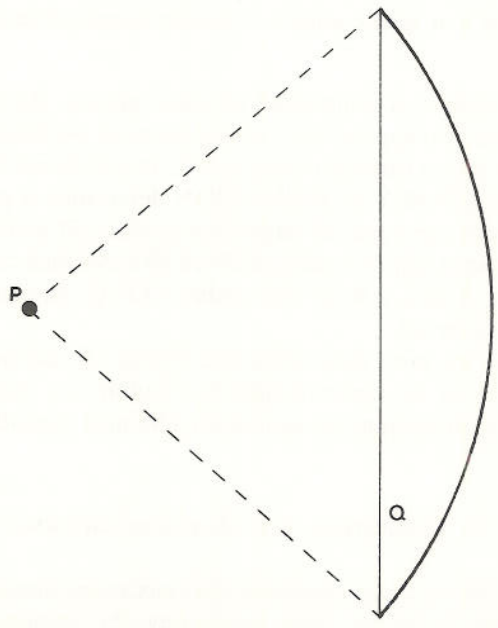

FIG. 4.1 Polygons for the adversary argument.

more complicated algorithm that also runs in linear time can be found in [1].

THEOREM 4.2. *Let P and Q denote convex polygons with m and n vertices, respectively. There exists an algorithm that computes $D(P, Q)$, along with points p in P and q in Q that realize it, in $O(m + n)$ time.*

Proof. The algorithm maintains two parallel supporting lines L_P and L_Q for P and for Q such that either P lies above L_P and Q lies below L_Q or P lies below L_P and Q above L_Q (see Fig. 4.2). If a line is vertical than we say that the right-hand side is below the line and the left-hand side is above it.

A pair of vertices (v, w), with v in P and w in Q is called *antipodal* if there exist two such parallel supporting lines L_P and L_Q that contain v and w, respectively. By Lemma 2.4, the points p in P and q in Q that realize $D(P, Q)$ are antipodal.

The algorithm computes all antipodal pairs of vertices by turning L_P and L_Q around P and Q. In Shamos [8], a similar strategy is used to compute the diameter of a convex polygon. We thus refer to [8] for details. An easy argument shows that there are at most $O(m + n)$ antipodal pairs, and that the algorithm requires constant time per pair. This implies that the algorithm sketched requires $O(m + n)$ time which completes the argument.

Marginally, we mention that $O(m + n)$ time also suffices to compute the maximum distance between two simple but not necessarily convex polygons. In a first step, the convex hulls of both polygons are determined. This can be done in $O(m + n)$ time with an algorithm of McCallum and Avis [5]. Then the algorithm sketched in the proof of Theorem 4.2 is used to compute

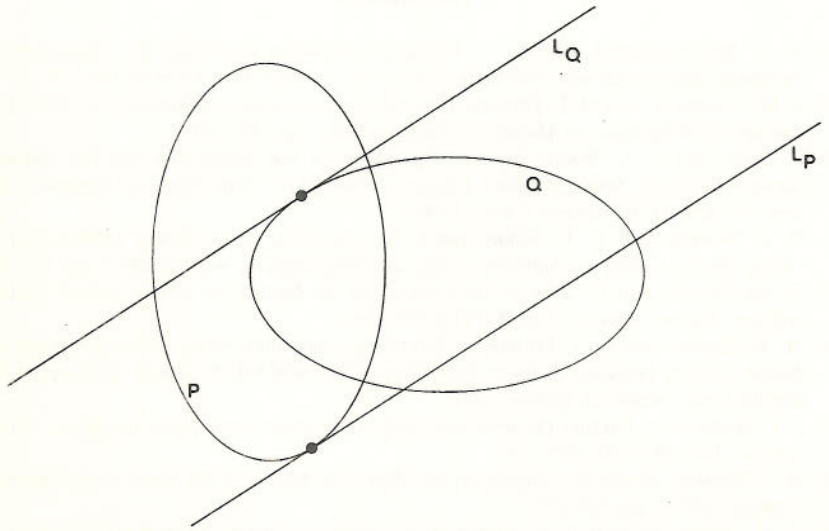

FIG. 4.2 Polygons with supporting lines.

the maximum distance between the two convex hulls. By Lemma 2.2, this distance is also the maximum distance between the two original polygons.

5. Discussion

This paper investigates the complexity of computing the extreme (Euclidean) distances realized by two convex polygons in the plane. For both problems, that is, for computing the minimum and maximum distance, optimal algorithms are described. The minimum distance between two convex polygons with m and n vertices, respectively, is determined in $O(\log m + \log n)$ time. This improves a previous result of Schwartz [7] whose algorithm requires $O((\log m)(\log n))$ time. For finding the maximum distance, $\Omega(m + n)$ time is shown to be necessary and sufficient.

Our results do not yield an efficient solution for determining the nearest vertices of convex polygons. Although, at first sight, this problem seems to be closely related to the minimum distance problem investigated in this paper, the complexity is $\Theta(m + n)$ as shown in [6].

The most important extension of this work is the investigation of the analogous problems in three dimensions. The computational complexity depends then highly on the kind of representation chosen for the convex polytopes. We refer to Dobkin and Kirkpatrick [4] whose hierarchical representations seem to be a promising approach for fast computation of the minimum distance.

References

1. B. K. Bhattacharya and G. T. Toussaint, Efficient algorithms for computing the maximum distance between two finite planar sets, *J. Algorithms* **4** (1983), 121–136.
2. B. M. Chazelle and D. P. Dobkin, Detection is easier than computation, *in* "Proc. 12th Annual ACM Sympos. on Theory of Comput., 1980," pp. 146–153.
3. F. Chin and C. K. Wang, Optimal algorithms for the intersection and the minimum distance problems between planar polygons, manuscript, Department of Computer Sci., Univ. of Alberta, Edmonton, Canada, 1982.
4. D. P. Dobkin and D. G. Kirkpatrick, Fast detection of polyhedral intersections, *in* "Proc. 9th Int. Colloq. on Automata, Lang., and Programming, Aarhus, 1982," pp. 154–165.
5. D. McCallum and D. Avis, A linear algorithm for finding the convex hull of a simple polygon, *Inform. Process. Lett.* **9** (1979), 201–206.
6. M. McKenna and G. T. Toussaint, Finding the minimum vertex distance between two disjoint convex polygons in linear time, Report No. SOCS-83.6, School of Comput. Sci., McGill Univ., Montreal, Quebec, 1983.
7. J. T. Schwartz, Finding the minimum distance between two convex polygons, *Inform. Process. Lett.* **13** (1981), 168–170.
8. M. I. Shamos, Geometric complexity, *in* "Proc. 7th Annual ACM Sympos. on Theory of Comput., 1975," pp. 224–233.
9. G. T. Toussaint and J. A. McAlear, A simple $O(n \log n)$ algorithm for finding the maximum distance between two finite planar sets, *Pattern Recognition Lett.*, in press.