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

Michael McKenna 
School of Computer Science 
McGill University, Montreal 
August 1983

A thesis submitted to the Faculty of Graduate Studies and Research 
in partial fulfillment of the requirements 
for the degree of Master of Science 

© Michael McKenna, 1983
FINDING THE MINIMUM VERTEX DISTANCE BETWEEN TWO DISJOINT CONVEX POLYGONS IN LINEAR TIME

Michael McKenna
School of Computer Science
McGill University, Montreal
August 1983

Abstract

Let V and W be two non-intersecting convex polygons whose vertices are defined by their cartesian coordinates in order. This thesis describes an original optimal algorithm that finds the minimum distance between the vertices of V and the vertices of W in linear time.

Résumé

Prenons deux polygones convexes V et W qui ne se chevauchent pas, où leurs sommets sont décrits en ordre par leurs coordonnées cartésiennes. Cette thèse montre un algorithme original et optimal qui trouve la distance minimum entre les sommets de V et les sommets de W dans un temps linéaire.

Index terms: algorithms, complexity, computational geometry, convex polygons, minimum distance, pattern recognition, relative neighborhood graph

CR categories: 3.36, 3.63, 5.25, 5.30, 5.5
ACKNOWLEDGEMENTS

I must thank my thesis supervisor Professor G.T. Toussaint for his infinite patience regarding the time and the many drafts it took to produce this thesis.

I also thank the Natural Sciences and Engineering Research Council of Canada for having generously supported this effort with a postgraduate scholarship.
1. Par la présente, l'auteur accorde à l'université McGill l'autorisation de mettre cette thèse à la disposition des lecteurs dans une bibliothèque de McGill ou une autre bibliothèque, soit sous sa forme actuelle, soit sous forme d'une reproduction. L'auteur détient cependant les autres droits de publications. Il est entendu, par ailleurs, que ni la thèse, ni les longs extraits de cette thèse ne pourront être imprimés ou reproduits par d'autres moyens sans l'autorisation écrite de l'auteur.

2. La présente autorisation entre en vigueur à la date indiquée ci-dessus à moins que le Comité exécutif du conseil n'ait voté de différer cette date. Dans ce cas, la date différée sera le 

Signature de l'auteur

Adresse permanente:

Signature du doyen si une date figure à l'alinea 2.

(English on reverse)
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACTS</td>
<td>1</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>2</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>3</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>5</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>6</td>
</tr>
<tr>
<td>Survey</td>
<td>6</td>
</tr>
<tr>
<td>Nearest vertices problem</td>
<td>6</td>
</tr>
<tr>
<td>Nearest points problem</td>
<td>6</td>
</tr>
<tr>
<td>2. PRELIMINARY DEFINITIONS AND RESULTS</td>
<td>9</td>
</tr>
<tr>
<td>Line segment</td>
<td>9</td>
</tr>
<tr>
<td>Bridge</td>
<td>9</td>
</tr>
<tr>
<td>Lemma 1</td>
<td>9</td>
</tr>
<tr>
<td>Corridor</td>
<td>9</td>
</tr>
<tr>
<td>Half-complement</td>
<td>9</td>
</tr>
<tr>
<td>Indecent</td>
<td>9</td>
</tr>
<tr>
<td>Lune</td>
<td>10</td>
</tr>
<tr>
<td>Lemma 2</td>
<td>10</td>
</tr>
<tr>
<td>Lemma 3</td>
<td>10</td>
</tr>
<tr>
<td>Lemma 4</td>
<td>10</td>
</tr>
<tr>
<td>Left, right, above, below</td>
<td>12</td>
</tr>
<tr>
<td>Relative neighborhood graph</td>
<td>13</td>
</tr>
<tr>
<td>Line of support</td>
<td>13</td>
</tr>
<tr>
<td>Diameter</td>
<td>13</td>
</tr>
<tr>
<td>3. SUPOWIT’S LINEAR RNG ALGORITHM FOR CONVEX POLYGONS</td>
<td>15</td>
</tr>
<tr>
<td>Lemma 5</td>
<td>15</td>
</tr>
<tr>
<td>Corollary 5.1</td>
<td>18</td>
</tr>
<tr>
<td>Lemma 6</td>
<td>18</td>
</tr>
<tr>
<td>Decomposition into subproblems S.1, S.2, S.3, S.4</td>
<td>20</td>
</tr>
<tr>
<td>3.1 Subproblem S.2, Phase one</td>
<td>20</td>
</tr>
<tr>
<td>Procedure PHASE1</td>
<td>23</td>
</tr>
<tr>
<td>Lemma 7</td>
<td>23</td>
</tr>
<tr>
<td>Corollary 7.1</td>
<td>24</td>
</tr>
<tr>
<td>Lemma 8</td>
<td>24</td>
</tr>
<tr>
<td>Lemma 9</td>
<td>24</td>
</tr>
<tr>
<td>3.2 Subproblem S.2, Phase two</td>
<td>25</td>
</tr>
<tr>
<td>Lemma 10</td>
<td>25</td>
</tr>
<tr>
<td>Procedure PHASE2</td>
<td>27</td>
</tr>
<tr>
<td>3.3 Subproblem S.3</td>
<td>28</td>
</tr>
<tr>
<td>Ray</td>
<td>28</td>
</tr>
<tr>
<td>Lemma 11</td>
<td>28</td>
</tr>
<tr>
<td>Corollary 11.1</td>
<td>30</td>
</tr>
<tr>
<td>Procedure KO</td>
<td>30</td>
</tr>
<tr>
<td>Theorem 1</td>
<td>31</td>
</tr>
</tbody>
</table>
### 4. THE GENERAL PROCEDURE FOR FINDING THE NEAREST PAIR OF VERTICES

**Procedure TENSTEP**

### 5. PERFORMING STEPS 6, 7 AND 8 OF PROCEDURE TENSTEP

- Lemma 12
- Corollary 12.1
- Corollary 12.2
- Lemma 13
- Corollary 13.1
- Corollary 13.2
- Lemma 14
- Function FINDMIN
- Lemma 15
- Lemma 16
- Lemma 17
- Lemma 18
- Lemma 19
- Theorem 2

### 6. PERFORMING STEP 9 OF PROCEDURE TENSTEP

- Cases 1 through 9
- Reduction to cases 1, 2, 5 and 6
- k-dominance
- Lemma 20
- Function SCANSET
- Lemma 21
- Function USERNG

#### 6.1 Cases 1, 2 and 5 of step 9

- Case 1
- Function CASE2
- Function CASE5

#### 6.2 Case 6 of step 9

- Decomposition into subcases 6.1, 6.2, 6.3 and 6.4
- Subcase 6.1
- Subcase 6.2
- Subcase 6.3
- Subcase 6.4
- Theorem 3

### 7. A LOWER BOUND ON THE COMPLEXITY OF THE PROBLEM

- Theorem 4

### 8. CONCLUSION

- The case of intersecting polygons
- Open problems
- All-nearest-vertices-between-sets problem

### REFERENCES
<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>16</td>
<td>39</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>17</td>
<td>43</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>18</td>
<td>43</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>19</td>
<td>47</td>
</tr>
<tr>
<td>5a</td>
<td>16</td>
<td>20</td>
<td>47</td>
</tr>
<tr>
<td>5b</td>
<td>16</td>
<td>21</td>
<td>48</td>
</tr>
<tr>
<td>6</td>
<td>19</td>
<td>22a</td>
<td>50</td>
</tr>
<tr>
<td>7</td>
<td>19</td>
<td>22b</td>
<td>50</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>23</td>
<td>53</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
<td>24</td>
<td>56</td>
</tr>
<tr>
<td>10</td>
<td>29</td>
<td>25a</td>
<td>58</td>
</tr>
<tr>
<td>11</td>
<td>29</td>
<td>25b</td>
<td>59</td>
</tr>
<tr>
<td>12</td>
<td>33</td>
<td>25c</td>
<td>60</td>
</tr>
<tr>
<td>13</td>
<td>36</td>
<td>25d</td>
<td>61</td>
</tr>
<tr>
<td>14</td>
<td>37</td>
<td>26</td>
<td>66</td>
</tr>
<tr>
<td>15</td>
<td>39</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

Let $V$ be a convex polygon described by a counterclockwise 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 thesis describes an original optimal $O(m+n)$ 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: $d(v_i, w_j) \leq d(v_k, w_l)$.

Recently, Chin and Wang [3] have independently found another algorithm for this same problem. Their solution is similar to the one in this paper to the extent that they perform an initial decomposition of the problem into four subproblems. However, their decomposition and solutions to the subproblems differ considerably from ours. In particular, the approach in this paper is based on the application of existing results for computing the relative neighborhood graph of a convex polygon [7].

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. Figure 1 shows how the solution for the "nearest vertices" problem can differ from the solution for the "nearest points" problem. The algorithm in this paper will find that $v_2$ and $w_2$ are the nearest vertices. However, Edelsbrunner's algorithm will find that vertex $v_1$ and edge-point $e$ are the nearest points.

This thesis is divided into eight chapters. The following chapter presents some preliminary definitions and lemmas. The third chapter describes Supowit's [7] linear algorithm for finding the relative neighborhood graph of a convex polygon. The fourth chapter outlines a linear-time ten-step procedure for finding the nearest pair of vertices between $V$ and $W$. Chapter five shows how steps six, seven and eight of the ten-step procedure can be performed in linear time; and chapter six shows how step nine can be performed in linear time. Chapter seven presents a proof by Toussaint [14] that the nearest-vertices problem has an $\Omega(m + n)$ lower bound. Finally, chapter eight presents some concluding remarks and open problems.
FIGURE 1
CHAPTER 2
PRELIMINARY DEFINITIONS AND RESULTS

**Definition:** A line segment \((p,q)\) (or simply a segment \((p,q)\)) is a finite line with endpoints \(p\) and \(q\).

**Definition:** The bridge between convex polygons \(V\) and \(W\) is the segment whose endpoints are the nearest points found by Edelsbrunner's algorithm. In figure 1, segment \((v_1,e)\) is the bridge between convex polygons \(V\) and \(W\). If the shortest distance between the polygons is realized by a pair of parallel edges then the bridge may be chosen from an infinite number of segments.

**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 vertices of \(V\) and \(W\) is also realized by this pair of vertices.

**Proof:** The distance between any pair of points in \(V\) and \(W\) will not be shorter than the bridge; thus the distance between any pair of vertices will not be shorter either. Q.E.D.

**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 of 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 \(d(q,r) < d(q,p)\).

In figure 2, segment \((p,q)\) induces corridor \(K\), which is bounded by lines \(l_1\) and \(l_2\); and lune \((p,q)\) is the shaded region. Note that the lune consists of the intersection of the interior of 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 \(a, b\) and \(c\) be points in the plane. If \(\angle(a,b,c) > 90^\circ\), then point \(b\) is in lune \((a,c)\).

**Proof:** \(\angle(a,b,c) > \angle(b,e,8)\) and \(\angle(a,b,e) > \angle(e,a,b)\); thus \(d(a,e) > d(a,b)\) and \(d(a,c) > d(b,c)\). Q.E.D.

**Lemma 3:** 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 \(s \in W\) in \(L\).

**Proof of condition i):** If point \(s \in V\) is in \(L\), then \(d(s,q) < d(p,q)\). This contradicts our assumption that \(p\) and \(q\) are the endpoints of the bridge between \(V\) and \(W\). The proof of condition ii) is similar. Q.E.D.

**Lemma 4:** The bridge between convex polygons \(V\) and \(W\) induces a corridor \(K\) falling between the polygons \(V\) and \(W\). In other words,

i) \(V \cap K = \emptyset\) and \(W \cap K = \emptyset\), and

ii) \(V\) and \(W\) are in separate half-complements of \(K\).
PROOF (by contradiction) of condition i): (See figure 3.) Let p and q be
the endpoints of the bridge. From lemma 3, 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 \emptyset. This means that there is some point
t \in V \cap K. Point t is in K, thus on segment (p,t) there is 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 the both the
lune and V which contradicts lemma 3.

PROOF of condition ii): Let H_1 be the half-complement of K which con­
tains 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.

**Definition:** Let \( H \) be a half-plane bounded by a vertical line. We will say that line \( I_1 \) lies below line \( I_2 \) in \( H \) if every point in \( I_1 \cap H \) is below line \( I_2 \). Alternately, line \( I_2 \) lies above line \( I_1 \) in \( H \).

**Definition:** Let \( P = \{p_1, p_2, \ldots, p_z\} \) be a finite list of points. The relative neighborhood graph of \( P \) (denoted \( \text{RNG}(P) \)) 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 \( (p_k, p_{j_k}) \) contains no point of set \( P \). In [12] it is shown that the number of edges in this graph cannot exceed \( 3z-6 \). Thus the number of edges in \( \text{RNG}(P) \) is always linear. If the points \( P = \{p_1, p_2, \ldots, p_z\} \) represent the vertices of a convex polygon in order, then \( \text{RNG}(P) \) can be found in linear time using an algorithm by Supowit [7], which is described in the next chapter of this paper.

**Definition:** A line of support of point set \( P \) is a line containing a point of \( P \) such that every point of \( P \) lies to one side of the line.

**Definition:** Let \( P = \{p_1, p_2, \ldots, p_z\} \) be any set of points in the plane. The diameter of set \( P \) is the maximum realizable distance between any two points in \( P \). In other words, the diameter equals \( \max_{1 \leq i, j \leq z} d(p_i, p_j) \). Shamos [6] shows how to find the diameter of a convex polygon in linear time by rotating two parallel lines of support around the polygon (as shown in figure 4) until the parallel lines are farthest apart. In figure 4, the diameter of \( P \) is realized by segment \( p_{\text{diam}} \).
FIGURE 4
CHAPTER 3
SUPOWIT'S LINEAR RNG ALGORITHM FOR CONVEX POLYGONS

In this chapter, Supowit's [7] linear algorithm for finding the relative neighborhood graph of convex polygon \( V = \{v_1, v_2, \ldots, v_n\} \) is described. The description in these pages follows his description except for lemma 5 which is more directly suited to our presentation. Although none of the results in this chapter are new, they are included because they are necessary for a thorough understanding of the later contributions of this thesis. Furthermore the description of Supowit's results given here is felt to be clearer than the description due to Supowit [7].

**Lemma 5:** Let \( a, b, c, a', \) and \( b' \) be points in the plane such that segment \((a',b')\) intersects segments \((a,c)\) and \((b,c)\). If point \( c \) is in lune \((a,b)\) and points \( a' \) and \( b' \) are not in lune \((a,b)\), then point \( c \) is in lune \((a',b')\).

**Proof:** (See figures 5a and 5b.) If point \( c \) is on edge \((a',b')\), then \( c \) is trivially in lune \((a',b')\); thus let us assume that \( c \) is not on \((a',b')\).

Without loss of generality, let us now assume that segment \((a,b)\) is horizontal, that \( c \) is above \((a,b)\), that \( a \) is left of \( b \) and \( a' \) is left of \( b' \), and that \( a' \) is not below \( b' \). Let us further assume that \( c \) is at the plane origin.

Point \( c \) is in lune \((a,b)\); therefore \( a \) is to the left of \( c \) and \( c \) is to the left of \( b \). Segment \((a',b')\) crosses segments \((a,c)\) and \((c,b)\); thus segment \((a',b')\) crosses the y-axis. We now consider three cases:
CASE 1) Both $a'$ and $b'$ are above the $x$-axis. This case is impossible since $(a', b')$ intersects $(a, c)$ and $(b, c)$.

CASE 2) Point $a'$ is above the $x$-axis and $b'$ is below the $x$-axis. (See figure 5a.) Edge $(a', b')$ intersects edges $(a, c)$ and $(b, c)$ so $(a', b')$ must be below $c$. The angle formed by the $x$-axis and the $y$-axis is $90^\circ$, so angle $(a', c, b')$ is greater than $90^\circ$. Thus $c$ is in lune $(a', b')$.

CASE 3) Both $a'$ and $b'$ lie below the $x$-axis. (See figure 5b.) Let $g$ be the point on the "left-arc" of lune $(a, b)$ such that edge $(g, b)$ is parallel to $(a', b')$, and let $l_{bis}$ be the line that bisects angle $(a, b, g)$. All points above $l_{bis}$ are closer to $g$ than to point $a$, so $d(c, a) > d(c, g)$. Points $a'$ and $b'$ are not in lune $(a, b)$, thus segment $(a', b')$ must intersect segments $(c, g)$ and $(c, b)$. Let $h_1$ and $h_2$ be the points where $(a', b')$ intersects segments $(c, g)$ and $(c, b)$ respectively.

Now, $c$ is in lune $(a, b)$, thus

$$d(a, b) > d(c, a) \quad \text{and} \quad d(a, b) > d(c, b).$$

We have $d(g, b) = d(a, b)$ and $d(c, a) > d(c, g)$, so

$$d(g, b) > d(c, g) \quad \text{and} \quad d(g, b) > d(c, b).$$

Triangles $(g, b, c)$ and $(h_1, h_2, c)$ are similar, thus

$$d(h_1, h_2) > d(c, h_1) \quad \text{and} \quad d(h_1, h_2) > d(c, h_2).$$

If we use the triangle inequality, then

$$d(a', b') > d(a', h_2) = d(a', h_1) + d(h_1, h_2) > d(a', h_1) + d(h_1, c) \geq d(a', c)$$

and

$$d(a', b') > d(h_1, b') = d(h_1, h_2) + d(h_2, b') > d(c, h_2) + d(h_2, b') \geq d(c, b').$$

Thus $c$ is in lune $(a', b')$.

Q.E.D. LEMMA 5.
COROLLARY 5.1: Let segment \((a',b')\) intersect segments \((a,c)\) and \((b,c)\). If point \(c\) is not in lune \((a',b')\), then \(c\) is not in lune \((a,b)\), or \(a'\) or \(b'\) is in lune \((a,b)\).

Now let the diameter of \(V\) be realized by vertices \(v_0\) and \(v_2\), and let \(l_{\text{diam}}\) be the line passing through \(v_0\) and \(v_2\). Let \(v_1\) be the vertices of \(V\) having the largest perpendicular distance from \(l_{\text{diam}}\) on each side of \(l_{\text{diam}}\). Let vertices \(v_1\) and \(v_3\) be placed so that the vertices \(v_0, v_1, v_2, v_3\) are in counterclockwise order (as in figure 6). Now for \(k=0,1,2,3\) let subchain \(V_k\) of polygon \(V\) consist of the counterclockwise list of vertices \(\{v_{i_k}, v_{i_k+1}, \ldots, v_{i_k+2}, v_{i_k+3}\}\). (\(k+1\) is taken modulo 4.) Chains \(V_0, V_1, V_2, V_3\) are shown in figure 6. The following lemma justifies the decomposition of polygon \(V\) into the four chains:

**LEMMA 6:** For \(k \in \{0,1,2,3\}\), if an edge of \(\text{RNG}(V)\) connects a vertex \(v_a\) of chain \(V_k\) to another vertex \(v_b\) of \(V_k\), then \(v_a\) and \(v_b\) are consecutive vertices of \(V_k\).

**PROOF:** Without loss of generality let \(k = 0\). Let us assume without loss of generality that the diameter \(l_{\text{diam}}\) of \(V\) lies on the \(x\)-axis and that vertex \(v_1\) lies on the positive \(y\)-axis. Let \(\omega\) be the origin. (See figure 7.)

Let us now assume that there is an edge \((v_a, v_b)\) of \(\text{RNG}(V)\) where \(v_a\) and \(v_b\) are non-consecutive. Thus there must be a vertex \(v'\) falling between \(v_a\) and \(v_b\) in the chain. By necessity angles \((\omega, v_0, v')\) and...
(ω, vi, v') are both less than 90°. Thus angle (v_i, v, v'_{i+1}) must be greater than 90°. It then follows that angle (v_a, v', v_b) is also greater than 90°. From lemma 2, vertex v' must then be in lune (v_a, v_b). This contradicts our assumption that edge (v_a, v_b) is in RNG(V). Q.E.D.

To find RNG(V), it will thus be sufficient to solve the following four subproblems:

S.1) Find vertices v_{i_0}, v_{i_1}, v_{i_2} and v_{i_3}.
S.2) Find the edges of RNG(V) which cross from V_0 ∪ V_1 to V_2 ∪ V_3.
S.3) Find the edges of RNG(V) which cross from V_1 ∪ V_2 to V_3 ∪ V_0.
S.4) Find the edges of RNG(V) which are on the boundary of polygon V.

Subproblem S.1 is solved trivially. Subproblems S.2 and S.3 are identical. We will thus look at subproblems S.2 and S.4.

3.1. Subproblem S.2, Phase one

Let T be the set of edges in RNG(V) which cross from V_0 ∪ V_1 to V_2 ∪ V_3. This subproblem is solved in two phases. The first phase constructs a set E of edges such that T ⊆ E, and the second phase deletes the edges of E that are not in T. Before we look at phase one, let us make a few observations.

Recall that V_0 ∪ V_1 = {v_{i_0}, ..., v_{i_1}, ..., v_{i_2-1}} and that V_2 ∪ V_3 = {v_{i_2}, ..., v_{i_3}, ..., v_{i_0-2}}. (These are counterclockwise lists.) Now in the quadrilateral (v_{i_0-1}, v_{i_0}, v_{i_2-1}, v_{i_2}), there is at least one angle ≥ 90°. For illustration assume that angle (v_{i_0}, v_{i_2-1}, v_{i_2}) ≥ 90°. (See figure 8.) For every vertex v_j in {v_{i_0}, v_{i_0+1}, ..., v_{i_2-2}}, angle
Thus from lemma 2, edge \((v_{i_2}, v_j)\) will not be in \(\text{RNG}(v)\). Therefore the only possible edge of set \(T\) which can join vertex \(v_{i_2}\) to subchain \(V_0 V_1\) is edge \((v_{i_2}, v_{i_2-1})\). If edge \((v_{i_2}, v_{i_2-1})\) is stored in set \(E\) as a candidate for membership in set \(T\), then we can discard vertex \(v_{i_2}\) because no other edges at \(v_{i_2}\) need to be considered. We could then replace \(v_{i_2}\) with \(v_{i_2+1}\) in the quadrilateral and look for another angle greater than 90°. (See figure 8.) To create set \(E\) in phase one we could thus perform the following procedure: (Comments are delimited by "(*) and "*".)
PROCEDURE PHASE1;

1. E ← ∅;

(* Let $v_p, v_q, v_r,$ and $v_s$ be the vertices of the quadrilateral. *)

Initialize the quadrilateral:

2. $p ← i_0 - 1; q ← i_0; r ← i_2 - 1; s ← i_2;

3. REPEAT

(* Find an angle in the quadrilateral which is $\geq 90^\circ$. *)

4. IF $\angle (v_p, v_q, v_r) \geq 90^\circ$ THEN

5. BEGIN Store edge $(v_p, v_q)$ in $E$; $p ← p - 1$; END;

6. ELSE IF $\angle (v_q, v_r, v_s) \geq 90^\circ$ THEN

7. BEGIN Store edge $(v_q, v_s)$ in $E$; $s ← s + 1$; END;

8. ELSE IF $\angle (v_r, v_s, v_p) \geq 90^\circ$ THEN

9. BEGIN Store edge $(v_r, v_s)$ in $E$; $r ← r - 1$; END;

10. ELSE (* $\angle (v_s, v_p, v_q) \geq 90^\circ$ *)

11. BEGIN Store edge $(v_s, v_q)$ in $E$; $q ← q + 1$; END;

12. UNTIL $p = s$ OR $q = r$;

13. IF $p = s$ THEN

14. FOR every vertex $v'$ remaining in $V_0 \cup V_1$ DO

15. Store in $E$ the edge $(v, v')$;

16. ELSE (* $q = r$ *)

17. FOR every vertex $v'$ remaining in $V_2 \cup V_3$ DO

18. Store in $E$ the edge $(v, v')$;

19. Discard the remaining vertices;

20. OUTPUT set $E$ as a superset of set $T$;

END. (* of procedure PHASE1 *)

**Lemma 7:** During the execution of procedure PHASE1, a vertex $v_j$ is
discarded only when all the edges of set $T$ that contain $v_j$ have been stored in set $E$.

**Proof by induction:** Let $v_j$ be the vertex discarded at the $j$th iteration of the REPEAT loop. At the first iteration of the REPEAT loop, the one edge containing $v_1$ which can possibly be in $T$ is stored in $E$. Then $v_1$ is discarded. At the $j$th iteration of the REPEAT loop, the edges of $T$ connecting $v_j$ to previously discarded vertices have been stored in $E$; and the one remaining edge containing $v_j$ which can possibly be in $T$ is stored in $E$. Then $v_j$ is discarded. After the last iteration of the REPEAT loop, all the remaining edges which can connect $V_0 \cup V_1$ to $V_2 \cup V_3$ are stored in $E$. Q.E.D.

**Corollary 7.1:** At the end of execution of procedure PHASE1, $T \subseteq E$.

Figure 9 shows an example of the edges in set $E$ after the execution of PHASE1.

**Lemma 8:** Procedure PHASE1 takes linear time to execute.

**Proof:** Procedure PHASE1 takes constant time for each discarded vertex of polygon $V$. Q.E.D.

**Lemma 9:** The edges of $E$ do not intersect except at endpoints.

**Proof:** Consider the evolution of edge $(v_p, v_q)$ during the execution of PHASE1. At the beginning of execution, the vertices in the circular list $V = \{v, v_{q+1}, \ldots, v_r, v_s, v_{s+1}, \ldots, v_p\}$ form a convex polygon; thus the later versions of $(v_p, v_q)$ will lie completely to one side of the beginning version of $(v_p, v_q)$. At the $j$th iteration of the REPEAT
After the last iteration of the REPEAT loop, all the remaining possible edges of T meet at a common vertex \( v_p \) or \( v_q \). Thus none of these remaining edges will cross each other. (In figure 9, the remaining edges all meet at \( v_p \).) The very last version of edge \((v_p, v_q)\) separates the remaining edges from the older versions of \((v_p, v_q)\); thus no version of \((v_p, v_q)\) crosses the remaining edges. Similarly, no version of \((v_r, v_s)\) crosses the remaining edges. Finally, the remaining edges separate the versions of \((v_p, v_q)\) from the versions of \((v_r, v_s)\). Q.E.D.

Lemma 9 completes the description of phase one of the solution to subproblem S.2.

3.2. Subproblem S.2, Phase two

**Lemma 10:** Let \( v_p \) and \( v_q \) be any two non-consecutive vertices of convex polygon \( V \); and let \( R_1 \) and \( R_2 \) be the two regions into which segment \((v_p, v_q)\) divides polygon \( V \). If \( a \) is a point in \( R_1 \) and \( b \) is a point in \( R_2 \), then segment \((a, b)\) crosses segment \((v_p, v_q)\).

**Proof:** Polygon \( V \) is convex, therefore every point of segment \((a, b)\) must be in \( V \). Points \( a \) and \( b \) are in \( R_1 \) and \( R_2 \) respectively; thus segment \((a, b)\) must cross the boundary between \( R_1 \) and \( R_2 \). Q.E.D.
Without loss of generality, let us now assume that edge \((v_1, v_2)\) is horizontal (as in figure 9). Let us look at any edge \((a', b')\) in \(E\) and any other edge \((a, b)\) in \(E\) which is to the right of \((a', b')\). From Lemma 10, for any point \(c\) to the left of \((a', b')\), segments \((c, a)\) and \((c, b)\) will intersect segment \((a', b')\). Let us assume that vertex \(c\) is not in lune \((a', b')\). From Corollary 5.1, either

1) \(c\) is not in lune \((a, b)\), or

2) \(c\) is in lune \((a, b)\), implying that \(a'\) or \(b'\) is also in lune \((a, b)\).

When edge \((a, b)\) is tested for membership in set \(T\), there is no need to look at point \(c\) because in case 1) \(c\) is not in lune \((a, b)\) and in case 2) a point other than \(c\) is also in lune \((a, b)\). Using this observation, we can perform the second phase of subproblem S.2 in the following manner:

In a left-to-right scan of \(E\), each edge \((a', b')\) of \(E\) is checked against each vertex \(c\) which is to the left of the edge. If vertex \(c\) is in lune \((a', b')\) then edge \((a', b')\) is removed from set \(E\). If vertex \(c\) is not in lune \((a', b')\) then vertex \(c\) is discarded, because for each edge \((a, b)\) in \(E\) to the right of \((a', b')\), there is no need to look at point \(c\).

We then repeat this procedure in a right-to-left scan of the edges.

A more formal description of the two scans is shown on the next page.
PROCEDURE PHASE2;
(* Input: The list E of edges from procedure PHASE1. *)
1. Sort the vertices \( v_k \) of polygon \( V \)
   by their \( x \)-coordinate in ascending order in linear time;
   (* Initialize a set \( C \) of points
   which are candidates for membership in lunes: *)
2. \( C \leftarrow \emptyset \);
   (* Scan the edges from left to right to remove edges from \( E \): *)
3. FOR \( k \leftarrow 1 \) TO \( n \) DO BEGIN
4.   FOR each edge \((v_j,v_k)\) such that \( v_j \) is left of \( v_k \) DO BEGIN
5.     FOR each point \( c \) \( \in \) \( C \) such that \( c \neq v_j \) DO BEGIN
6.       IF \( c \) is in lune \((v_j,v_k)\) THEN BEGIN
7.         remove edge \((v_j,v_k)\) from set \( E \);
8.         goto nextedge; END
9.     ELSE BEGIN
10.        remove point \( c \) from set \( C \);
11.     END
12. nextedge:
13. END;
14. END;
15. add point \( v_k \) to set \( C \);
16. END;
   (* Scan the edges from right to left to remove edges from \( E \): *)
17. \( C \leftarrow \emptyset \);
18. FOR \( k \leftarrow n \) DOWNTO \( 1 \) DO BEGIN
19.   FOR each edge \((v_k,v_j)\) such that \( v_j \) is right of \( v_k \) DO BEGIN
20.     FOR each point \( c \) \( \in \) \( C \) such that \( c \neq v_j \) DO BEGIN
21.       IF \( c \) is in lune \((v_k,v_j)\) THEN BEGIN
22.         remove edge \((v_k,v_j)\) from set \( E \);
23.         goto nextedge; END
24.     ELSE BEGIN
25.        remove point \( c \) from set \( C \);
26.     END
27. nextedge:
29. END;
30. add point \( v_k \) to set \( C \);
31. END;
32. OUTPUT set \( E \) as the set of edges in \( \text{RNG}(V) \) which go from chain \( V_0 \cup V_1 \) to chain \( V_2 \cup V_3 \);
END (* of procedure PHASE2 *)

With a careful choice of data structures, each addition or removal of an element in sets \( E \) and \( C \) will take constant time. Thus procedure PHASE2 takes linear time. Procedures PHASE1 and PHASE2 together solve subproblem S.2 (the problem of finding the edges of \( \text{RNG}(V) \) which go from chain \( V_0 \cup V_1 \) to chain \( V_2 \cup V_3 \)). The solution for subproblem S.3 (finding the edges of \( \text{RNG}(V) \) going from \( V_1 \cup V_2 \) to \( V_3 \cup V_0 \)) is similar except that in PHASE2 we sort the vertices along the axis determined by edge \( (v_1, v_3) \).

3.3. Subproblem S.4

In this subproblem, we wish to find the edges of \( \text{RNG}(V) \) that are on the boundary of convex polygon \( V \). We begin with a definition and some lemmas.

**DEFINITION:** For a point \( p \) and some \( k \) in \( \{0, 1, 2, 3, 4, 5\} \), \( \text{ray}(p, k) \) is the half-line drawn from \( p \) going in the direction \( (k \times 60)° \). Rays \( (p, 0) \) through \( (p, 5) \) are shown in figure 10.

**LEMMA 11:** If point \( c \) is in lune \( (a, b) \), then angle \( (a, c, b) \) is greater than \( 60° \).

**PROOF:** \( d(a, c) < d(a, b) \) and \( d(b, c) < d(a, b) \). Thus from the sine law, angle \( (a, b, c) < \) angle \( (a, c, b) \) and angle \( (b, a, c) < \) angle \( (a, c, b) \). Angle
FIGURE 10

FIGURE 11
(b,a,c) + angle (a,b,c) + angle (a,c,b) = 180°, thus angle (a,c,b) > 60°.

COROLLARY 11.1: If point c is in lune (a,b), then there must be some k in {0,1,2,3,4,5} such that ray (c,k) intersects segment (a,b).

If for each k in {0,1,2,3,4,5} we find all the cases where
i) some ray (v_i,k) crosses some edge (v_j,v_{j+1}) of V and
ii) point v_i is in lune (v_j,v_{j+1}),
then we will find all the edges of polygon V which do not belong in RNG(V).

For k=0, the following procedure will find all the cases where some ray (v_i,0) crosses some edge (v_j,v_{j+1}) and v_i is in lune (v_j,v_{j+1}):

PROCEDURE KO;
(* Initialize the set S of edges which do not belong in RNG(V): *)
1. S ← ∅;
2. Find the vertex v_{min} of V having the minimum y-coordinate;
3. Find the vertex v_{max} of V having the maximum y-coordinate;
(* The vertices v_{min} and v_{max} delimit a left-chain and a right-chain of polygon V. Label the vertices on the left-chain as \{v_1,v_2,...,v_q\} in ascending order, and label the vertices on the right-chain as \{v_1',v_2',...,v_r\} in ascending order. The rays (v_i,0) and the right chain are shown in figure 11. *)
4. j ← 1;
5. FOR i ← 2 TO q-1 DO BEGIN
6. \hspace{1em} WHILE ray(v_i,0) does not cross edge (w_j,w_{j+1}) DO
7. \hspace{2em} j ← j + 1;
8. \hspace{1em} IF vertex v_i is in lune (w_j,w_{j+1}) THEN
9. \hspace{2em} add edge (w_j,w_{j+1}) to set S;
10. END;
11. OUTPUT set S as a partial list of edges not belonging in RNG(V);
END (* of procedure KO *)
In procedure KO, the left-chain and the right-chain are traversed in linear time. Thus the procedure is executed in linear time. Similar procedures can be used for \( k = 1, 2, 3, 4, 5 \). The union of all the sets \( S \) will give us the set of edges on the boundary of \( V \) which do not belong in \( \text{RNG}(V) \). From here it is trivial to find the edges of \( V \) which belong in \( \text{RNG}(V) \).

We have now finished showing how to solve subproblems S.1 through S.4 in linear time. We have thus proven the following theorem:

**THEOREM 1:** The relative neighborhood graph of a convex polygon can be found in \( O(n) \) time.
CHAPTER 4

THE GENERAL PROCEDURE FOR FINDING THE NEAREST PAIR OF VERTICES

In this chapter, 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 decomposition of the nearest vertices problem into the four subproblems in 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 \( \omega \) be the midpoint of this bridge. Translate the convex polygons so that \( \omega \) is at the origin.
3. Rotate polygons \( V \) and \( W \) around origin \( \omega \) to make the bridge horizontal. From lemma 4, it follows that the horizontal bridge induces a vertical corridor between \( V \) and \( W \). (See figure 12.) Let \( I_1 \) and \( I_2 \) be the left and right boundaries of the vertical corridor. 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 \( v_{\text{max}} \) and \( w_{\text{max}} \). If the maximum \( y \)-coordinate of polygon \( V \) is realized by a horizontal edge, then assign the left vertex of this edge to \( v_{\text{max}} \). If the maximum \( y \)-coordinate
FIGURE 12
of polygon \( W \) is realized by a horizontal edge, then assign the right vertex of this edge to \( w_{\text{max}} \).

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

The vertices \( v_{\text{max}} \) and \( v_{\text{min}} \) delimit a left-chain and a right-chain in polygon \( V \). Similarly, vertices \( w_{\text{max}} \) and \( w_{\text{min}} \) 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. (See figure 12.)

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

PERFORMING STEPS 6, 7 AND 8 OF PROCEDURE TENSTEP

The subproblems in steps 6 and 7 are special cases of the following more general subproblem:

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

Figure 13 shows an example of this more general subproblem. We will solve the more general subproblem. The solution will be readily applicable to the cases in steps 6, 7 and 8.

**Lemma 12:** 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:** (See figure 14.) Let $l_1$ be the perpendicular bisector of edge $(b_i, b_{i+1})$ and let $h_i$ be the part of $l_1$ 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, either $h_i$ 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 12.1:** 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 $l_i$. 
FIGURE 13
If we look at steps 1 and 5 of procedure TENSTEP, we note that the first or last edge of $W_L$ might be horizontal. (See figure 15.) If $l_1$ is vertical, we will say that no points on the left of the corridor fall below $l_1$. If $l_{n-1}$ is vertical, we will say that all points on the left of the corridor fall below $l_{n-1}$.

**COROLLARY 12.2:** 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 $l_{i+1}$. Q.E.D.

**LEMMA 13:** 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 figure 16.)

**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 13.1:** Vertex $a_j$ also lies above bisectors $l_{k-2}, l_{k-3}, \ldots, l_1$.

**COROLLARY 13.2:** Vertex $a_j$ also lies below bisectors $l_{k+1}, l_{k+2}, \ldots, l_{n-1}$.

**LEMMA 14:** Let $b_k \in W_L$ be the vertex nearest to $a_j \in V_S$.

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

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

**PROOF:**

1) If $i < k$, then $a_j$ is above $l_i$; thus $d(a_j, b_{i+1}) < d(a_j, b_i)$.

2) 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 14 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 only 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 \( W \). If \( a_j \) was 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 \( \{a_1, a_2, \ldots, a_\alpha\} \) of vertices in \( V_S \), and the list \( \{b_1, b_2, \ldots, b_\beta\} \) of vertices in \( W_L \). *)

1. \( j \leftarrow 1; \)
2. Find \( k \) such that \( d(a_1, b_k) \) is minimized;
3. \( d_{\text{min}} \leftarrow d(a_1, b_k); \)
4. REPEAT
5. \( j \leftarrow j + 1; \)
6. IF \( d(a_j, b_{k-1}) < d(a_j, b_k) \) THEN
7. \hspace{1em} WHILE \( d(a_j, b_{k-1}) < d(a_j, b_k) \) DO
8. \hspace{2em} \( k \leftarrow k - 1; \)
9. ELSE
10. \hspace{1em} IF \( d(a_j, b_{k+1}) < d(a_j, b_k) \) THEN
11. \hspace{2em} WHILE \( d(a_j, b_{k+1}) < d(a_j, b_k) \) DO
12. \hspace{3em} \( k \leftarrow k + 1; \)
13. \hspace{1em} IF \( d(a_j, b_k) < d_{\text{min}} \) THEN \( d_{\text{min}} \leftarrow d(a_j, b_k); \)
14. \hspace{1em} UNTIL \( j = \alpha; \)
15. RETURN \( d_{\text{min}} \) as the minimum distance between \( V_S \) and \( W_L \);
END. (* of function FINDMIN *)

Procedure FINDMIN will work because for every \( a_j \) in \( V_S \), statements 6 through 12 will minimize the unimodal function \( d(a_j, b_k) \). Thus at the
end of each REPEAT loop iteration, variable $k$ points to the vertex $b_k$ in $W_L$ nearest to $a_j$. Statement 13 assures that $d_{\text{min}}$ 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 $\alpha$ vertices and $W_L$ has $\beta$ 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 through 12 are performed.

**Lemma 15:** Just before the execution of step 5, variable $k_0$ points to the vertex $b_{k_0}$ nearest to $a_{j_1}$.

**Proof:** If the REPEAT loop is being performed for the first time, then $j_1 = 1$; and from steps 1, 2 and 3, variable $k_0$ points to the vertex $b_{k_0}$ nearest to $a_1$. If the REPEAT loop is not being performed for the first time, then the previous execution of the REPEAT loop will have found $k_0$ such that $b_{k_0}$ is the vertex of $W_L$ closest to $a_{j_1}$. Q.E.D.

**Lemma 16:** Just before the execution of step 5 of FINDMIN, these conditions hold:

1) For all $i < k_0$, $a_{j_1}$ lies above bisector $l_{i}$.

2) For all $i \geq k_0$, $a_{j_1}$ lies below bisector $l_{i}$.

**Proof of condition 1):** The vertex of $W_L$ closest to $a_{j_1}$ is $b_{k_0}$, so from lemma 13 $a_{j_1}$ lies above bisector $l_{k_0-1}$; and from corollary 13.1 $a_{j_1}$ also
lies above $l_{k_0-2}$, $l_{k_0-3}$, $\ldots$, $l_1$. The proof of condition ii) is similar.

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 16, $a_{j_1}$ 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 17:** During the entire execution of function FINDMIN, step 8 is performed at most $2\beta$ times.

**PROOF:** Let $N$ be the number of intersections between edges in $V_S$ and bisectors of edges in $W_L$. (For example, in figure 17, $N = 7$.) 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 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.

Step 12 of function FINDMIN is executed only when $d(a_j, b_{k+1}) < d(a_j, b_k)$, i.e. only when bisector $l_k$ is below $a_j$. The execution of step 12 implies that during the current execution of the REPEAT loop, variable $k$ is only incremented. (i.e. $k > k_0$.) Thus from condition ii) of lemma 16, $a_{j_1}$ is below $l_k$ which is in turn below $a_{j_2}$. In other words, step 12 is performed only when edge $(a_{j_1}, a_{j_2})$ crosses from below $l_k$ to above $l_k$.

For a given edge $(a_{j_1}, a_{j_2})$ of $V$, the REPEAT loop is performed only once. If step 12 is performed, then during this one iteration of the REPEAT loop, variable $k$ will move only upward. Thus if a given edge $(a_{j_1}, a_{j_2})$ intersects a bisector $l_k$, then step 12 is performed at most once for that intersection.

**Lemma 18:** During the entire execution of function FINDMIN, step 12 is performed at most $2B$ times.

**Proof:** The proof is similar to that of lemma 17.

**Lemma 19:** 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. The REPEAT loop is executed $\alpha-1$ times; thus steps 4, 5, 6, 9, 10, 13 and 14 each take $O(\alpha)$ time. From lemma 17, step 8
takes $O(\beta)$ time. Step 7 is executed the same number of times as step 8, plus once for each failed WHILE test. This WHILE test can fail only once for each iteration of the REPEAT loop. Thus step 7 is executed $O(\alpha) + O(\beta) = O(\alpha + \beta)$ times. From lemma 18, step 12 takes $O(\beta)$ time. Step 11 is executed the same number of times as step 12, plus once for each failed WHILE test. This WHILE test can fail only once for each iteration of the REPEAT loop. Thus step 11 is executed $O(\alpha) + O(\beta) = O(\alpha + \beta)$ times. Q.E.D.

Since $\alpha \leq m$ and $\beta \leq n$, the previous lemma implies the following result:

**THEOREM 2:** Function FINDMIN takes $O(m + n)$ time to execute.

In linear time FINDMIN finds 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 chapter four 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.

(The reduction of step 8 down to step 6 is illustrated in figure 18.)
CHAPTER 6
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 figure 19. Let \( U \) be the convex hull of \( V_L \cup W_R \) illustrated in figure 20. For any set \( S \) of points, let \( B(S) \) be 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 \( v_{\text{max}}', v_{\text{min}}', w_{\text{max}} \) and \( w_{\text{min}} \) each does or does not determine a vertex of \( U \). The vertex of \( U \) having the maximum \( y \)-coordinate must be either \( v_{\text{max}} \) or \( w_{\text{max}} \); and the vertex of \( U \) with the minimum \( y \)-coordinate must be either \( v_{\text{min}} \) or \( w_{\text{min}} \). Thus only 9 of the 16 logical cases are possible. (These 9 cases are illustrated in figure 21.)

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

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:

1) Find \( B(V_L \cup W_R) \) in linear time. (Toussaint [9] and Shamos [6] show linear-time algorithms which find the convex hull of the union of two convex polygons.)
FIGURE 19

\[ V_L \]

\[ V_{\text{max}} \]

\[ V_{\text{min}} \]

\[ W_R \]

\[ W_{\text{max}} \]

\[ W_{\text{min}} \]

FIGURE 20
CASE 1

CASE 2

CASE 3

CASE 4

CASE 5

CASE 6

CASE 7

CASE 8

CASE 9

FIGURE 21
(i) 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 down 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 down to case 2, and case 8 can be reduced down to case 6. Case 4 can be reduced down 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 down to case 2 by negating both the x and y-coordinates of the vertices. To solve the problem of performing step 9 of TENSTEP in linear time, we are thus left to solve for cases 1, 2, 5 and 6.

At this point, some 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 \( S \). (In figure 22a, point \( s_1 \) 1-dominates set \( S \) and \( s_3 \) 3-dominates set \( S \). In figure 22b, point \( s_2 \) 2-dominates set \( S \) and point \( s_4 \) 4-dominates set \( S \).)

**Lemma 20:** 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 \( s_i \).
Figure 22a

Figure 22b
PROOF: Consider the case $k = 1$. Let $s_1$ have coordinates $(x_1, y_1)$, and let point $p$ have coordinates $(x_p, y_p)$. For every point $s_j 
eq s_1$, $x_j < x_1 < x_p$ and $y_j < y_1 < y_p$. Therefore $(x_p - x_1)^2 < (x_p - x_j)^2$ and $(y_p - y_1)^2 < (y_p - y_j)^2$. Thus $d(p, s_1) < d(p, s_j)$. The proofs for $k = 2, 3$ and 4 are similar. Q.E.D.

Now, if point $s_1$ $k$-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 $s_1$ and some point in $P$. The following function will thus find the shortest distance between sets $S$ and $P$ in linear time:

FUNCTION SCANSET($s_1, P$);
1. $d_{\text{min}} \leftarrow \text{infinity}$;
2. FOR each point $p \in P$ DO
3. \hspace{1em} IF $d(s_1, p) < d_{\text{min}}$ THEN $d_{\text{min}} \leftarrow d(s_1, p)$;
4. RETURN WITH $d_{\text{min}}$ as the shortest distance between sets $S$ and $P$;
END. (* of function SCANSET *)

LEMMA 21: 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 $\text{RNG}(S_1 \cup S_2)$.

PROOF: This result was first proven in [11]. Here is an alternate simpler proof: Let the shortest distance between $S_1$ and $S_2$ be realized by points $a \in S_1$ and $b \in S_2$. For any point $b' \in S_2$, $d(a, b') \geq d(a, b)$. Thus lune $(a, b)$ contains no point $b' \in S_2$. Similarly, for any point $a' \in S_1$, $d(a', b) \geq d(a, b)$. Thus lune $(a, b)$ contains no point $a' \in S_1$. Therefore lune $(a, b)$ contains no point in $S_1 \cup S_2$. Q.E.D.
If $C_1$ and $C_2$ are disjoint subchains of convex polygon $P = \{p_1, p_2, \ldots, p_n\}$, 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 $RNG(C_1 \cup C_2)$ which crosses from $C_1$ to $C_2$. The following function, called USERNG, uses this technique:

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

In [11], this solution is applied to a 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 of figure 21.

6.1 Cases 1, 2 and 5 of Step 9

If case 1 holds, then chains $V_L$ and $W_R$ are two subchains of a convex polygon. Thus we can find the shortest distance between chains $V_L$ and $W_R$ by applying the USERNG function to these two chains. Case 2 of figure 21 is illustrated in more detail in figure 23. One can see that
CASE 2

FIGURE 23
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 figure 23, $W' = \{w_{max}, \ldots, w_c\}$. )

Let $W''$ be the list of vertices in $W_R$ which are not on $B(V_L \cup W_R)$. 
( In figure 23, $W'' = \{w_{c+1}, \ldots, w_{min}\}$. )

Let $V'$ be the list of vertices in $V_L$ which are above $w_{min}$. 
( In figure 23, $V' = \{v_{max}, \ldots, v_{h}\}$. )

Let $V''$ be the list of vertices in $V_L$ which are not above $w_{min}$. 
( In figure 23, $V'' = \{v_{h+1}, \ldots, v_{min}\}$. )

1. $d_{min} \leftarrow$ 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($w_{min}$, $V''$) is used because $w_{min}$ 3-dominates the rest of $W''$ and every point in $V'$ 3-dominates $W''$.

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 figure 24 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 figure 24, \( W'' = \{w_{e+1}, \ldots, w_f \} \). )

Let W' be the vertices in \( W_R \) that are above \( W'' \).

( In figure 24, \( W' = \{w_{\max}, \ldots, w_e \} \). )

Let W''' be the vertices in \( W_R \) that are below \( W'' \).

( In figure 24, \( W''' = \{w_{f+1}, \ldots, w_{\min} \} \). )

Let V'' be the vertices in \( V_L \) that are below \( w_{\max} \) and above \( w_{\min} \).

( In figure 24, \( V'' = \{v_{g+1}, \ldots, v_{h} \} \). )

Let V' be the vertices in \( V_L \) that are not below \( w_{\max} \).

( In figure 24, \( V' = \{v_{\max}, \ldots, v_{g} \} \). )

Let V''' be the vertices in \( V_L \) that are not above \( w_{\min} \).

( In figure 24, \( V''' = \{v_{h+1}, \ldots, v_{\min} \} \). )

1. \( d_{\min} \leftarrow \) smallest of the five values returned by these functions:

   i) \( \text{SCANSET}(w_{\max}, V') \)

   ii) \( \text{USERNG}(V'' \cup V''' \cup W') \)

   iii) \( \text{USERNG}(V_L \cup W'') \)

   iv) \( \text{USERNG}(V' \cup V'' \cup W''') \)

   v) \( \text{SCANSET}(w_{\min}, V''') \)

2. RETURN WITH \( d_{\min} \) as the shortest distance between \( V_L \) and \( W_R \);

END. (* of function CASE5 *)

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 \( W' \).
CASE 5

FIGURE 24
and every point in \( V' \) 2-dominates \( W' \). The second \textit{SCANSET} will find the shortest distance between \( V'' \) and \( W''' \) because \( v_{\text{min}} \) 3-dominates the rest of \( W''' \) and every point in \( V'' \) 3-dominates \( W''' \). Thus \textit{FUNCTION CASE5} will correctly find the minimum distance in case 5 in \( O(m+n) \) time.

### 6.2 Case 6 of Step 9

Case 6 is illustrated in figures 25a through 25d. 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 \( w_{\text{max}} \), and let \( w_h \) be the highest vertex of \( W_R \) which is below \( v_{\text{min}} \). Vertices \( v_c \), \( w_c \), \( v_h \) and \( w_h \) are illustrated in figures 25a through 25d. 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 think that the following procedure will find the shortest distance between \( V_L \) and \( W_R \):

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

\[
1) \ \text{SCANSET}(w_{\text{max}}, V') \\
2) \ \text{USERNG}(V'' \cup V''' \cup W') \\
3) \ \text{USERNG}(V_L, W') \\
4) \ \text{USERNG}(V' \cup V'' \cup W''') \\
5) \ \text{SCANSET}(v_{\text{min}}, W''')
\]

This will not work in all situations, however. If \( v_c \) is higher than \( v_h \) (as in figure 25a), then the third use of \textit{USERNG} need not work because \( V'' \) is not in \( B(V' \cup V'' \cup W''') \). A simple sequence of \textit{SCANSET}s and \textit{USERNG}s that works for all occurrences of case 6 was not found.
CASE 6.1

FIGURE 25a
CASE 6.2

FIGURE 25b
CASE 6.3

FIGURE 25c
CASE 6.4

FIGURE 25d
However, case 6 can be further decomposed into the following four sub-cases:

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 order in figures 25a through 25d. For case 6.1, the following statement will find the shortest distance between $V_L$ and $W_R$:

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

1) $\text{SCANSET}(w_{\text{max}}, V' \cup V'')$
2) $\text{USERNG}(V''', W')$
3) $\text{USERNG}(V_L, W')$
4) $\text{USERNG}(V', W''')$
5) $\text{SCANSET}(v_{\text{min}}, W''')$

The first $\text{SCANSET}$ will find the shortest distance between $V' \cup V''$ and $W'$, and the second $\text{SCANSET}$ will find the shortest distance between $V'' \cup V'''$ and $W'''$. For case 6.2, the following statement will find the shortest distance between $V_L$ and $W_R$. 
\[ d_{\text{min}} \leftarrow \text{smallest of the five values returned by these functions:} \]

1. \( \text{SCANSET}(w_{\text{max}}, V' \cup V'') \)
2. \( \text{USERNG}(V''', W') \)
3. \( \text{SCANSET}(w_h, V_L) \)
4. \( \text{USERNG}(V', W''') \)
5. \( \text{SCANSET}(v_{\text{min}}, W''') \)

The first \text{SCANSET} will return with the shortest distance between 
\( V' \cup V'' \) and \( W' \). The second \text{SCANSET} will return with the shortest dis-
tance 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 \text{SCANSET} will return 
with the shortest distance between \( V'' \cup V''' \) and \( W''' \). For case 6.3, 
the following statement will find the shortest distance between \( V_L \) and 
\( W_R \):

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

1. \( \text{SCANSET}(w_{\text{max}}, V') \)
2. \( \text{USERNG}(V' \cup V''', W') \)
3. \( \text{USERNG}(V_L, W'') \)
4. \( \text{USERNG}(V' \cup V'', W''') \)
5. \( \text{SCANSET}(v_{\text{min}}, W''') \)

The first \text{SCANSET} finds the shortest distance between \( V' \) and \( W' \),
and the second \text{SCANSET} finds the minimum distance between \( V''' \) and \( W''' \).
Finally, for case 6.4, this last statement will find the shortest dis-
tance between \( V_L \) and \( W_R \):
\( d_{\text{min}} \leftarrow \text{smallest of the five values returned by these functions:} \)

1) \( \text{SCANSET}(w_{\text{max}}, V') \)

2) \( \text{USERNG}(V'' \cup V', W') \)

3) \( \text{SCANSET}(v_h, V_L) \)

4) \( \text{USERNG}(V' \cup V'', W''') \)

5) \( \text{SCANSET}(v_{\text{min}}, W''') \)

The first \( \text{SCANSET} \) finds the shortest distance between \( V' \) and \( W' \). The second \( \text{SCANSET} \) 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 \( \text{SCANSET} \) 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. 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:

**THEOREM 3:** The minimum vertex-distance distance between \( V \) and \( W \) can be computed in \( O(m+n) \) time.
CHAPTER 7
A LOWER BOUND ON THE COMPLEXITY OF THE PROBLEM

THEOREM 4: The complexity of finding the shortest distance between the vertices of two convex polygons is $O(m+n)$.

PROOF (due to Toussaint [14]): Two polygons requiring a linear search are constructed and illustrated in figure 26. 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.
CHAPTER 8

CONCLUSION

In this paper we have exhibited linear upper and lower bounds on the problem of finding the closest pair of vertices between two 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 the distance between sets [11]. An immediate extension of this problem, not treated here, is the case of intersecting polygons. By combining techniques used in this paper with existing results on polygon decomposition 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 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.
REFERENCES


