Comp. & Maths. with Appls. Vol. 11, No. 12, pp. 1227–1242, 1985
Printed in Great Britain.

0097–4943/85 $3.00 + .00

FINDING THE MINIMUM VERTEX DISTANCE BETWEEN TWO DISJOINT CONVEX POLYGONS IN LINEAR TIME

MICHAEL MCKENNA and GODFRIED T. TOUSSAINT
School of Computer Science, McGill University, 805 Sherbrooke Street West, Montreal, Quebec, Canada H3A 2K6

Communicated by Ervin Y. Rodin

Abstract Let $V = \{v_1, v_2, \ldots, v_m\}$ and $W = \{w_1, w_2, \ldots, w_n\}$ be two linearly separable convex polygons whose vertices are specified by their cartesian coordinates in order. An algorithm with $O(m + n)$ worst-case time complexity is described for finding the minimum euclidean distance between a vertex v_i in V and a vertex w_j in W. It is also shown that the algorithm is optimal.

1. INTRODUCTION

Let V be a convex polygon described by an ordered circular list $\{v_1, v_2, \ldots, v_m\}$ of its m vertices. Each edge of V is described by a pair (v_i, v_{i+1}) of consecutive vertices. We will say that the set of points in the polygon includes the interior of the polygon and its boundary. Let W be another convex polygon similarly described by another list $\{w_1, w_2, \ldots, w_n\}$ of its n vertices. We assume that V and W do not intersect.

Let $d(p,q)$ be the Euclidean distance between points p and q. This paper describes an $O(m + n)$ optimal algorithm for finding a pair of vertices (v_i, w_j) such that for any other pair of vertices (v_k, w_l), the following condition holds: $d(v_i, w_j) \leq d(v_k, w_l)$.

This problem is significant for several reasons. In the theory of geometric complexity it was an open question whether the problem could be solved in linear time. In practical applications in pattern recognition and cluster analysis[8, 9] one often encounters problems where it is desired to compute distances between sets of points representing objects or pattern classes. Finally, it is likely that other geometric problems may require the computation of this distance for their solution.

Recently, Chin and Wang[3] have independently found another algorithm for this same problem. Their solution is similar to ours to the extent that they also perform an initial decomposition of the problem into four subproblems. However, their decomposition and solutions to the subproblems vary considerably from ours. In particular, our approach is based on the application of existing results for computing the relative neighbourhood graph of a convex polygon[7]. These differences will be highlighted in later sections.

Edelsbrunner[4] describes an optimal $O(\log m + \log n)$ algorithm for finding the minimum distance between two convex polygons, where the nearest points are not restricted to vertices. This improves an earlier algorithm for this problem due to Schwartz[5] which runs in $O((\log m)(\log n))$ time.

The following section presents some preliminary definitions and lemmas. Section 3 outlines a linear-time ten-step procedure for finding the nearest pair of vertices between V and W. Section 4 shows how steps six, seven and eight of the ten-step procedure can be performed in linear time and Sec. 5 shows how step nine can be performed in linear time. Section 6 presents a proof that the nearest-vertices problem has an $\Omega(m + n)$ worst-case lower-bound. Finally, Sec. 7 presents some concluding remarks and open problems. As a final comment we remark that where the proofs can readily be reconstructed by the reader, we omit them in the interest of brevity.

2. PRELIMINARY DEFINITIONS AND RESULTS

DEFINITION

The *bridge* between convex polygons V and W is the segment whose endpoints are the nearest points found by Edelsbrunner's algorithm. Note that the bridge may be realized by a vertex and an edge-point.

LEMMA 1

If the bridge between V and W is realized by the pair of vertices (v_i,w_j), then the shortest distance between the vertices of V and W is also realized by this pair of vertices.

DEFINITION

A *corridor* is a region bounded by two distinct parallel lines. We will say that the set of points in the corridor does not include the bounding lines. The complement of a corridor consists two disconnected closed half-planes. Each of these half-planes will be called a *half-complement*.

DEFINITION

We will say that every segment *induces* a corridor having the following characteristics:

(1) the bounding lines of the corridor are perpendicular to the inducing segment, and
(2) one bounding line passes through each endpoint of the segment.

DEFINITION

The *lune* of a segment (p,q) is the set of points r in the plane such that $d(p,r) < d(p,q)$ and $d(q,r) < d(q,p)$. Note that the lune consists of the intersection of the interior of the two circles. Both of these circles have a radius equal to $d(p,q)$. One circle is centered at point p, and the other is centered at point q.

LEMMA 2

Let p and q be the endpoints of the bridge between V and W, and let L be the lune of bridge (p,q). The following two conditions hold:

(i) There cannot be any point $s \in V$ in L, and
(ii) There cannot be any point $t \in W$ in L.

LEMMA 3

The bridge between convex polygons V and W induces a corridor K falling between the polygons V and W. In other words,

(i) $V \cap K = \phi$ and $W \cap K = \phi$, and
(ii) V and W lie in separate half-complements of K.

Proof (by contradiction) of condition (i). (See Fig. 1.) Let p and q be the endpoints of the bridge. From Lemma 2, there cannot be any point of V or W in the lune of bridge (p,q). Without loss of generality let us now assume that $V \cap K \neq \phi$. This means that there exists some point $t \in V \cap K$. Point t is in K, thus on segment (p,t) there must exist a point z in the lune. Polygon V is convex, therefore segment (p,t) lies totally within V. Therefore point $z \in V$. Thus point z falls in both the lune and V which contradicts Lemma 2.

Proof of condition (ii). Let H_1 be the half-complement of K which contains bridge-endpoint $p \in V$, and let H_2 be the half-complement which contains bridge-endpoint $q \in W$. Polygon V is connected, and H_1 and H_2 are disjoint. Thus from condition i), V lies entirely in H_1 or it lies entirely in H_2. Now $p \in V$ and $p \in H_1$, therefore V lies entirely in H_1. Similarly, polygon W lies entirely in H_1 or in H_2. Now $q \in W$ and $q \in H_2$, thus W lies entirely in H_2. Q.E.D.

DEFINITION

A point p is *left* of a point q if the x-coordinate of point p is less than the x-coordinate of point q. Here, point q is *right* of p. Similarly, a point p is *above* a point q if the y-coordinate of point p is greater than the y-coordinate of point q. Here, point q is *below* point p.

DEFINITION

We will say that a point p is *above* a non-vertical line l if a vertical half-line drawn downward from point p intersects line l. Here, line l lies *below* point p.

Similarly, a point p is *below* a non-vertical line l if a vertical half-line drawn upward from point p intersects line l. Here, line l lies *above* point p.

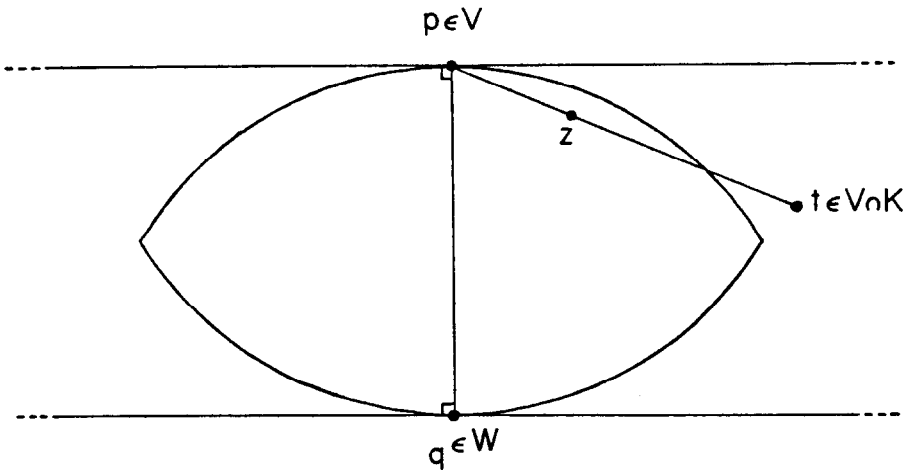

Fig. 1.

DEFINITION

Let H be a half-plane bounded by a vertical line. We will say that line l_1 lies *below* l_2 in H if every point in $l_1 \cap$ H is below line l_2. Alternately, line l_2 lies *above* line l_1 in H.

3. THE GENERAL PROCEDURE FOR FINDING THE NEAREST PAIR OF VERTICES

In this section, the algorithm to find the nearest distance between the vertices of two convex polygons V and W is presented as a ten-step procedure called TENSTEP.

Steps 6, 7, 8 and 9 are four subproblems. Thus the ten-step procedure is actually a linear-time decompositon of the "nearest vertices" problem into the four subproblems of steps 6, 7, 8 and 9.

PROCEDURE TENSTEP

1. Find the bridge between V and W using Edelsbrunner's algorithm. If the bridge is realized by two vertices then stop, because these two vertices also realize the shortest distance between vertices of V and W. Otherwise continue.

2. Let ω be the midpoint of this bridge. Translate the convex polygons so that ω is at the origin.

3. Rotate polygons V and W around origin ω to make the bridge horizontal. From Lemma 3, it follows that the horizontal bridge induces a vertical corridor between V and W. From here on in this paper, the term "corridor" will refer to this vertical corridor.

4. Find the vertices of V and W having the maximum y-coordinates. Call these vertices v_{max} and w_{max}. If the maximum y-coordinate of polygon V is realized by a horizontal edge, then assign the left vertex of this edge to v_{max}. If the maximum y-coordinate of polygon W is realized by a horizontal edge, then assign the right vertex of this edge to w_{max}.

5. Find the vertices of V and W having the minimum y-coordinates. Call these vertices v_{min} and w_{min}. If the minimum y-coordinate of polygon V is realized by a horizontal edge, then assign the left vertex of this edge to v_{min}. If the minimum y-coordinate of polygon W is realized by a horizontal edge, then assign the right vertex of this edge to w_{min}.

The vertices v_{max} and v_{min} delimit a left-chain and a right chain in polygon V. Similarly, vertices w_{max} and w_{min} delimit a left-chain and a right-chain in polygon W. We will call these chains V_L, V_R, W_L and W_R respectively.

6. Find the minimum distance d_1 between a vertex in V_L and a vertex in W_L.

7. Find the minimum distance d_2 between a vertex in V_R and a vertex in W_L.

8. Find the minimum distance d_3 between a vertex in V_R and a vertex in W_R.

9. Find the minimum distance d_4 between a vertex in V_L and a vertex in W_R.

10. The minimum distance between vertices of V and W is the smallest distance in the set $\{d_1,d_2,d_3,d_4\}$.

Steps 1, 2, 3, 4, and 5 above are computationally trivial and are easily done in $O(m + n)$ time. Thus we turn our attention to steps 6 through 9.

We note here that this decomposition is different from that used by Chin and Wang[3]. Although they also obtain four chains, their chains are not delimited by v_{max}, v_{min}, w_{max}, w_{min}, but by other vertices of V and W. The decomposition used here allows the problem to be solved using existing algorithms for the relative neighbourhood graph[7]. Thus the advantage of the method proposed here over that in [3] is that no new specialized code is needed.

4. PERFORMING STEPS 6, 7 AND 8 OF PROCEDURE TENSTEP

The subproblems in steps 6 and 7 are special cases of the following more general problem.

Let W_L be the left-chain of W as determined by steps 4 and 5 of TENSTEP; and let $\{b_1,b_2, \ldots ,b_\beta\}$ be the list of its vertices in ascending order. Let V_S be any connected subchain of V represented by a list $\{a_1,a_2, \ldots ,a_\alpha\}$ of its vertices in counterclockwise order. In linear time, find the minimum distance between a vertex in V_S and a vertex in W_L.

We will solve the more general subproblem. The solution will be readily applicable to the cases in steps 6, 7 and 8.

LEMMA 4

The perpendicular bisectors of two non-horizontal edges (b_i,b_{i+1}) and (b_{i+1},b_{i+2}) of W_L will intersect somewhere to the right of point b_{i+1}.

Proof. Let l_i be the perpendicular bisector of edge (b_i,b_{i+1}) and let h_i be the part of l_i which is on the right of edge (b_i,b_{i+1}). Angle (b_i,b_{i+1},b_{i+2}) is a right turn, so the intersection of l_i and l_{i+1} will be where h_i and h_{i+1} intersect. Now, at least one of h_i or h_{i+1} lies to the right of point b_{i+1} so l_i and l_{i+1} will intersect to the right of point b_{i+1}. Q.E.D.

COROLLARY 4.1

If edges (b_i,b_{i+1}) and (b_{i+1},b_{i+2}) are non-horizontal, then on the left of the corridor, bisector l_{i+1} lies above bisector l_i.

If we look at steps 4 and 5 of procedure TENSTEP, we note that the first or last edge of W_L might be horizontal. If l_1 is vertical, we will say that *no* points on the left of the corridor fall below l_1. If $l_{\beta-1}$ is vertical, we will say that *all* points on the left of the corridor fall below $l_{\beta-1}$.

COROLLARY 4.2

Let edges (b_i,b_{i+1}) and (b_{i+1},b_{i+2}) be any two consecutive edges in chain W_L. Every point z on the left of the corridor which lies above l_{i+1} will also lie above l_i. Every point z' on the left of the corridor which lies below l_i will also lie below l_{i+1}.

LEMMA 5

If among all vertices in W_L, b_k is the vertex nearest to $a_j \in V_S$, then a_j lies below l_k and above l_{k-1}. (See Fig. 2.)

Proof. $d(a_j,b_k) < d(a_j,b_{k-1})$; thus a_j lies above bisector l_{k-1}. Also, $d(a_j,b_k) < d(a_j,b_{k+1})$; thus a_j lies below bisector l_k. Q.E.D.

COROLLARY 5.1

Vertex a_j also lies above bisectors $l_{k-1},l_{k-3}, \ldots ,l_1$.

COROLLARY 5.2

Vertex a_j also lies below bisectors $l_{k+1},l_{k+2}, \ldots ,l_{\beta-1}$.

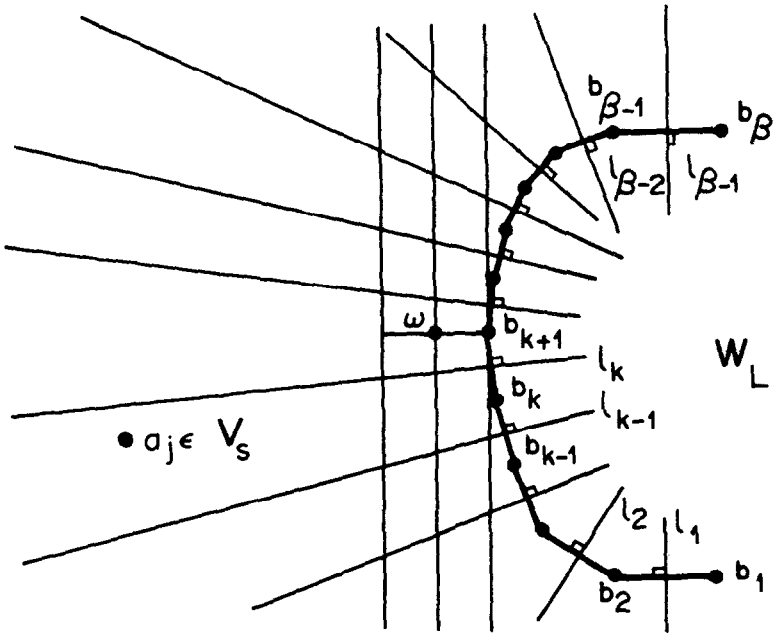

Fig. 2.

LEMMA 6

Let $b_k \in W_L$ be the vertex nearest to $a_j \in V_S$.

(i) For $i < k$, $d(a_j,b_{i+1}) < d(a_j,b_i)$.
(ii) For $i \geq k$, $d(a_j,b_i) < d(a_j,b_{i+1})$.

Proof.

(i) If $i < k$, then a_j is above l_i; thus $d(a_j,b_{i+1}) < d(a_j,b_i)$.
(ii) If $i \geq k$, then a_j is below l_i; thus $d(a_j,b_i) < d(a_j,b_{i+1})$. Q.E.D.

Lemma 6 in effect says: Given any $a_j \in V_S$, the function $d(a_j,b_k)$ for $k = \{1,2, \ldots \beta\}$ is unimodal, i.e., it has one local minimum. This is a key result which allows this problem to be solved in linear time and follows essentially from the fact that a_j lies outside of W. If a_j were in the interior of W, then $d(a_j,b_k)$ could be a multimodal function as was recently shown by Avis, Toussaint, and Bhattacharya[1] and Toussaint[10].

Since $d(a_j,b_k)$ is unimodal, the following function (called FINDMIN) will find the nearest vertex b_k for each vertex a_j in V_S.

FUNCTION FINDMIN

```
(* input: The list {a1,a2, . . . ,aα} of vertices in VS, and the list {b1,b2, . . . ,bβ} of
        vertices in WL.*)
1. j ← 1;
2. Find k such that d(a1,bk) is minimized;
3. dmin ← d(a1,bk);
4. REPEAT
5.    j ← j + 1;
6.    IF d(aj,bk-1) < d(aj,bk) THEN
7.       WHILE d(aj,bk-1) < d(aj,bk) DO
8.          k ← k − 1;
9.    ELSE
10.      IF d(aj,bk+1) < d(aj,bk) THEN
11.      WHILE d(aj,bk+1) < d(aj,bk) DO
```

12. $k \leftarrow k - 1$;
13. IF $d(a_j,b_k) < d_{\min}$ THEN $d_{\min} \leftarrow d(a_j,b_k)$;
14. UNTIL $j = \alpha$;
15. RETURN WITH $d_{\min}$ as the minimum distance between V_S and W_L;

END. (* of function FINDMIN *)

Procedure FINDMIN will work correctly because for every a_j in V_S, statements 6 thru 12 will minimize the unimodal function $d(a_j,b_k)$. Thus at the end of each REPEAT loop, variable k points to the vertex b_k in W_L nearest to a_j. Statement 13 assures that $d_{\min}$ at the end of FINDMIN's execution will hold the lowest of the minimum distances between pairs (a_j,b_k).

We will now show that if V_S has α vertices and W_L has β vertices, then function FINDMIN takes $O(\alpha + \beta)$ time.

For each execution of the REPEAT loop, let j_1 be the value of variable j just before step 5 is performed, and let j_2 be the value of j after step 5 is performed. Also, let k_0 be the value of k just before steps 6 thru 12 are performed.

LEMMA 7

Just before the execution of step 5, variable k_0 points to the vertex b_{k_0} nearest to a_{j_1}.

LEMMA 8

Just before the execution of step 5 of FINDMIN, these conditions hold:

(i) For all $i < k_0$, a_{j_1} lies above bisector l_i.
(ii) For all $i \geq k_0$, a_{j_1} lies below bisector l_i.

Now, step 8 of FINDMIN is performed only when $d(a_{j_2},b_{k-1}) < d(a_{j_2},b_k)$; i.e. only when bisector l_{k-1} is above a_{j_2}. The execution of step 8 implies that during the current iteration of the REPEAT loop, variable k is only decremented. (i.e. $k < k_0$.) Thus from condition i) of Lemma 8, a_{j_1} is above l_{k-1} which is in turn above a_{j_2}. In other words, step 8 is performed only when edge (a_{j_1},a_{j_2}) crosses from above l_{k-1} to below l_{k-1}.

For a given edge (a_{j_1},a_{j_2}) of V_S, the REPEAT loop is performed only once. If step 8 is performed, then during this one iteration of the REPEAT loop, variable k will move only downward. Thus if a given edge (a_{j_1},a_{j_2}) intersects a bisector l_{k-1}, then step 8 will be performed at most once for that intersection.

LEMMA 9

During the entire execution of function FINDMIN, step 8 is performed at most 2β times.

Proof. Let N be the number of intersections between edges in V_S and bisectors of edges in W_L. From the previous discussion, we know that step 8 is performed only when edge (a_{j_1},a_{j_2}) intersects bisector l_{k-1}. We also know that step 8 is performed at most once for each of these intersections. Thus the number of executions of step 8 is at most N. Now, V_S is a subchain of a convex polygon V. Thus a given bisector l_{k-1} will intersect V_S at most twice. Therefore N is at most twice the number of bisectors, which equals twice the number of edges in W_L. The number of executions of step 8 is at most N, which is in turn at most twice the number of edges in W_L. Q.E.D.

LEMMA 10

During the entire execution of function FINDMIN, step 12 is performed at most 2β times.

Proof. The proof is similar to that of Lemma 9.

LEMMA 11

The execution of function FINDMIN takes $O(\alpha + \beta)$ time.

Proof. Steps 1, 3 and 15 of FINDMIN each take constant time. Step 2 takes $O(\beta)$ time. Steps 4, 5, 6, 9, 10, 13, and 14 are each executed at most $(\alpha - 1)$ times. Steps 8 and 12 are each executed at most 2β times. Steps 7 and 11 are each executed at most $(2\beta + \alpha - 1)$ times. Q.E.D.

Since $\alpha < m$ and $\beta < n$, the previous lemmas imply the following result.

THEOREM 1

Function FINDMIN takes $O(m + n)$ time to execute.

In linear time FINDMIN determines the nearest distance between vertices in chain W_L and vertices in any subchain V_S of polygon V. Function FINDMIN can be used to perform steps 6 and 7 of procedure TENSTEP in section three of this paper.

Step 8 of TENSTEP can be performed by simply negating the x-coordinate of chains V_R and W_R, which reduces step 8 down to step 6.

5. PERFORMING STEP 9 OF PROCEDURE TENSTEP

In step 9 we wish to find the shortest distance between vertices of chains V_L and W_R in linear time. Chain V_L is a subchain of convex polygon V, so the vertices of V_L will determine a convex polygon. Similarly, chain W_R will also determine a convex polygon as illustrated in Fig. 3. Let U be the convex hull of $V_L \cup W_R$ illustrated in Fig. 4. For a set S of points, let $B(S)$ denote the boundary of the convex hull of S. Thus $B(V_L \cup W_R) = B(U)$.

Now, for any two chains V_L and W_R, 16 logical cases arise determined by whether $v_{max}, v_{min}, w_{max}$ and w_{min} each does or does not determine a vertex of U. The vertex of U having the maximum y-coordinate must be either v_{max} or w_{max}; and the vertex of U with the minimum y-coordinate must be either v_{min} or w_{min}. Thus only 9 of the 16 logical cases are possible. (These 9 cases are illustrated in Fig. 5.)

Recall that the horizontal bridge connects a point in V to a point in W. Thus v_{max} and w_{max} must both be above the bridge; and v_{min} and w_{min} must both be below the bridge. Because of this, there cannot be a case where v_{min} is above w_{max} or where v_{max} is below w_{min}.

In the rest of this paper, let us say that the set of points in a chain will consist of vertices, and not edge-points. One can perform the following linear-time decomposition on the problem of performing step 9 of procedure TENSTEP:

(i) Find $B(V_L \cup W_R)$ in linear time. (Toussaint[9] and Shamos[6] exhibit linear-time algorithms which find the convex hull of the union of two convex polygons.)

(ii) In linear time, determine which of the 9 possible cases holds.

(iii) Solve the problem for the appropriate case.

Now, case 9 can be reduced to case 5 in linear time by negating the x-coordinate of the vertices in V_L and W_R. Similarly, case 3 can be reduced to case 2, and case 8 can be reduced to case 6. Case 4 can be reduced to case 2 in linear time by negating the y-coordinate of the vertices in V_L and W_R; and case 7 can be reduced to case 2 by negating both the x and y-

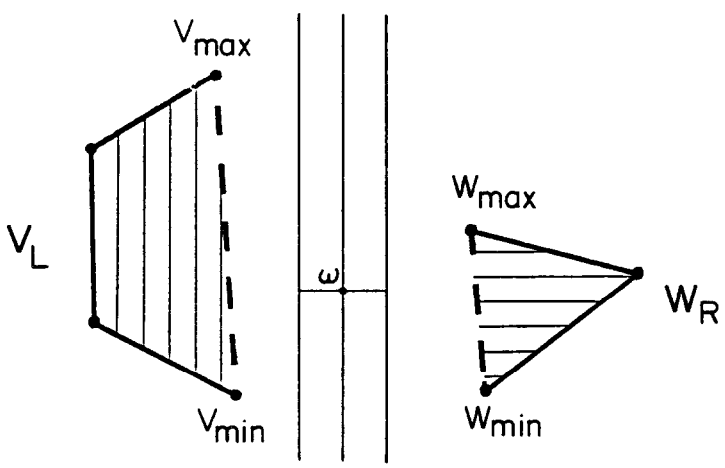

Fig. 3.

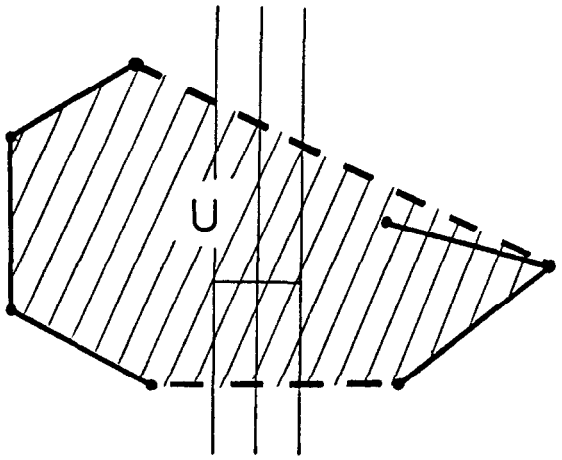

Fig. 4.

coordinates of the vertices. To solve step 9 of TENSTEP in linear time, we are thus left to solve for cases 1, 2, 5 and 6.

At this point, some more definitions and lemmas are introduced:

Let point s_i be an element of point set $S = \{s_1, s_2, \ldots, s_z\}$.

DEFINITION

If s_i has a greater x-coordinate and a greater y-coordinate than every other point in S, then we will say that point s_i 1-*dominates* the rest of set S. If s_i has a smaller x-coordinate and a larger y-coordinate than every other point in S, then point s_i 2-*dominates* the rest of set S. If point s_i has a smaller x-coordinate and a smaller y-coordinate than all the other points, then s_i 3-*dominates* the rest of set S. If s_i has a larger x-coordinate and a smaller y-coordinate than all the other points, then s_i 4-*dominates* the rest of S.

LEMMA 12

If point S_i k-dominates the rest of set S, and if some point p not in S k-dominates all of set S, then the point in set S closest to p is s_i.

Now, if point s_i k-dominates the rest of set S, and if every point in set $P = \{p_1, p_2, \ldots, p_z\}$ in turn k-dominates all of set S, then the shortest distance between sets S and P is realized by s_i and some point in P. The following function will thus find the shortest distance between sets S and P in linear time:

FUNCTION SCANSET(s_i,P);

1. $d_{\min} \leftarrow$ infinity;
2. FOR each point $p \in P$ DO
3. IF $d(s_i,p) < d_{\min}$ THEN $d_{\min} \leftarrow d(s_i,p)$;
4. RETURN WITH $d_{\min}$ as the shortest distance between sets S and P;

END. (* of function SCANSET *)

DEFINITION

Let $P = \{p_1, p_2, \ldots, p_z\}$ be a finite list of points. The *relative neighborhood graph* of P (denoted RNG(P)) is the list of edges $\{(p_{i_1},p_{j_1}),(p_{i_2},p_{j_2}), \ldots, (p_{i_n},p_{j_n})\}$ such that the lune of each edge (p_i,p_j) is empty.

In [12], it is shown that the number of edges in this graph cannot exceed $3z - 6$. Thus the number of edges in RNG(P) is always linear. If the points $P = \{p_1, p_2, \ldots, p_z\}$ represent the vertices of a convex polygon in order, then RNG(P) can be found in linear time using an algorithm by Supowit[7].

LEMMA 13

Let S_1 and S_2 be two finite sets of points. The shortest distance between sets S_1 and S_2 will be realized by an edge in RNG($S_1 \cup S_2$). This result has been proven in [11].

If C_1 and C_2 are disjoint subchains of convex polygon $P = \{p_1, p_2, \ldots, p_z\}$, then we can get the shortest distance between vertices of C_1 and C_2 in linear time by using the following algorithm by Toussaint and Bhattacharya[11]: we use Supowit's[7] linear algorithm to find RNG($C_1 \cup C_2$), and then we locate the shortest edge of RNG($C_1 \cup C_2$) that connects a vertex of C_i with a vertex of C_1. The following function, called USERNG, uses this technique:

FUNCTION USERNG(C_1,C_2)

1. Find RNG($C_i \cup C_2$) using Supowit's linear algorithm;
2. $d_{min} \leftarrow$ infinity;
3. FOR each edge (a,b) of RNG($C_1 \cup C_2$) DO
4. IF $a \in C_1$ AND $b \in C_2$
   OR $b \in C_1$ AND $a \in C_2$ THEN
5. IF $d(a,b) < d_{min}$ THEN $d_{min} \leftarrow d(a,b)$;
6. ENDFOR;
7. RETURN WITH d_{min} as the shortest distance between C_1 and C_2;

END. (* of function USERNG *)

In [11] this solution is applied to the more general problem where the vertices of P are arbitrarily colored with two colors, and we want to find the closest pair of vertices of opposite colors.

Let us now return to the problem of solving cases 1, 2, 5 and 6 in Fig. 5. If case 1 holds, then chains V_L and W_R are two subchains of a convex polygon. Thus we can find the shortest

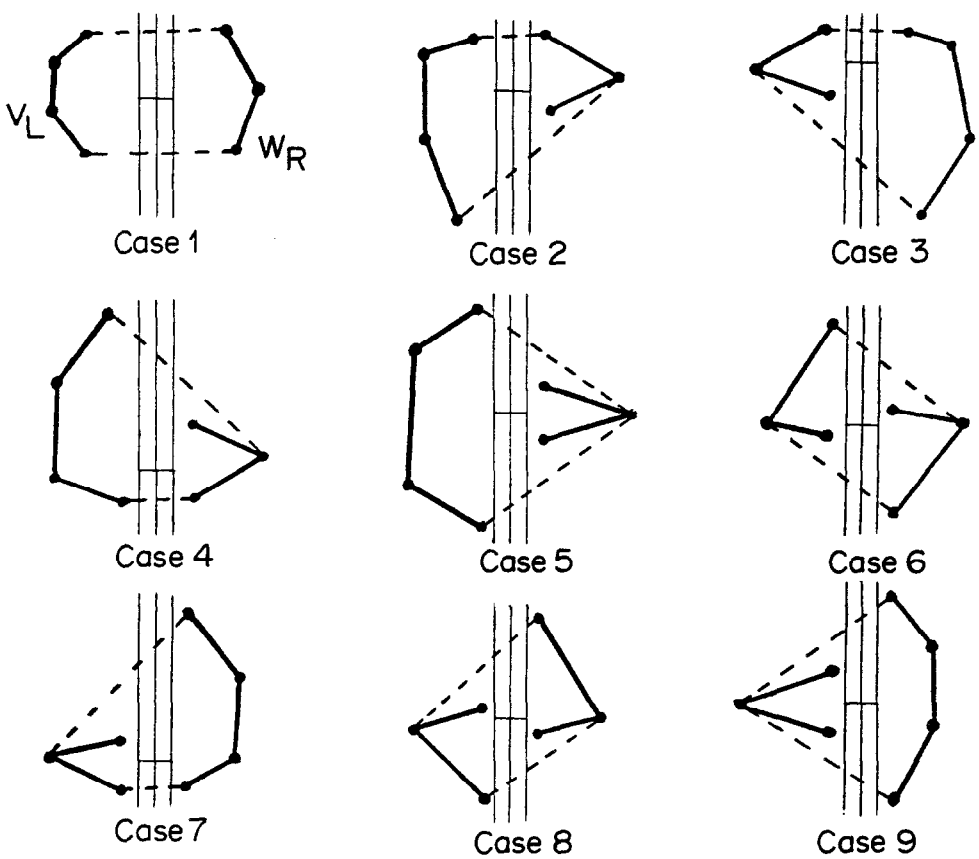

Fig. 5.

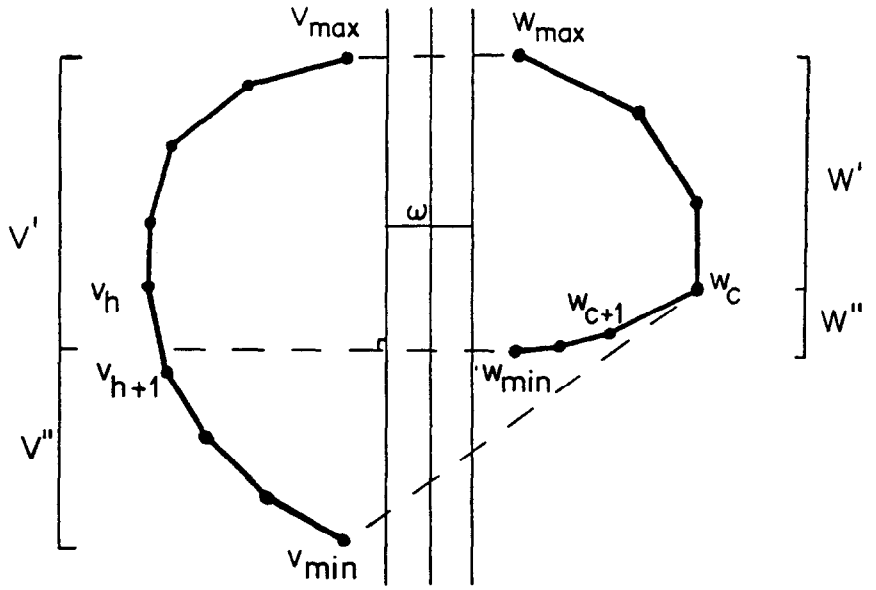

Fig. 6.

distance between chains V_L and W_R by applying USERNG to these two chains. Case 2 of Fig. 5 is illustrated in more detail in Fig. 6. One can see that USERNG cannot be applied for case 2 because not all the vertices of $V_L \cup W_R$ are in $B(V_L \cup W_R)$. However, chains V_L and W_R can be broken into subchains that can be processed in linear time. The following function, called CASE2, shows how this is done:

FUNCTION CASE2

(* Let W′ be the list of vertices in W_R which are on $B(V_L \cup W_R)$.
(In Fig. 6, W′ = $\{w_{max}, \ldots, w_c\}$.)
Let W″ be the list of vertices in W_R which are not on $B(V_L \cup W_R)$.
(In Fig. 6, W″ = $\{w_{c+1}, \ldots, w_{min}\}$.)
Let V′ be the list of vertices in V_L which are above w_{min}.
(In Fig. 6, V′ = $\{v_{max}, \ldots, v_h\}$.)
Let V″ be the list of vertices in V_L which are not above w_{min}.
(In Fig. 6, V″ = $\{v_{h+1}, \ldots, v_{min}\}$.)

1. d_{min} ← smallest of the three values returned by these functions:
   (i) USERNG(V_L,W′)
   (ii) USERNG(V′,W″)
   (iii) SCANSET(w_{min},V″)
2. RETURN WITH d_{min} as the shortest distance between V_L and W_R;

END. (* of function CASE2 *)

Function USERNG(V_L,W′) can be used because V_L and W′ are both on $B(V_L \cup W')$ and thus form a convex polygon. Similarly, chains V′ and W″ form a convex polygon. SCANSET(w_{min},V″) can be used because w_{min} 3-dominates the rest of W″ and every point in V″ 3-dominates W″. Thus FUNCTION CASE2 works correctly and takes $O(m + n)$ time.

Now consider case 5. Here we can decompose the problem into five subproblems by splitting V_L and W_R each into three chains as illustrated in Fig. 7 and using FUNCTION CASE5 defined below.

FUNCTION CASE5

(* Let W″ be the list of vertices in W_R that are on $B(V_L \cup W_R)$.
(In Fig. 7, W″ = $\{w_{c+1}, \ldots, w_f\}$.)

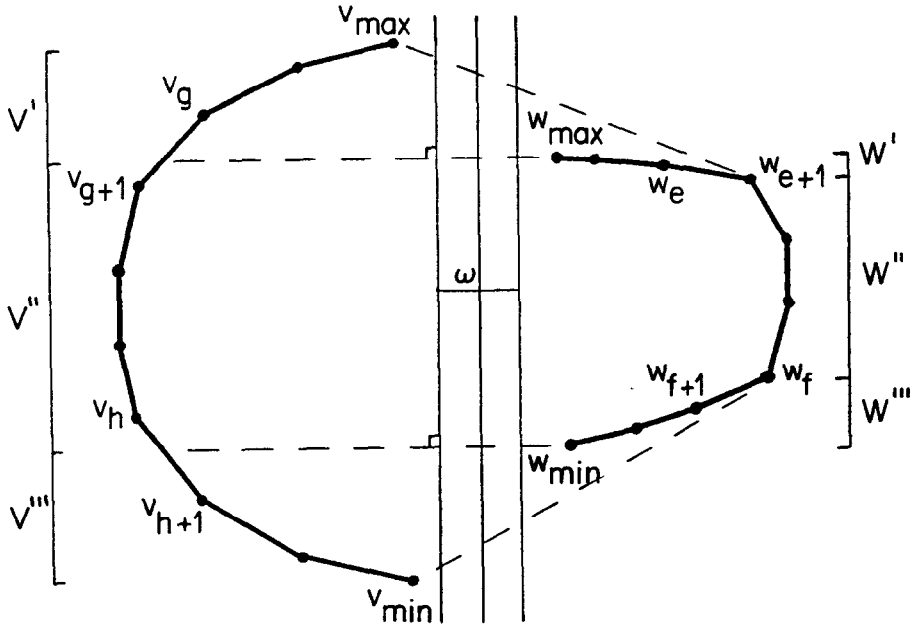

Fig. 7.

Let W′ be the vertices in W_R that are above W″.
(In Fig. 7, W′ = $\{w_{max}, \ldots, w_e\}$.)
Let W‴ be the vertices in W_R that are below W″.
(In Fig. 7, W‴ = $\{w_{f+1}, \ldots, w_{min}\}$.)
Let V″ be the vertices in V_L that are below w_{max} and above w_{min}.
(In Fig. 7, V″ = $\{v_{g+1}, \ldots, v_h\}$.)
Let V′ be the vertices in V_L that are not below w_{max}.
(In Fig. 7, V′ = $\{v_{max}, \ldots, v_g\}$.)
Let V‴ be the vertices in V_L that are not above w_{min}.
(In Fig. 7, V‴ = $\{v_{h+1}, \ldots, v_{min}\}$.) *)

1. d_{min} ← smallest of the five values returned by these functions:

   (i) SCANSET(w_{max},V′)
   (ii) USERNG(V″ ∪ V‴,W′)
   (iii) USERNG(V_L,W″)
   (iv) USERNG(V′ ∪ V″,W‴)
   (v) SCANSET(w_{min},V‴)
2. RETURN WITH d_{min} as the shortest distance between V_L and W_R;

END. (* of function CASES *)

The three uses of USERNG are justified because in each case a convex polygon is being processed. The first SCANSET will find the shortest distance between V′ and W′ because w_{max} 2-dominates the rest of W′ and every point in V′ 2-dominates W′. The second SCANSET will find the shortest distance between V‴ and W‴ because w_{min} 3-dominates the rest of W‴ and every point in V‴ 3-dominates W‴. Thus FUNCTION CASE5 will correctly find the minimum distance in case 5 in $O(m + n)$ time.

Next, consider case 6, which is illustrated in Fig. 8. Let v_c be the lowest vertex in $V_L \cap B(U)$, and let w_c be the highest vertex in $W_R \cap B(U)$. Let v_h be the lowest vertex of V_L which is above w_{max}, and let w_h be the highest vertex of W_R which is below v_{min}. Vertices v_c, w_c, v_h and w_h are illustrated in Fig. 8. Vertices v_c and v_h will divide V_L into an upper, middle and lower chain. Call these chains V′, V″ and V‴ respectively. Similarly, let W′, W″ and W‴ be the upper, middle and lower chains of W_R. Looking at case 6, one might at first glance

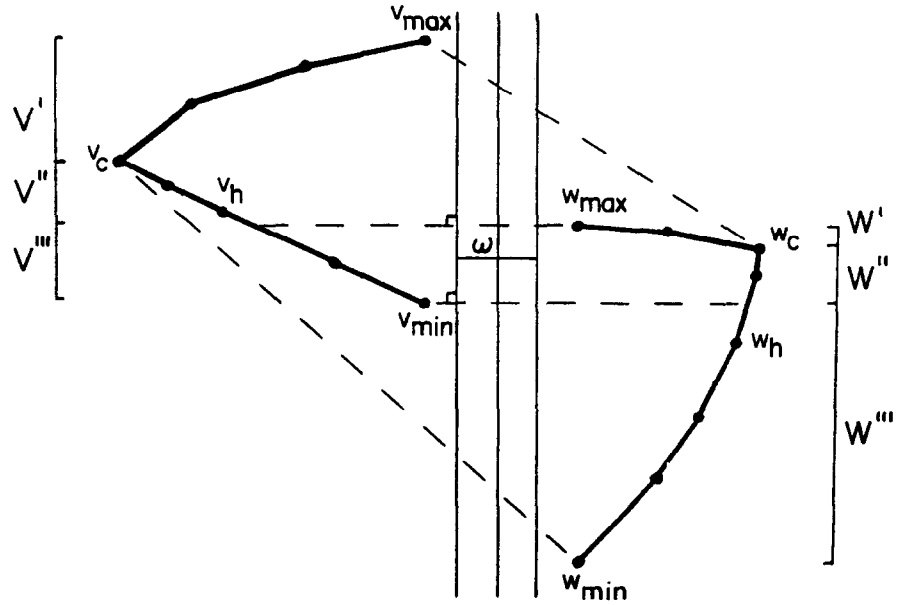

Fig. 8(a). (Case 6.1).

think that the following procedure will find the shortest distance between V_L and W_R:

$d_{min} \leftarrow$ smallest of the five values returned by these functions:

(i) SCANSET(w_{max},V')
(ii) USERNG(V'' ∪ V''',W')
(iii) USERNG(V_L,W'')
(iv) USERNG(V' ∪ V'',W''')
(v) SCANSET(v_{min},W''')

This will not work in all situations, however. If v_c is higher than v_h [as in Fig. 8(a)], then the third use of USERNG need not work because V'' is not in B(V' ∪ V'' ∪ W'''). A simple sequence of SCANSET's and USERNG's that works for all occurrences of case 6 was not

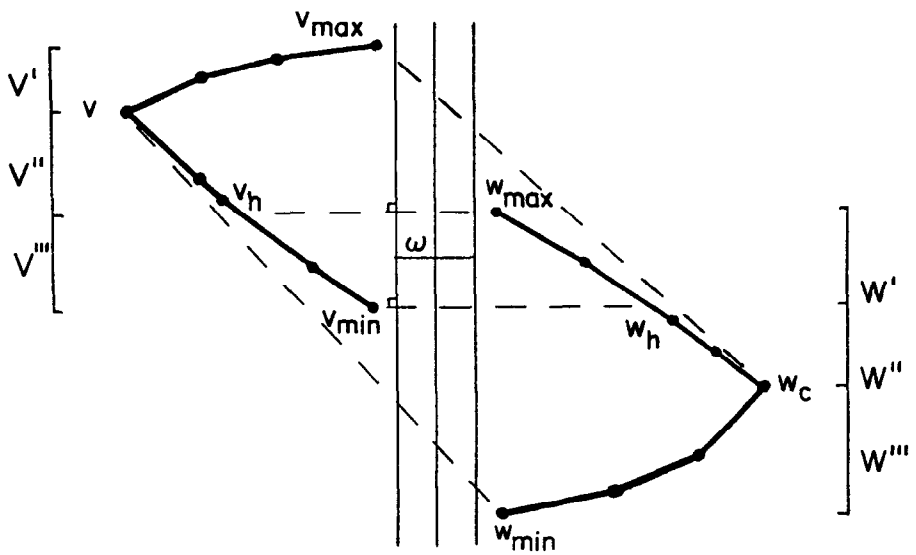

Fig. 8(b). (Case 6.2).

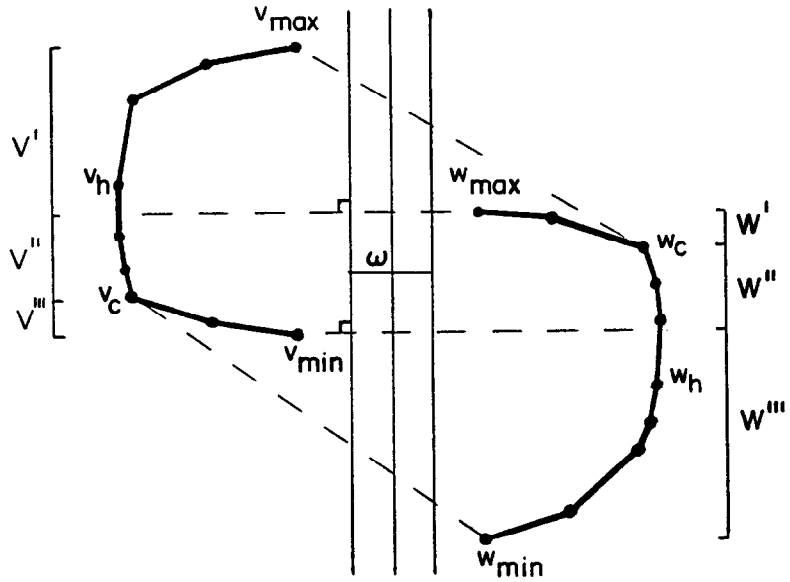

Fig. 8(c). (Case 6.3).

found. However, case 6 can be further decomposed into the following four subcases:

(6.1) v_c is above v_h and w_c is above w_h.
(6.2) v_c is above v_h and w_c is below w_h.
(6.3) v_c is below v_h and w_c is above w_h.
(6.4) v_c is below v_h and w_c is below w_h.

Each of these subcases is illustrated in Fig. 8. For case 6.1, the following statement will find the shortest distance between V_L and W_R:

$d_{min} \leftarrow$ smallest of the five values returned by these functions:

(i) SCANSET(w_{max}, V' ∪ V'')

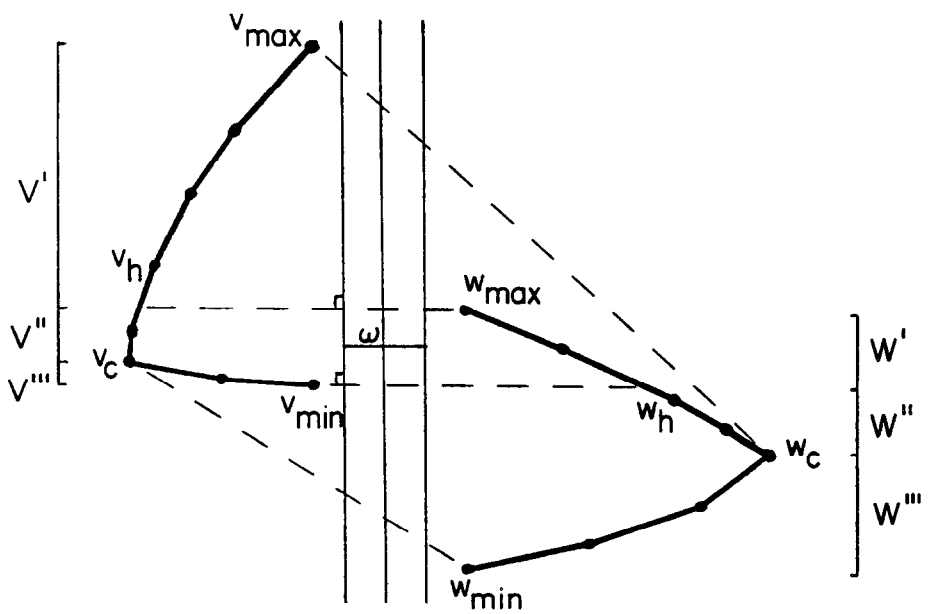

Fig. 8(d). (Case 6.4).

(ii) USERNG(V''',W')
(iii) USERNG(V_L,W'')
(iv) USERNG(V',W''')
(v) SCANSET(v_{min},W''')

The first SCANSET will find the shortest distance between V' ∪ V'' and W', and the second SCANSET will find the shortest distance between V'' ∪ V''' and W'''. For case 6.2, the following statement will find the shortest distance between V_L and W_R:

d_{min} ← smallest of the five values returned by these functions:

(i) SCANSET(w_{max},V' ∪ V'')
(ii) USERNG(V''',W')
(iii) SCANSET(w_h,V_L)
(iv) USERNG(V',W''')
(v) SCANSET(v_{min},W''')

The first SCANSET will return with the shortest distance between V' ∪ V'' and W'. The second SCANSET will return with the shortest distance between V_L and W'', because w_h 2-dominates the rest of W'', and every point in V_L 2-dominates all of W''. The third SCANSET will return with the shortest distance between V'' ∪ V''' and W'''. For case 6.3, the following statement will find the shortest distance between V_L and W_R:

d_{min} ← smallest of the five values returned by these functions:

(i) SCANSET(w_{max},V')
(ii) USERNG(V'' ∪ V''',W')
(iii) USERNG(V_L,W'')
(iv) USERNG(V' ∪ V'',W''')
(v) SCANSET(v_{min},W''')

The first SCANSET finds the shortest distance between V' and W' and the second SCANSET finds the minimum distance between V''' and W'''. Finally, for case 6.4, this last statement will find the shortest distance between V_L and W_R:

d_{min} ← smallest of the five values returned by these functions:

(i) SCANSET(w_{max},V')
(ii) USERNG(V'' ∪ V''',W')
(iii) SCANSET(w_h,V_L)
(iv) USERNG(V' ∪ V'',W''')
(v) SCANSET(v_{min},W''')

The first SCANSET finds the shortest distance between V' and W'. The second SCANSET finds the shortest distance between V_L and W'', because point w_h 2-dominates the rest of W'', and every point in V_L 2-dominates W''. The third SCANSET finds the minimum distance between V''' and W'''.

Finding v_c,v_h,w_c, and w_h and determining their relative positions by y coordinates can clearly be done in linear time. Thus, determining which of the four subcases applies can also be done in linear time. Since each subcase requires only linear time we may conclude that step 9 can be executed correctly in $O(m + n)$ time.

So far we have shown then that steps 1–9 of procedure TENSTEP can each be performed in linear time or faster. Step 10 obviously takes constant time and we have therefore established the following theorem.

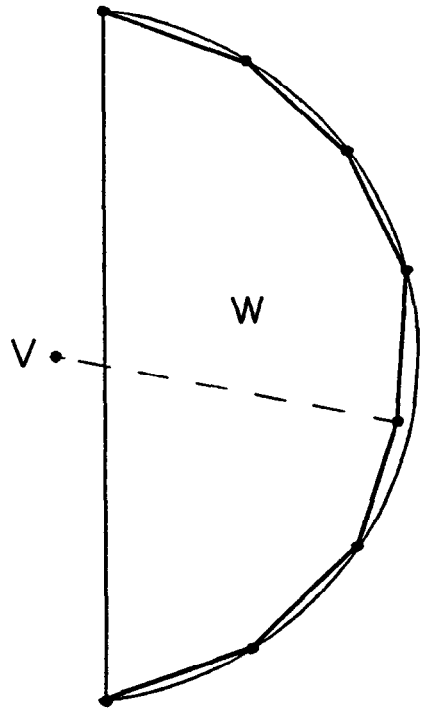

Fig. 9.

THEOREM 2

The minimum vertex-distance between V and W can be computed correctly in $O(m + n)$ time.

6. A LOWER BOUND ON THE COMPLEXITY OF THE PROBLEM

THEOREM 3

The complexity of finding the shortest distance between the vertices of two convex polygons is $\Omega(m + n)$.

Proof. Two polygons requiring a linear search are constructed and illustrated in Fig. 9. We place the vertices of polygon W on an arc of a circle, and we let polygon V consist of only one vertex v located at the center of the circle. If one of the vertices w_j of W is perturbed slightly toward v while maintaining the convexity of W, then the shortest vertex distance between V and W will be realized by v and the perturbed vertex. To correctly find the shortest distance, an algorithm will have to perform a linear scan through the vertices of W. Q.E.D.

7. CONCLUSION

In this paper we have exhibited linear upper and lower bounds on the problem of finding the closest pair of vertices between two linearly separable convex polygons. The algorithm establishing the upper bound is based extensively on the application of existing results on unimodality[1, 10], the relative neighborhood graph[7, 12] and distance between sets[11]. An immediate extension of this problem, not treated here, is the case of *intersecting* polygons. By combining the techniques used in this paper with existing results on polygon decomposition and convex polygon intersections it is possible to show that the general problem that includes intersecting polygons can also be solved in $O(m + n)$ time[13].

Several related problems remain open questions. One concerns the extension of the above problems to three dimensions. Another is the *all-nearest-vertices-between-sets* problem for convex polygons. Here, for every vertex v_i in V, we wish to find the closest vertex in W, and

for every vertex w_j in W, we wish to find the closest vertex in V. The chief difficulty of this problem lies in the cases where some closest vertices lie in the outer chains.

As mentioned in the introduction an alternate linear algorithm for this problem has been independently discovered by Chin and Wang[3]. Hence the lower bound proof presented here also establishes the optimality of their algorithm. Thus we observe two different ways of obtaining linear time algorithms. An interesting open problem from the practical point of view would be to compare actual running times of implementations of both of these algorithms.

REFERENCES

1. D. Avis, G. T. Toussaint and B. K. Bhattacharya, On the multimodality of distances in convex polygons. *Comp. Math. Applic.* **8**, (1982) 153–156.
2. F. Chin and C. A. Wang, "Optimal algorithms for the intersection and the minimum distance problems between planar polygons". technical report, University of Alberta (1982).
3. F. Chin and C. A. Wang, "Minimum vertex distance problem between two convex polygons", technical report, University of Alberta (1983).
4. H. Edelsbrunner, "On Computing the extreme distances between two convex polygons", technical report F96, Technical University of Gratz (1982).
5. J. T. Schwartz, Finding the minimum distance between two convex polygons. *Infon. Proc. Lett.* 168 (1981).
6. M. I. Shamos, "Computational geometry", Ph.D. thesis, Yale University (1978).
7. K. J. Supowit, The relative neighborhood graph, with an application to minimum spanning trees. *J.A.C.M.* **30**, 428 (1983).
8. G. T. Toussaint, Pattern recognition and geometrical complexity. *Proc. 5th International Conference on Pattern Recognition*, Miami Beach, 1324 (1980).
9. G. T. Toussaint, Computational geometric problems in pattern recognition. *Pattern Recognition Theory and Applications*, 73 (1981).
10. G. T. Toussaint, Complexity, convexity, and unimodality, *Int. J. Comp. Infon. Sci.* **13**, 197 (1984).
11. G. T. Toussaint and B. K. Bhattacharya, Optimal algorithms for computing the minimum distance between two finite planar sets. *Fifth International Congress of Cybernetics and Systems*, Mexico City (1981).
12. G. T. Toussaint, The relative neighborhood graph of a finite planar set. *Pattern Recognition* **12**, 261 (1980).
13. G. T. Toussaint, "An optimal algorithm for computing the minimum vertex distance between two crossing convex polygons", technical report no. SOCS-83.7, McGill University (1983).