WEDGE PLACEMENT OPTIMIZATION PROBLEMS

by

Marek Teichmann

School of Computer Science
McGill University, Montréal

October 1989

A THESIS SUBMITTED TO FACULTY OF GRADUATE STUDIES AND RESEARCH
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

Abstract

In the Euclidean plane, consider a point set S and a *wedge* W defined as being the intersection of two half-planes whose bounding lines intersect at an apex O and form a fixed angle.

We study the problem of determining the position of W relative to S, with translations and rotations allowed, that minimizes the distance between O and S and is subject to the constraint that S remains inside W. One measure of distance considered is the minimum Euclidean distance from O to S. An $O(n \log n)$ time algorithm is presented for the case where S is a set of n points or a simple polygon with n vertices. If we are given an appropriate Voronoi diagram, the solution can be obtained in $O(n)$ time. Also an $O(n)$ time algorithm is given for convex polygons.

Other measures of distance and related problems are also discussed.

Résumé

Dans le plan euclidien, nous considérons un ensemble S de points et un *coin* W défini comme l'intersection de deux demi-plans dont les droites déterminantes se croisent en O et forment un angle fixe.

Nous étudions le problème de déterminer la position de W par rapport à S, translations et rotations permises, qui minimise la distance entre O et S et qui est telle que S reste dans l'intérieur de W. Une mesure de la distance entre O et S étudiée est la distance euclidienne minimale pour laquelle est présenté un algorithme qui trouve la position optimale en temps $O(n \log n)$ lorsque S est un ensemble de n points ou un polygone simple de n sommets. Si un diagramme de Voronoi approprié nous est donné ou si le polygone est convexe, la solution peut être obtenue en temps $O(n)$.

D'autres mesures de distance ainsi que des problèmes apparentés sont aussi exposés

Acknowledgements

I would like to thank my thesis supervisor, Godfried Toussaint, for offering the Computational Geometry course where these ideas were born, for his advice and for providing a continuing creative setting for this work. I also appreciate the advice of David Avis, as well as the helpful comments of Michael Houle, Gilles Pesant and Alain Leblanc.

I also thank the Natural Sciences and Engineering Research Council of Canada for their generous support through a postgraduate scholarship.

Contents

Abstract 1

Résumé 2

Acknowledgements 3

1 Introduction 6

2 Voronoi Diagrams 10
- 2.1 Proximity Problems 10
- 2.2 Definitions 12
- 2.3 Some Properties 14
- 2.4 Computing the Voronoi Diagrams 15
  - 2.4.1 The Site Voronoi Diagram Case 16
  - 2.4.2 The Polygon Voronoi Diagram Case 17
  - 2.4.3 The Merge of Two Voronoi Diagrams 18
- 2.5 Data Structures for Implementing Voronoi Diagrams 19

3 The CWP-cp/cs problems for convex polygons 21
- 3.1 Observations 22
  - 3.1.1 Contact Vertex Pairs 24
- 3.2 The Path of the Apex 26
  - 3.2.1 Obtaining the Minimum Distance 27
- 3.3 Algorithms CWP-cp and CWP-cs 30

4 The CWP-p and -s problems 38
- 4.1 Properties of Voronoi diagrams and $CLOUD(P,\omega)$ 40
- 4.2 Algorithms CWP-p and CWP-s 44

5 Conclusion and Further Research 50

Bibliography 54

List of Figures

2.1 Voronoi diagram of point sites. 11
2.2 Generalized Voronoi diagram of a polygon (thin solid lines.) 11
2.3 Bisector of a point and a closed segment. 12
2.4 Example of a strip and e-promontory. 14

3.1 Closest point is on an edge of P. 23
3.2 Non stable optimal placement. 24
3.3 Finding the next stable placement. 25
3.4 Possible closest pairs. 28
3.5 Showing how $c(a)$ moves in the same direction as a. 31
3.6 Example of $CLOUD(P, \pi/2)$. 34
3.7 Theorem 3.13 Case $\omega < \pi/2$. 35
3.8 Theorem 3.13. Case $\omega \geq \pi/2$. 36

4.1 Closest point of P behind a hull edge. 39
4.2 Obtaining the convex hull from the Voronoi diagram. 40
4.3 Showing monotonicity of a v-path and its relationship with cloud arcs 42
4.4 A late arc of a circle is contained inside the next circle. 43
4.5 Finding the intersections of $CLOUD(H, \omega)$ and $VOR(P)$ 47

5.1 A type of robot hand. 51

Chapter 1

Introduction

The study of optimization problems is an important area of Computational Geometry. Proximity problems in the Euclidean plane, such as finding the closest pair of a point set [Kn73] or the largest empty circle in a point set [Sh77,To83b], involve static optimization. More recently, dynamic optimization problems involving objects that can be translated, rotated or varied in size have been considered. Chazelle [Ch83] and Fortune [Fo85] give algorithms to determine whether one polygon can be positioned such that it is contained in another and to exhibit such a *containing placement*. The problem of finding the largest polygon, subject to some restriction, that fits inside a given polygon, with various definitions of 'largest' (area, perimeter) has also been studied [BDDG82,Ch86,CY84]. Those restrictions concern convexity, number of vertices, and specification of angles. Similarly, the problem of finding the smallest polygon containing another fixed polygon and satisfying similar restrictions has been looked at [Ch86,KL85,OAMB84].

In this thesis, we will be considering a set of closely related problems concerning a set, either a polygon or a set of points, and a corner or *wedge* of fixed angle. The wedge is defined as the intersection of two half-planes, and the intersection point of the two defining half-lines is its *apex*. The problem is then to minimize the 'distance' between the apex and the set, allowing translations and rotations of the wedge and subject to the constraint that the set remains inside the wedge. Several definitions of distance will be considered for both types of sets.

In the case of polygons, the distance minimized is the usual distance between the apex and the boundary of the polygon. This will be called problem CWP-p, Closest Wedge Placement for simple Polygons, or CWP-cp for convex polygons. In problems CWP-s for point sets, and CWP-cs for sets of vertices of convex polygons, it is the distance from the apex to the closest point in the set. Another measure of closeness for both types of sets, the smallest radius circle centered at the apex and containing the set, will be examined in the MiniMax-WP problem. Finally we will mention the problem of closing off the wedge with a line segment such that the triangle created contains our set and has minimum area or perimeter, this will be the triangle problem. Similar questions without angular restrictions have been addressed in [Ch86,KL85].

These problems have interpretations in other areas. The study of movable objects mentioned above is closely related to motion planning and compliant motion planning. Indeed, to solve our CWP problems we compute the path taken by the apex as the wedge rotates around the set, remaining in contact with it. The work of Hopcroft and Wilfong in [HW84] has shown the importance of the study of compliant motion. The CWP problems can also be considered an instance of *local motion planning* [Ya86b] in which the motion of an arbitrary polygonal robot is planned in a fixed environment. Other examples of local motion planning are the problems of *moving a chair through a door* [St82] or *around a corner in a corridor* solved algorithmically by Yap [Ya86a] and Maddila and Yap [MY86] respectively.

In classical motion planning, the question of finding a continuous motion from one configuration to another has been addressed, but it may be interesting to optimize the final configuration in some way. For instance, the triangle problem solves the problem of finding a placement of a polygonal robot in a corner, so as to occupy as little space as possible in some large room. When solving the CWP-p problem, we want to waste as little space as possible in the corner, by minimizing the radius of the largest empty circle centered at the apex.

Another relevant and relatively new area in Computational Geometry is the study of visibility problems [OR87]. A point p is *visible* from point q and vice-versa

if the line segment $\overline{pq}$ intersects no object. In particular, one may be interested in the placement of guards subject to some restriction in order to see the interior or exterior of a polygon. Those guards have a 360 degree field of vision. However, if we consider a guard to be camera with a certain limited field of vision represented by our wedge, then the CWP-p/s problem becomes a *camera location problem* and finds the position closest to the object and direction of the camera such that it completely sees a polygonal object or set of sites. If the depth of field of the camera is limited then one might want to get as close as possible to the furthest point and solve the MiniMax-WP problem for sets.

The model of computation used will be the *real RAM* as described in [Sh77]. As $O(1)$ time operations on real numbers, it allows arithmetic operations, comparisons, indirect addressing with integer addresses and analytic functions. It does not include the floor function but permits the calculation in constant time of the intersection of any two of the following: a circle, an arc, a line (segment), a parabolic arc, etc...

In Chapter 2 we will describe some properties and the algorithmic construction of the Voronoi diagrams of sets of segments and points and of point sets. These data structures are used for solving the CWP-p and CWP-s problems respectively without the convexity restrictions. The Voronoi diagram for points is also used in some of the other problems mentioned above.

Chapter 3 characterizes the motion of the wedge around a convex polygon and presents some properties that allow us to obtain an algorithm that solves the CWP-cp problem in time linear in the number of vertices of the polygon. The algorithm is modified to solve the CWP-cs problem in the same time complexity.

Chapter 4 extends the results in Chapter 3 and, using the data structure presented in Chapter 2, presents an algorithm that solves the CWP-p and CWP-s problems in $O(n \log n)$ time, where n is the size of the polygon or point set. Moreover, since the step of the algorithm for CWP-p/s that computes the Voronoi diagram is the only step that takes more that $O(n)$ time, we can obtain the solution to the CWP-p/s problems in $O(n)$ time if we are given this diagram. These algorithms will be presented in [Te89].

Finally, Chapter 5 will summarize the results and discuss the other variants of the problem as well as some open problems.

Chapter 2

Voronoi Diagrams

2.1 Proximity Problems

The Voronoi diagram is one of the most fundamental data structures of computational geometry [Sh77]

Given a set of n points or *sites* in the Euclidean plane, we wish to know which points of the plane are closer to some site than any other site. The Voronoi diagram is the partition of the plane into regions, each region being the locus of points closer to a particular site.

The Voronoi diagram can be used as a powerful tool to give efficient algorithms for various geometric proximity problems. These include finding the closest pair of sites, the closest neighbor of each site, the Euclidean minimum spanning tree of the sites or the largest site-free circle with center inside a simple polygon [To83b].

We will be considering two types of Voronoi diagrams: the Voronoi diagram for point sites and the generalized Voronoi diagram for line segments and points with the restriction that these line segments must form a simple polygon and the points are the segment endpoints. Those points and segments will be called *Voronoi elements*. Both diagrams have very similar structures and properties as well as similar algorithms to compute them and will be presented together. Examples of these diagrams are given in Figures 2.1 and 2.2 respectively.

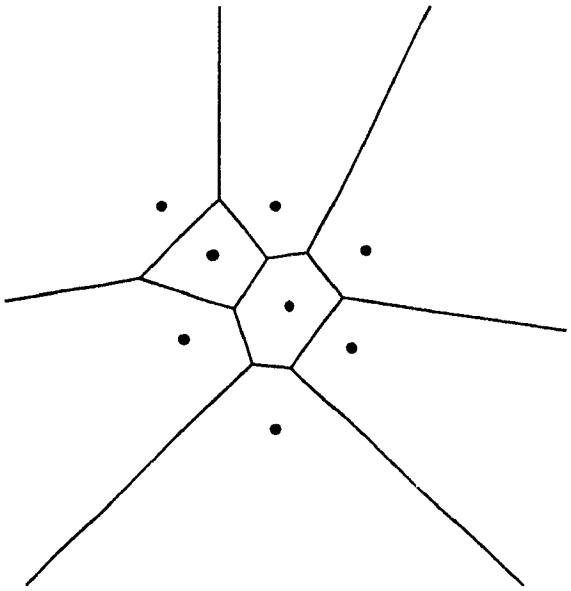

Figure 2.1: Voronoi diagram of point sites.

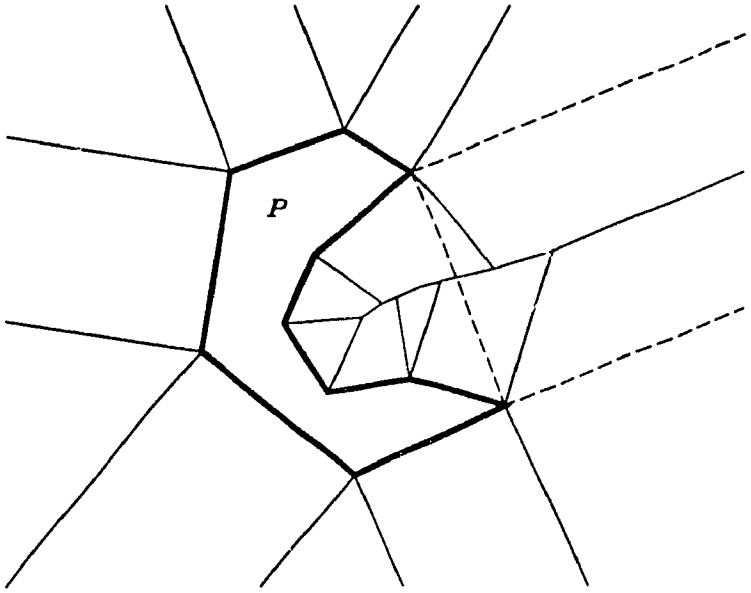

Figure 2.2: Generalized Voronoi diagram of a polygon (thin solid lines.)

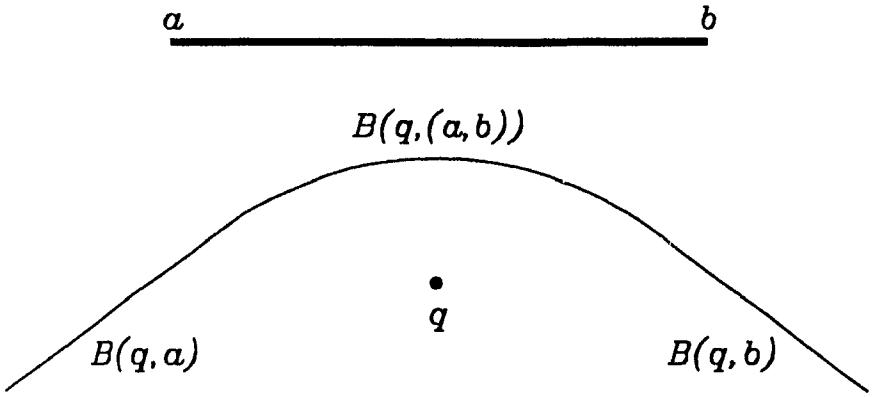

Figure 2.3: Bisector of a point and a closed segment.

2.2 Definitions

To begin, we will need a few definitions. We denote the closed line segment between two points a and b by $\overline{ab}$, and the open segment by (a,b). Also *line* ab refers to the line passing through a and b.

The 'projection' $p(q,\overline{ab})$ of a point q onto a closed segment $\overline{ab}$ is the intersection point of the line ab and the line perpendicular to ab and passing through q.

The Euclidean distance between two points q and r is denoted by $d(q,r)$. The distance $d(q,\overline{ab})$ between a point q and a closed segment $\overline{ab}$ in the Euclidean metric is the distance $d(q,x)$, $x = p(q,\overline{ab})$, between q and its projection x onto $\overline{ab}$ if x belongs to $\overline{ab}$ and is $\min(d(q,a),d(q,b))$ otherwise. Also let $c_{\mathcal{S}}(q)$ denote the point closest to q in the set $\mathcal{S}$.

The *bisector* $B(e_i,e_j)$ of two elements e_i and e_j is the locus of points equidistant from e_i and e_j but not closer to any other element and is connected. Similarly, the bisector $B(X,Y)$ of two sets of elements is the locus of points equidistant from X and Y, i.e. $B(X,Y) = \{q : \min_{e\in X} d(q,e) = \min_{e\in Y} d(q,e)\}$.

A bisector $B(e_i,e_j)$ will be *oriented* such that e_i lies to the left of it and e_j to the right of it. We can similarly orient bisector $B(X,Y)$. These bisectors will be line segments and portions of parabolas [LD81]. For example, see Figure 2.3, the bisector of a point q and a segment $\overline{ab}$ has three components: the half-lines $B(q,a)$

and $B(q,b)$, and a portion of a parabola whose focus and directrix are the point q and the line ab respectively.

With these definitions we can now define the region $h(e_i,e_j)$ as the locus of points closer to element e_i than to element e_j and similarly for two sets of elements.

Given a set S of n elements $\{e_i : i = 1,\ldots,n\}$, we define the *Voronoi region* $V(e_i)$ to be the intersection of all regions $h(\cdot,\cdot)$ containing e_i, *i.e.* $V(e_i) = \bigcap_{j\neq i} h(e_i,e_j)$, which is the locus of points closer to e_i than to any other element. The boundary edges of $V(e_i)$ are called Voronoi edges and its vertices, Voronoi points or Voronoi vertices. The collection of those is the Voronoi diagram $VOR(S)$.

We will also need the following definition that will be used to characterize Voronoi regions.

A region R in the plane is *weakly-internally-visible* from *kernel* C, C convex subset of R, if for any point $r \in R$ there is a point $c \in C$ such that the line segment $\overline{rc}$ lies completely within R.

We will take a *simple polygon* to be the region of the plane delimited by a closed simple polygonal path and containing the boundary.

Given a simple polygon P of n vertices $q_0,q_1,\ldots,q_{n-1}$, a vertex q_i is called *convex* if the internal angle at q_i is less than π and *reflex* otherwise. We will assume without loss of generality that no three consecutive vertices are collinear.

Let s_i denote the segment (q_i,q_{i+1})[1]. A maximal *reflex chain* is a sequence of elements $s_j,q_{j+1},s_{j+1},\ldots,q_{k-1},s_{k-1}$ such that the vertices q_j and q_k are convex and $q_{j+1},\ldots,q_{k-1}$ are reflex. A convex chain is defined similarly. We will omit the word maximal as there will be no ambiguity.

Let S be a point set. We will denote the convex hull of S by $CH(S)$ [Sh77]. Let H be $CH(S)$. We define the *normals* of H to be the half-lines extending from the vertices of H and perpendicular to the edges incident to those vertices. There are two normals per edge. These normals partition the exterior of H into V-shaped wedges each having the corresponding vertex as apex and regions called *strips*, bounded by an edge of H and the two incident normals. If $\overline{uv}$ is the edge of H

[1] Indices will be taken modulo the size of the set they index

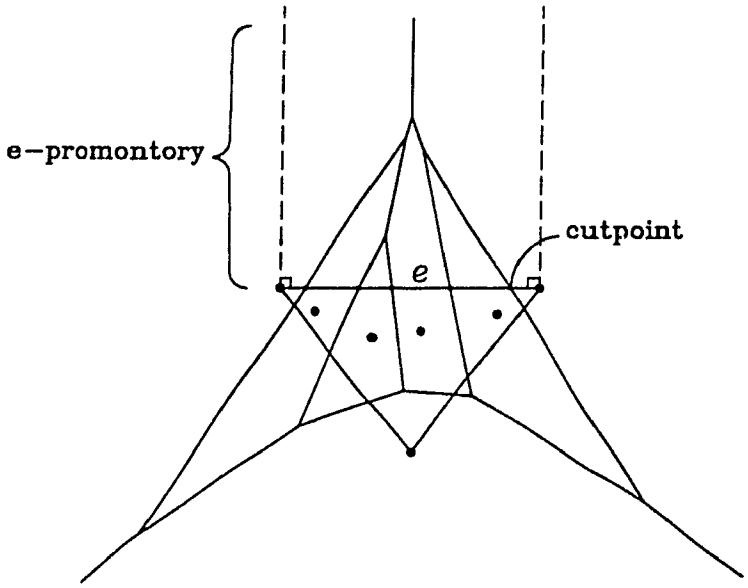

Figure 2.4: Example of a strip and e-promontory.

then let $NORMAL(u, v)$ be the normal to $\overline{uv}$ at u. See Figure 2.4. The points of intersection of $VOR(S)$ with H are called *cutpoints* and the Voronoi diagram intersected with the strip defined by an edge e (e-strip) is called the *e-promontory.* This terminology is due to [ACGOY88].

2.3 Some Properties

We will next describe the main properties of the Voronoi diagrams necessary for their algorithmic construction as well as for the CWP problems.

The following lemma is fundamental.

Lemma 2.1 [LD81] $V(e_i)$ *is weakly-internally-visible from kernel* e_i.

Proof We will show that given a point q, if $q \in V(e_i)$ then $\overline{qq'}$ lies completely in $V(e_i)$, where $q' = c_{e_i}(q)$. Assume a point x on $\overline{qq'}$ is in $V(e_j), i \neq j$. Let $x' = c_{e_j}(x)$. We have $d(q, x') \leq d(q, x) + d(x, x')$ by the triangle inequality. Since $d(x, x') = d(x, e_j) < d(x, e_i)$, we get $d(q, x') < d(q, x) + d(x, e_i)$. But $d(q, x) + d(x, q') = d(qq')$

and $d(x,e_i) = d(x,q')$ which implies that $d(q,x') < d(q,q') = d(q,e_i)$. Therefore $d(q,e_j) < d(q,e_i)$ which contradicts the hypothesis that $x \in V(e_j)$. ∎

We state the following additional facts from [LDS1] without proof.

Lemma 2.2 *The Voronoi regions $V(e_i)$ and $V(e_j)$ share an edge if and only if there exists a point q such that the circle centered at q with radius $d(q,e_i) = d(q,e_j)$ does not contain any point of other elements in its interior or on its boundary.*

The following theorem shows that when a point x is moved on the boundary of $V(e_i)$ in a given direction then its projection $c_{e_i}(x)$ also moves along e_i in the same direction. If e_i is a point then the projection is fixed at that point.

Theorem 2.3 *Let x_1 and x_2 be two points on the boundary of $V(e_i)$ and x_1' and x_2' their images under $c_{e_i}(\cdot)$. Either one of the two segments $\overline{x_1x_1'}$ or $\overline{x_2x_2'}$ properly contains the other or they do not intersect except possibly at endpoints.*

Another basic fact about Voronoi diagrams is

Theorem 2.4 *The element e_i is on the convex hull $CH(S)$ of S if and only if its associated Voronoi region is unbounded.*

And finally, we state a result that limits the size of the diagrams.

Theorem 2.5 *Given a set S of n elements, then the number of Voronoi edges and the number of Voronoi points in $VOR(S)$ are both $O(n)$.*

2.4 Computing the Voronoi Diagrams

The types of Voronoi diagrams considered here will be restricted versions of the type for line segments and points. We will need essentially two types of diagrams. For the CWP-s problem, the usual Voronoi diagram for point sites will be needed. For the CWP-p problem, we will require the Voronoi diagram for the exterior of a simple polygon, where the latter is considered to be a set of open segments and

vertices. Both problems can be solved in $O(n \log n)$ time where n is the number of elements. The solution for the first type was first presented by Shamos and Hoey [SH75]. An efficient solution to a problem closely related to the second, namely the computation of the medial axis transformation of the *interior* of a simple polygon (see [DH73] for a definition and properties), was given by Lee [Le82]. In section 2.4.2, we will describe how to modify his algorithm to obtain the Voronoi diagram for the exterior of a polygon in the same time complexity.

A popular problem-solving paradigm, the divide and conquer method, can be used to compute these diagrams. For the two problems the implementation of the splitting and merging is different. However, the merge step can be described in a uniform manner for both types of diagrams, as they are conceptually equivalent. In [Fo86], Fortune presents an elegant sweep line algorithm for the computation of the Voronoi diagram for line segments and points. However, the divide and conquer method, with its scanning and traversal procedures, is much closer to the type of algorithms considered in this thesis and will thus be described instead.

Let S be the set of n elements (points and edges). The general divide and conquer algorithm proceeds as follows. First it partitions S into two subset S_1 and S_2 of approximately equal size. It then recursively constructs $VOR(S_1)$ and $VOR(S_2)$. Finally $VOR(S_1)$ and $VOR(S_2)$ are merged. This algorithm runs in $O(n \log n)$ time provided the merge can be done in time proportional to n. We will next describe the splitting and merge steps for the two cases.

2.4.1 The Site Voronoi Diagram Case

When elements are points only, the split will be done by sorting the points by x-coordinate and dividing the resulting sequence into two parts, one of size $\lfloor \frac{n}{2} \rfloor$, the other of size $\lceil \frac{n}{2} \rceil$

Using such a split, it can be shown [Sh77] that there exists a *merge curve*, which is in fact $B(S_1, S_2)$. It is a monotone chain of line segments such that $VOR(S_1 \cup S_2) = (VOR(S_1) \cap h(S_1, S_2)) \cup (VOR(S_2) \cap h(S_2, S_1))$ where $h(X, Y)$ is the region of the plane on the left side of the oriented bisector $B(X, Y)$. More-

over, the first and last elements of the chain are infinite rays and are the bisectors of the two supporting segments of $CH(S_1)$ and $CH(S_2)$ by Lemma 2.2. These convex hulls can be maintained recursively [Sh77] or can be 'read off' the Voronoi diagrams in linear time [PS85, p. 215]. One of the two infinite rays is chosen as the *starting bisector* for the merge.

2.4.2 The Polygon Voronoi Diagram Case

When elements are both points and line segments, the bisector of two sets of elements may in general have several disjoint components. However in the case of two polygonal chains, the bisector is composed of only one piece [Le82]. Moreover it is clear that the Voronoi diagram for the exterior of a convex polygon is exactly the set of normals of all the edges of the polygon. The same is true for the diagram of one side of a convex chain. (An edge $\overline{ab}$ is considered to be the set $\{a, b, (a,b)\}$ and its Voronoi diagram consists of the two lines perpendicular to (a,b) and passing through a and b.)

In [Le82], D.T. Lee computes the Voronoi diagram for the interior of a simple polygon by first computing the diagram for the 'interior side' of reflex chains which are easily obtained by scanning the edges of the polygon in linear time. Note that the process of computing this diagram is identical to computing the diagram of the exterior side of convex chains.

We will need the following definitions:

> For a simple closed curve C, let $INT(C)$ denote the interior of C.
>
> A *pocket* of P is one connected component of the closure of $CH(P) \setminus INT(P)$. It is associated with the convex hull edge or pocket *hd* that is contained in it.

In Lee's algorithm, the diagrams of these reflex chains, say $C_1, C_2, \ldots, C_m$ are then merged two at a time to obtain $VOR(C_1 \cup C_2)$, $VOR(C_3 \cup C_4)$, etc... The process is repeated recursively. At each stage $O(n)$ work is done for the merge and there are $O(m) \subset O(n)$ chains to be processed. The total time required is therefore $O(n \log n)$.

Our algorithm will be similar to the one just described, but we will compute the diagram of the exterior side of convex chains as noted above. These chains are the part of the boundary of the polygon coinciding with its convex hull Moreover, we will compute the diagram for each pocket of the polygon separately. By the following lemma, the merge will be just a union of all the diagrams computed since they are all totally independent (non-overlapping). This is illustrated in Figure 2.2 in which the strip corresponding to a convex hull edge is shown in dashed lines.

One can thus define the e-promontory for Voronoi diagrams of polygons. It consists of either exactly the boundary of the Voronoi region of e if e is an edge of P and of $CH(P)$ or the part outside of $CH(P)$ of the diagram of a pocket if e is a lid of a pocket.

Lemma 2.6 *Let e be an edge of $CH(P)$. The e-promontory is contained in the strip corresponding to e.*

Proof If e is also an edge of P, the e-promontory is the boundary of the e-strip. Otherwise, let $e = \overline{uv}$. Then $V(u)$ and $V(v)$ are unbounded by Theorem 2.4. Moreover $NORMAL(u, v)$ cannot intersect any other Voronoi region, since any point of $NORMAL(u, v)$ is closer to u than any other element. The same is true for $NORMAL(v, u)$, hence no Voronoi region other than $V(u)$ and $V(v)$ intersects the normals (except maybe at u and v). ∎

For a given pocket, the split will be done by dividing the set of convex chains into two sets of adjacent chains of approximately the same size. Some chains may be reduced to only an edge. The split will be at a reflex vertex and the *starting bisector* will be a ray bisecting the angle formed by the two edges adjacent to that vertex.

2.4.3 The Merge of Two Voronoi Diagrams

We assume now that $VOR(S_1)$ and $VOR(S_2)$ are available and show how the two diagrams can be merged in $O(n)$ time. This is done by constructing the merge curve $B(S_1, S_2)$ and deleting the edges of S_1 on the right of $B(S_1, S_2)$ and the edges of S_2 on the left.

We can imagine moving a point x along the current bisector $B(e_i, e_j)$, $e_i \in S_1, e_j \in S_2$, until it enters a new Voronoi region, say of $e_s \in S_1$. x will then continue to the right of $V(e_s)$ along $B(e_s, e_j)$ since at this point e_s became closer to x than e_i.

The current (oriented) bisector $B(e_i, e_j)$ is initialized to be the *starting bisector* Starting with the last examined edge, we scan the edges of the adjacent Voronoi region on the left of $B(e_i, e_j)$, $V(e_i)$, in counterclockwise direction to find the edge $B(e_i, e_s)$ which intersects the current bisector. The same scan is performed clockwise on $V(e_j)$ to find the edge $B(e_t, e_j)$. We then choose the new bisector that first intersects the current one in the sense of the previous paragraph.

If $B(e_i, e_s)$ intersects $B(e_i, e_j)$ first then the next current bisector will be $B(e_s, e_j)$, otherwise it will be $B(e_i, e_t)$.

For both types of Voronoi diagrams, the construction of $B(S_1, S_2)$ terminates when no more Voronoi edges intersect the merge curve.

Finally, the following theorem from [LD81] states that the scan described above will work without backtracking, and establishes along with Theorem 2.5 that the merge can be done in $O(n)$ time. Hence the Voronoi diagrams can each be computed in $O(n \log n)$ time.

Theorem 2.7 *Let e_i and e_j be elements of S_1 and S_2 respectively, and let $B(e_i, e_j)$ be constructed during the merge process. No point of $V(e_i)$ in $VOR(S_1)$ which lies to the right of the oriented bisector $B(e_i, e_j)$ will be included in the region $V(e_i)$ in the final diagram and similarly for $V(e_j)$. Furthermore, the scanning of $V(e_i)$ in a counterclockwise direction and $V(e_j)$ in a clockwise direction will find the first intersection between $B(e_i, e_j)$ and either $V(e_i)$ or $V(e_j)$.*

2.5 Data Structures for Implementing Voronoi Diagrams

We will need to perform rapidly various operations on these diagrams. One data structure allowing the necessary operations is the Doubly Connected Edge List or

DCEL [MP78]. In this data structure, for each edge there is a node containing information about adjacent vertices, faces, and the next counterclockwise edge at each vertex. We can find the edges enclosing a given face, or edges and faces adjacent to a given vertex in time proportional to the output. In particular, in constant time we can find the face (or Voronoi region) on the 'other side' of an edge.

Similar data structures are the Winged Edge [Ba75] and a slightly more general one, the Quad-Edge data structure which is presented in [GS85] along with an implementation of an elegant divide and conquer algorithm for site Voronoi diagrams. In particular, this last data structure can represent any planar subdivision where the edges are not necessarily line segments.

Chapter 3

The CWP-cp/cs problems for convex polygons

In this chapter, we describe solutions to the following restricted versions of the optimization problems described in Chapter 1. In the first problem the polygon P must be convex and in the second, the set of points is in fact the set of vertices of a convex polygon, giving them additional structure and thus removing the requirement of constructing the convex hull[1]. The approach taken also eliminates the need to compute Voronoi diagrams that require complex data structures and produces a simple algorithm for the convex case. We define formally the problems as follows:

> **Problem CWP-cp (Closest Wedge Placement for a convex polygon).** Given a convex polygon P with n vertices and a wedge W with fixed angle ω, apex O, determine the position of W relative to P, allowing translations and rotations, such that $d(O, P)$ is minimized and P is contained inside W. Determine also the point of P that realizes the minimal distance to O.
>
> **Problem CWP-cs (for Set of vertices of a Convex polygon).** Same setting as CWP-cp problem but we minimize $\min\{d(O, v) : v \text{ vertex of } P\}$,

[1] An earlier version of the material in this chapter appears in [To88]

and we assume that P is in standard form in the sense that no three consecutive vertices are collinear.

The following notation is used:

$d(O, P)$ is the Euclidean distance between O and the polygon P (*i.e.* $d(O, P) = \min\{d(O.x) \cdot x \in P\}$.) We will similarly use $d(O, S)$ for a point set S.

$CIRCLE(O, r)$ represents the circle centered at O with radius r.

$\widehat{ab}$ is the arc extending from point a to point b and is part of a circle whose center is given by the context. A *subarc* of $\widehat{ab}$ is a segment of $\widehat{ab}$.

$WEDGE(O, W_1, \omega)$ represents a wedge having O for apex, W_1 as one of its half-lines and the other half-line (denoted W_2) is counter-clockwise from W_1 forming an angle ω with W_1. Such a wedge will also be denoted (W_1, W_2)

3.1 Observations

The problems CWP-cp and CWP-cs become trivial when the wedge angle ω is greater than the largest internal angle of P since in that case we can place O at any vertex and the minimal distance is zero. We can therefore assume the contrary.

Also, for problem CWP-cp it is not enough to look only at the vertices of P for the closest point to O. Figure 3.1 illustrates the case where the closest point is on an edge of P. Similarly to Chazelle [Ch83], we define

a *placement* of the wedge W as a given position of W and P relative to one another.

A *stable* placement of the wedge $W = WEDGE(O, W_1, \omega)$ is a placement such that an edge of P lies on one of the wedge half-lines.

An *optimal* placement is a placement that solves the CWP-cp or the CWP-cs problem depending on the context.

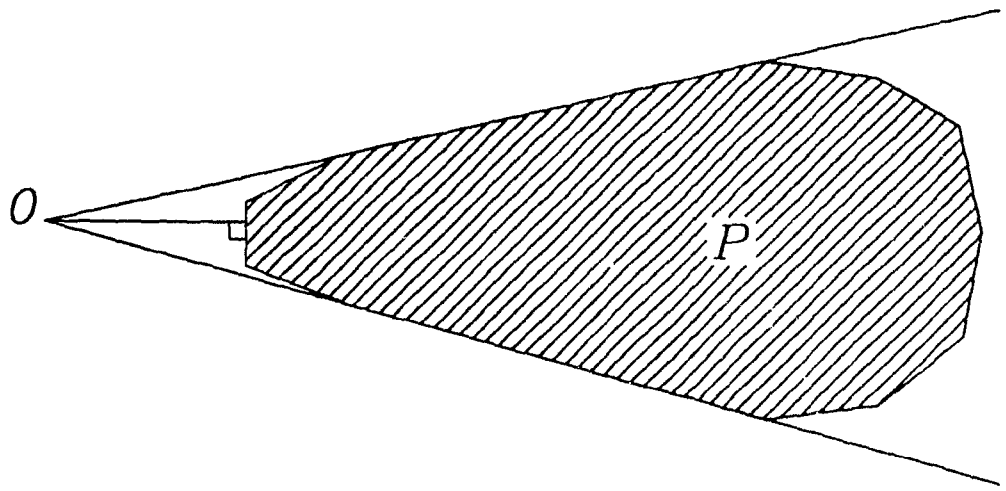

Figure 3.1: Closest point is on an edge of P.

A *contact point* is the point or vertex at which the polygon touches a given wedge half-line.

We note that it is not enough to look at stable placements to determine the closest position of P. Figure 3.2 illustrates a placement which is not stable but where P can't get closer to O. However by the following lemma, which applies to both definitions of optimality, only placements such that P is supported by each of the wedge half-lines need to be considered. This is what will be meant by *placement* in the remainder of the thesis.

Lemma 3.1 *Let P be a convex polygon and $W = WEDGE(O, W_1, \omega)$. Then the optimal placement of P and W is such that there is a vertex of P on each of the half-lines W_1, W_2 defining W.*

Proof There are 2 cases:

Case (a): There is one vertex of P in contact with W. Then we can rotate P about O in the direction away from the contact point, while all points of P remain at the same distance from O. We then fall into the next case.

Case (b): There are no vertices in contact with W. Then one can translate P towards O in the direction of the bisector of W. Clearly all points become closer to

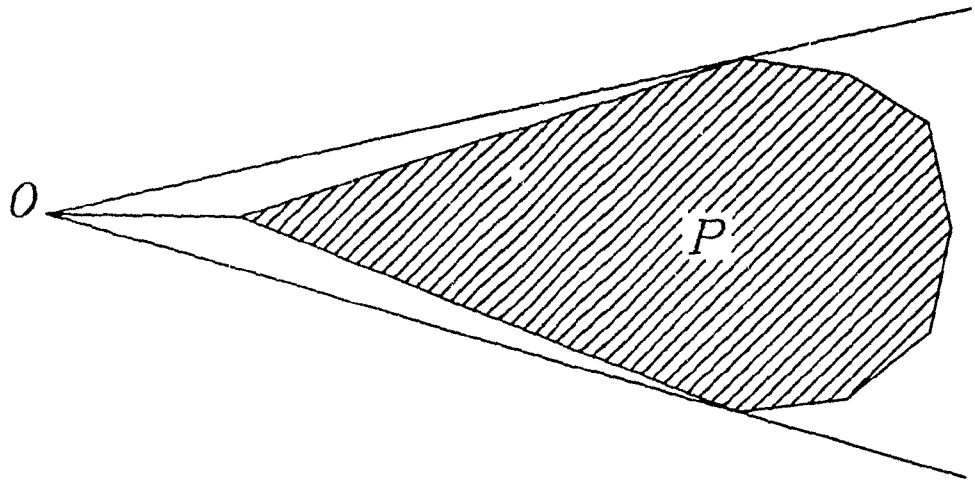

Figure 3.2: Non stable optimal placement.

O, which leads to a contradiction. ∎

Stable placements will be useful for the computation of the path followed by O as the wedge rotates around P. A stable placement can be represented by a triplet of points (a, u, v), where a is the position of the wedge apex O, and u and v are the contact vertices of P. We will need a starting stable placement, which can be computed in $O(\log n)$ time with the following procedure.

Algorithm STABLE-PLACEMENT

Input: Convex polygon P, angle ω.
Output: Stable placement of wedge W.

1. Choose an edge of P and place one directed line W_1 of W on it such that P is on the left of the line (check any vertex not on W_1.) Define a direction $W_2^{\perp}$ at a counter-clockwise angle of $\omega + \pi/2$ with W_1.
2. Using binary search, find in $O(\log n)$ time the vertex of P furthest in the direction of $W_2^{\perp}$ and position W_2 to pass through it.

3.1.1 Contact Vertex Pairs

We will next describe an algorithm for obtaining all the contact pairs and the stable placements of the wedge (and of O). The wedge will be rotated clockwise. The

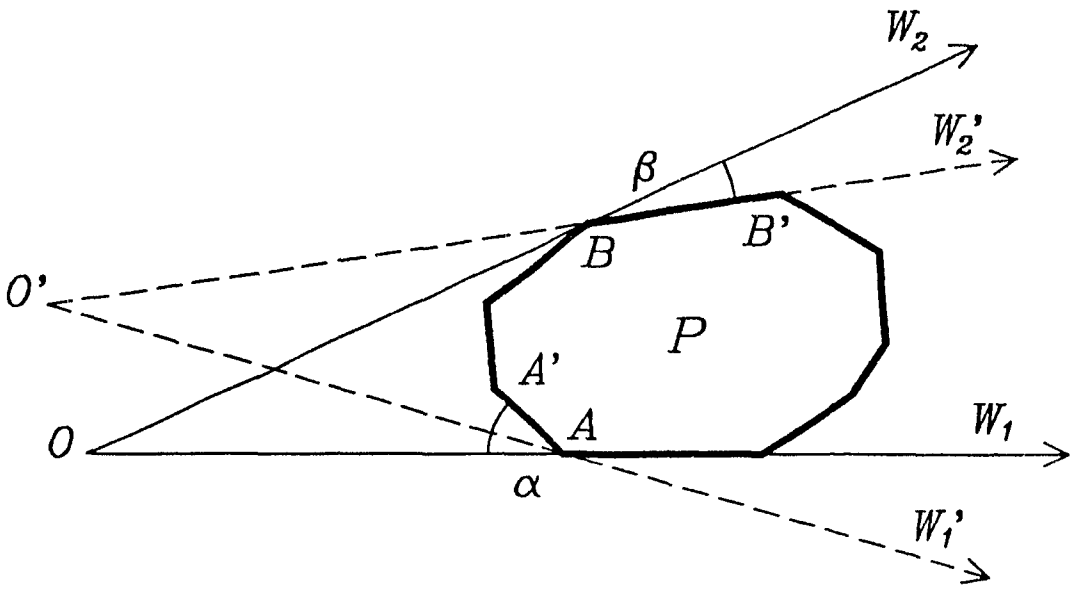

Figure 3.3: Finding the next stable placement.

algorithm represents yet another application of the 'rotating caliper' approach to solving geometric problems [To83a].

If there are more than two vertices on the wedge half-lines then for each half-line, we consider the ones that are the furthest clockwise on P to be the actual contact vertices. There are exactly two contact vertices at any time. Figure 3.3 illustrates the following algorithm. We define W as $WEDGE(O, W_1, \omega) = (W_1, W_2)$.

Algorithm ROTATING-WEDGE

Input: Convex polygon P, angle ω

Output: List of stable placements of W and corresponding contact pairs

1. Find a stable placement using the preceding algorithm. This gives two contact vertices A and B of P, and an initial placement of O.
2. Let A' and B' be the next vertices of P clockwise from A and B respectively. Let α be the angle formed by W_1 and the edge $\overline{A'A}$, and $\beta = \angle(\overline{B'B}, W_2)$.
3. If $\alpha = \beta$ or the internal angle at B is less than ω, the next contact pair will be (A', B'), and the new wedge $W' = (W_1', W_2')$ will be such that $\overline{BB'}$ is on W_2'. Let $(A, B) \leftarrow (A', B')$.

Else if $\alpha > \beta$, the next contact pair is (A, B'), and the new wedge W' will be such that $\overline{BB'}$ is on W_2'. Let $(A, B) \leftarrow (A, B')$.

Else if $\alpha < \beta$, the next contact pair is (A', B), and the new wedge W' will be such that $\overline{AA'}$ is on W_1'. Let $(A, B) \leftarrow (A', B)$.

4 Reposition the wedge, record the pair and the placement of O and repeat from step 2 until the initial placement of step 1 is reached.

There are $O(n)$ contact pairs since each edge determines two stable placements. During the execution of the above algorithm, W_1 lies on successive edges of P. In step 4, we therefore return to the original placement of W.

This algorithm runs in linear time, since all stable placements are traversed exactly once. Note that there can be up to $2n$ contact pairs as in the case of a regular n-gon and an appropriate angle ω.

We will now characterize the set of all placements of O for a given polygon P.

3.2 The Path of the Apex

In [Ch83] the placement of P with respect to two edges of a fixed polygon enclosing P is examined. This enclosing polygon can be considered as forming a wedge. Here however, this point of view with P moving leads to an overly complex formula for the distance from O to P as a function of their relative angle. We look therefore at problem CWP-cp from a different point of view and we examine the path that O follows as the wedge moves around the polygon in a continuous fashion. The problem requires solving the following subproblem:

> We are given two vertices A and B of P each sliding on its own half-line, the two half-lines forming a wedge of fixed angle ω. If the wedge is subject to the restriction that it never intersects the interior of P, then what is the path of the apex O relative to P?

Lemma 3.2 *This path is an arc segment with the containing circle passing through A and B and of radius $d(A, B)/(2\sin\omega)$.*

Proof Follows directly from [Eu], book 3, proposition 26. ∎

Given the points A and B and the angle ω, this arc can be represented for example by a structure containing the center of the circle and two points (or two angles) and a radius and can be obtained in constant time.

The path of O will be a chain of circular arcs around the polygon. Each arc is determined by a pair of contact vertices and its endpoints are two consecutive stable placements of O, as given by the ROTATING-WEDGE algorithm. If the internal angle at some vertex is smaller than ω, O will touch P at that vertex. If $\omega = \pi$ the path will be equal to P. Due to its shape (see Figure 3.6),

> the closed chain of arcs determined by the ROTATING-WEDGE scan about P will be called the *cloud* of P and denoted $CLOUD(P,\omega)$.

3.2.1 Obtaining the Minimum Distance

To solve problem CWP-cp we need to determine the *closest pair* of points, one on an edge of P and one on $CLOUD(P,\omega)$, such that their distance is as small as possible. The following will be for a given pair of contact vertices A and B.

> For a given arc of $CLOUD(P,\omega)$, we define the part of the polygonal chain of P that is between the contact vertices A and B, and on the same side of the line AB as the arc, as the *inner chain.*
>
> Let a set S of points be *visible* from a given point q, where q and S are not in the interior of P, if for all $s \in S$, $\overline{qs}$ does not intersect the interior of P or the wedge half-lines.

We then have the following lemmas.

Lemma 3.3 *For a given placement of O, the closest point to O in the CWP-cp optimal placement must be on the inner chain.*

Proof For all positions of O on the arc determined by two contact points, the inner chain is visible from O. ∎

Hence the edges of P on the inner chain are the only candidates for the closest pair.

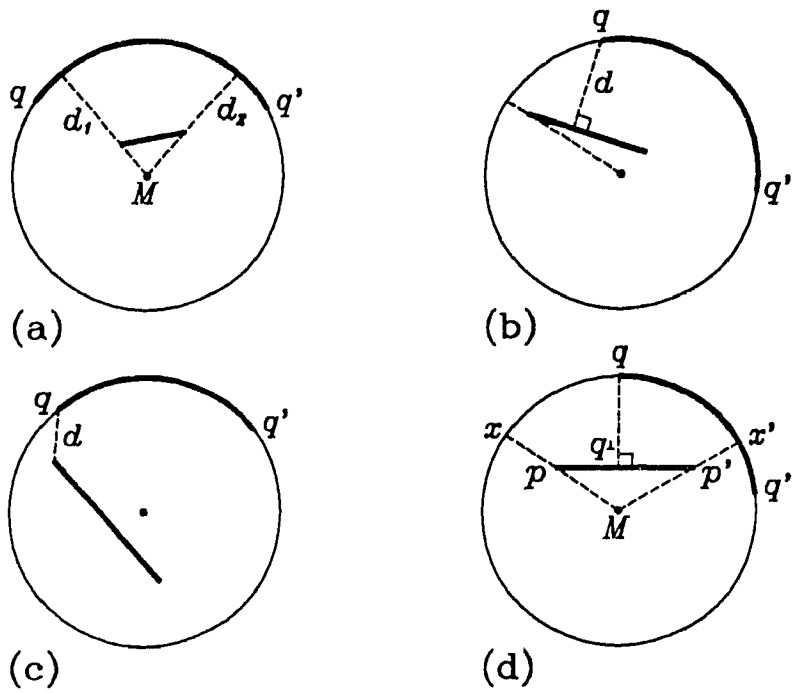

Figure 3.4: Possible closest pairs.

Lemma 3.4 *The edges of P on the inner chain are inside the circle determined by the arc and passing through A and B.*

Proof The inner chain is by convexity on the same side of line AB as O and is inside all triangles qAB for all q on the arc. But qAB is inside the circle (which passes through q, A and B). ∎

There are various situations to consider for the closest pair, depending on the position of the edge relative to the arc, and the size of the arc. In Figure 3.4(a), the closest pair is on a ray originating from the center M of the circle. In (b), we must consider the perpendicular projection of q onto the edge and in (c), the closest pair is an arc endpoint and a vertex.

Refer to Figure 3.4(d) for the following lemma.

Lemma 3.5 *Let p be a fixed point inside $CIRCLE(M, r) = C$ and q on C. Then $d(p, q)$ is unimodal as q travels along C making one turn. It has one minimum at*

$q = x$ and one maximum at $q = y$, where x is the closest of the two intersection points of C and the line passing through p and M, and y is the other intersection point.

Proof It is clear that x and y realize the minimum and maximum distance to p, respectively. Let q_1, q_2 be two points of C on the same side of the diameter $\overline{xy}$, such that $|\angle xMq_1| < |\angle xMq_2|$. Then p is on the same side of the perpendicular bisector of $\overline{q_1q_2}$ as q_1. Hence $d(q_1,p) < d(q_2,p)$ with equality only when $q_1 = q_2$ or $p = M$. ∎

Corollary 3.6 *Let p, x and C be as in Lemma 3.5. The closest point to p of an arc $\widehat{qq'}$ on C is the point s such that $|\angle xMs|$ is minimized $(-\pi/2 < \angle xMs < \pi/2)$. s can be one of x, q and q', and can be determined in constant time.*

We also note that for an arbitrary point q and segment $\overline{ab}$, the point $c_{\overline{ab}}(q)$ and hence the distance $d(q,\overline{ab})$ can be determined in constant time.

Theorem 3.7 *The points realizing the minimum distance (and hence the distance itself) between an arc and a segment contained in the interior of the circle containing the arc can be obtained in constant time.*

Proof We consider the Voronoi diagram of the set $\{p,p',(p,p')\}$. It splits the arc into at most five subarcs of the form $q_i\widehat{q_{i+1}}$. To find $d(q_i\widehat{q_{i+1}},p)$ or $d(q_i\widehat{q_{i+1}},p')$ we apply Corollary 3.6 on the subarcs in $V(p)$ and $V(p')$ respectively. To find $d(q_i\widehat{q_{i+1}},\overline{pp'})$ for the subarcs in $V((p,p'))$, we notice that the endpoints of each subarc are closer to $\overline{pp'}$ than any other point of this subarc, therefore only $d(q_i,\overline{pp'})$, $q_i \in V((p,p'))$ remain to be checked. Thus the number of point pairs whose distance must be calculated is constant. ∎

This theorem gives rise to a constant time algorithm for determining the distance between an edge and an arc positioned as specified in the statement of the theorem. This algorithm will be refered to as DAE in the sections to follow. Also, a constant time algorithm that determines the distance between a vertex of P and an arc arises from Corollary 3.6, and will be called DAV.

3.3 Algorithms CWP-cp and CWP-cs

Before presenting the solution to the CWP-cp/cs problems, we first establish a result that helps avoid repeated work in our algorithms, when determining the minimum distance between the path of the wedge apex and the polygon. We need some definitions:

> A point d on $CLOUD(P,\omega)$ is *between* a and b on the cloud if, starting at a and traversing the cloud clockwise, we reach d before b. This fact will denoted by the expression $a < d < b$.
>
> Let $v_0, \ldots, v_m$ be the inner chain with vertices listed in clockwise order on the polygon P. It is visible from every point of $\overset{\frown}{ab}$ where $\overset{\frown}{ab}$ is the set of positions of O for which v_0 and v_m are the contact vertices of W.
>
> We will also use the abbreviation $c(a)$ to mean $c_P(a)$.

We can now say that p is *before* q on the inner chain, if we encounter p before q while traversing the inner chain from v_0 to v_m. This fact is denoted by $p < q$ or $p \leq q$ if we can have $p = q$. (p and q are not necessarily vertices of P.)

We will show that for a point a on $CLOUD(P,\omega)$ moving in a given direction, $c(a)$ also moves in the same direction without backtracking. This will be similar to Theorem 2.3 for Voronoi regions in Chapter 2.

Lemma 3.8 *Let a be a fixed point outside a convex polygon P, and p a variable point on the boundary of P visible from a. Then the function $D(p) = d(a,p)$ is unimodal, with the minimum occurring at $p = c(a)$, as p moves counterclockwise along the boundary.*

Proof For each edge, by the triangle inequality $D(p)$ decreases strictly as p approaches $c(a)$ and is a continuous function. ∎

Here $\overset{\frown}{ab}$ will be a fixed arc of $CLOUD(P,\omega)$, determined by the contact vertices v_0 and v_m and the circle is that which contains this arc and intersects the contact vertices as described by Lemma 3.2.

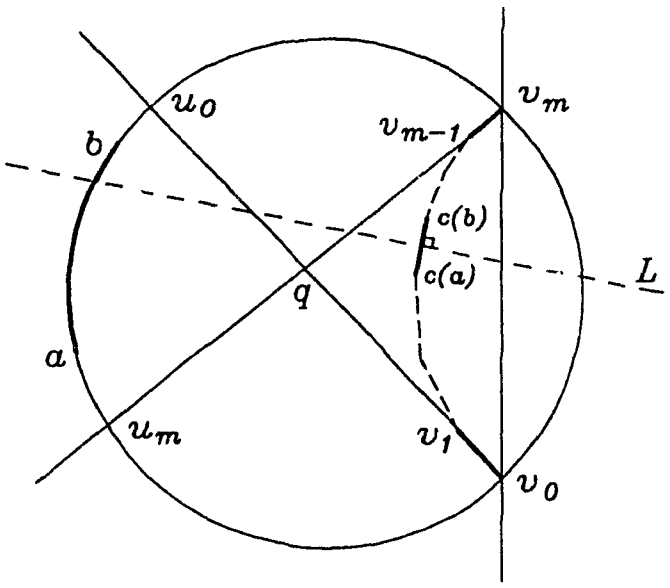

Figure 3.5: Showing how $c(a)$ moves in the same direction as a.

Lemma 3.9 *The arc $\widehat{ab}$ is in the region of the circle delimited by the lines extending $\overline{v_0v_1}$ and $\overline{v_{m-1}v_m}$, and intersecting the circle at u_0 and u_m, respectively. That is $u_m < a < b < u_0$. Moreover, $q = \overline{v_0u_0} \cap \overline{v_mu_m}$ is in the interior of the circle.*

Proof See Figure 3.5. If O was placed outside $\widehat{u_mu_0}$ then one of the wedge half-lines would intersect the interior of P. Since v_1 and v_{m-1} are in the interior of the triangle Ov_0v_m, for all $O \in \widehat{u_mu_0}$, the lines extending $\overline{v_0v_1}$ and $\overline{v_{m-1}v_m}$ intersect at q, and q is in the interior of the circle. ∎

Note also that the inner chain is contained in triangle v_0qv_m, where $q = \overline{v_0u_0} \cap \overline{v_mu_m}$.

Lemma 3.10 *Given an arc $\widehat{ab}$ of CLOUD(P,ω) with $u_m < a < b < u_0$, then we have $c(a) \leq c(b)$.*

Proof Assume $c(a) \neq c(b)$. $c(a)$ and $c(b)$ are inside triangle v_0qv_m. Let L be the perpendicular bisector of $\overline{c(a)c(b)}$. With respect to L, the point a must be on the side of $c(a)$, and b on the side of $c(b)$; otherwise, we get a contradiction.

We want to show that L crosses $\overline{v_0v_m}$ (see Figure 3.5.) Assume the contrary. Since L crosses $\widehat{ab}$ and $\widehat{u_mu_0}$, L cannot intersect $\overline{u_0v_0}$ or $\overline{u_mv_m}$ and hence does not enter triangle v_0qv_m. But the midpoint of $\overline{c(a)c(b)}$ through which L passes is in this triangle, which contradicts our assumption.

Therefore L intersects $\overline{v_0v_m}$. Since L cannot cross the inner chain twice, L separates the arc $\widehat{ab}$ and the chain into two pieces. Thus, traversing the inner chain starting at v_0, we must encounter $c(a)$ before $c(b)$. Hence $c(a) \leq c(b)$. ∎

Lemma 3.10 is valid for any subarc of $\widehat{ab}$. Therefore, if d is on $\widehat{ab}$, then $c(a) \leq c(d) \leq c(b)$; *i.e.* the point of P of the closest pair of points between $\widehat{ab}$ and P can always be found between $c(a)$ and $c(b)$ on the inner chain.

Theorem 3.11 *If $a, b \in CLOUD(P, \omega)$ and $a < b$, then $c(a) \leq c(b)$.*

Proof If a and b are on the same arc, by Lemma 3.10 we are done. Else we have the chain of arcs (clockwise): $a_i\widehat{a_{i+1}}, a_{i+1}\widehat{a_{i+2}}, \ldots, a_{j-1}\widehat{a_j}$, where the a_i are successive stable placements of O and $a \in a_i\widehat{a_{i+1}}, b \in a_{j-1}\widehat{a_j}$. Then by Lemma 3.10 we have

$$c(a) \leq c(a_{i+1}) \leq c(a_{i+2}) \leq \ldots \leq c(a_{j-1}) \leq c(b). \quad \blacksquare$$

In this way the set of arc endpoints partitions P into chains or sections that we will call $c(\widehat{ab})$ for each $\widehat{ab}$, extending our notation. So for a given arc $\widehat{ab}$, the algorithm for CWP-cp has to look for the closest pair only on the edges of $c(\widehat{ab})$.

Lemma 3.8 ensures that the distance between an arc endpoint b on $CLOUD(P, \omega)$ and p on P reaches a minimum exactly once (at $c(b)$) as p moves along the boundary of P. This fact along with Theorem 3.11 allows us to determine $c(b)$ given $c(a)$ without backtracking. This can be done by iteratively checking the distance between b and edges succeeding $c(a)$ until this distance increases. The preceding edge then contains $c(b)$, and we can advance a to b. Doing this for a linear number of arc endpoints a, we can obtain all the $c(a)$'s in linear time in one scan around the boundary of P.

For arc endpoints a_i, we can introduce new vertices at the $c(a_i)$—no more than $2n$ new vertices—to simplify the complexity analysis. We can then claim that the algorithm never examines two edges twice, and hence is linear.

Algorithm CWP-cp.

Input: Convex polygon P, angle ω.
Output: CWP-cp optimal placement of wedge apex O and $c(O)$.

1. Let $\mathcal{R}$ be an unordered set of point pairs, initially empty.
2. Perform a ROTATING-WEDGE scan around P to obtain the initial placement of apex and the list of stable placements and corresponding contact pairs, (a_i, u_i, v_i), $i = 0, \ldots, t-1$, $t \in O(n)$.
3. Find $c(a_0)$ for a_0 the initial placement of apex given by step 2.
4. For each stable placement (a_i, u_i, v_i):
   4.1. Compute the arc $a_i\widehat{a_{i+1}}$ of the circle as described in Lemma 3.2 (indices are taken modulo t.)
   4.2. Determine $c(a_{i+1})$ as described above by traversing the inner chain clockwise from $c(a_i)$.
   4.3. For each edge on inner chain between $c(a_i)$ and $c(a_{i+1})$ (including those containing $c(a_i)$ and $c(a_{i+1})$) do:
      Using algorithm DAE, get the closest pair of points on $a_i\widehat{a_{i+1}}$ and the current edge and add it to $\mathcal{R}$.
5. Obtain the closest pair (O, p) of the pairs of points in $\mathcal{R}$, and report the corresponding wedge placement.

The comment preceding the algorithm establishes the following result.

Theorem 3.12 *Problem CWP-cp for a convex polygon P with n vertices can be solved in $O(n)$ time.*

Figure 3.6 shows an example of $CLOUD(P,\omega)$ for a wedge with angle $\omega = \pi/2$. The optimal placement of the wedge apex is at O, and the closest point on P to O is v, a vertex of P. This placement is therefore also a solution to CWP-cs.

We have a very similar result for CWP-cs. Let $cv(a)$ be the closest visible vertex to a point a outside P. It is the only vertex that is candidate for being part of the closest pair, as is shown in the next theorem. We will prove a fact similar to Lemma 3.3 for the closest vertex in the optimal placement of W and P, as defined by problem CWP-cs.

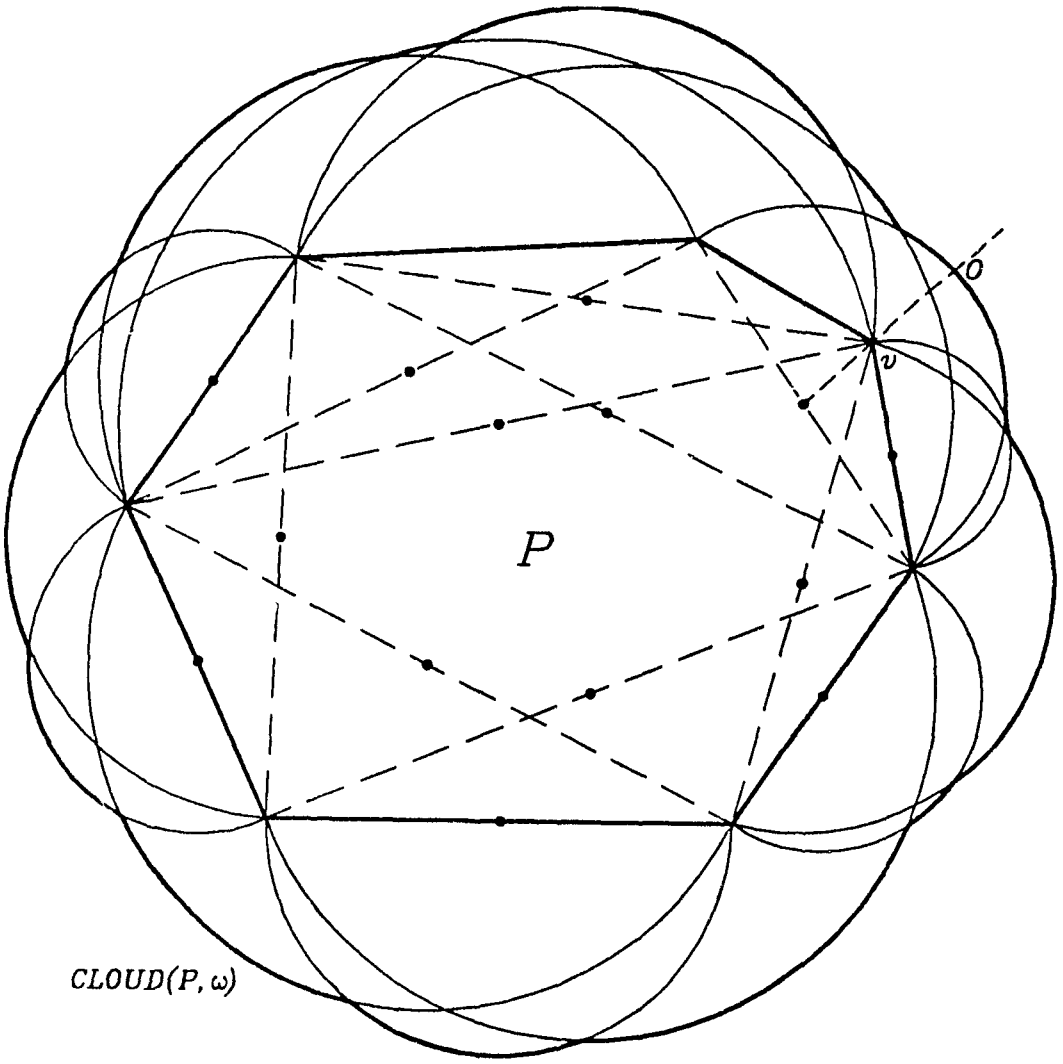

Figure 3.6: Example of $CLOUD(P, \pi/2)$.

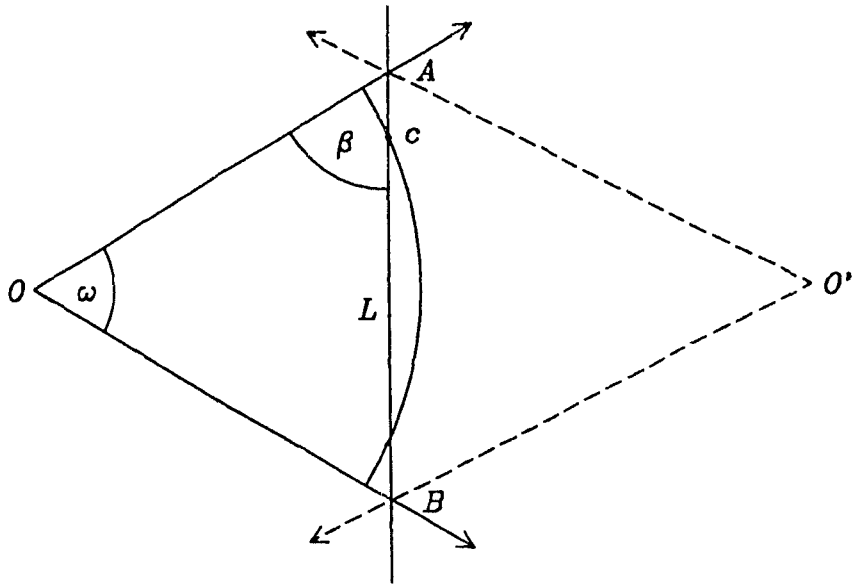

Figure 3.7: Theorem 3.13: Case $\omega < \pi/2$.

Theorem 3.13 *Given a placement of polygon P and wedge $W = WEDGE(O, W_1, \omega) = (W_1, W_2)$ optimal for problem CWP-cs, then the vertex closest to O is on the inner chain.*

Proof Suppose the closest vertex c is not on the inner chain. Then all vertices must be on the outside of the circle $CIRCLE(O, r)$, where $r = d(O, c)$. Let L be the line of support of P passing through c, with P on the same side of L as O. Let A be a point of intersection of L with W, say on W_2. The internal angle β formed by L and W_2 (on the side of O) must be less than $\pi/2$, since c is on the circle, and (by Lemma 3.1) A is outside of the circle.

Case $\omega < \pi/2$ (see Figure 3.7):

Since $\omega < \pi/2$, we must have another intersection point B on the other half-line. There must be vertices at A and B, otherwise we could place a wedge W' with apex O' such that $d(O', c) < d(O, c)$. W' is a copy of W reflected with respect to L. But W' is not in contact with P, since P is contained in triangle AOB. By Lemma 3.1, this is not the optimal placement. But, by the definition of CWP-cp, P has no three consecutive collinear vertices, which contradicts our assumptions, and proves

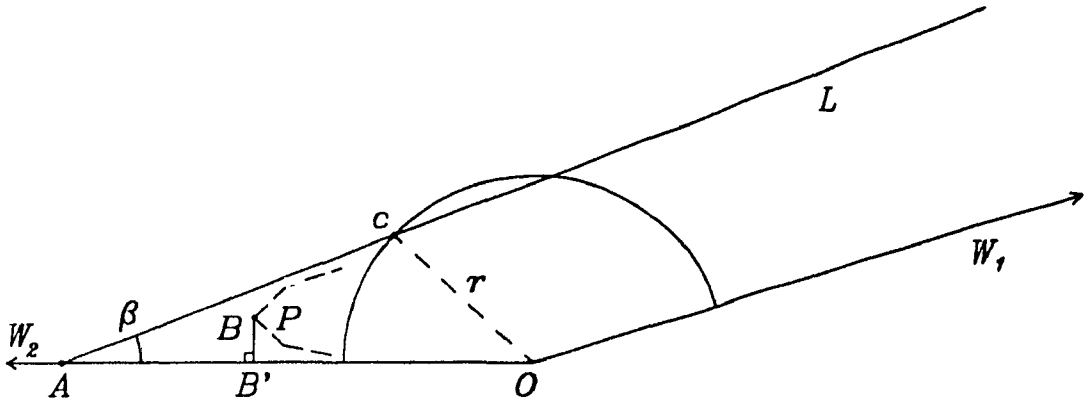

Figure 3.8: Theorem 3.13: Case $\omega \geq \pi/2$.

the theorem for $\omega < \pi/2$.

Case $\omega \geq \pi/2$ (see Figure 3.8):
In this case the supporting line does not necessarily intersect both wedge half-lines. Note that $\beta \leq \pi/2$ implies that $\beta \leq \omega$.

All the vertices of P must reside inside the wedge, and in the half-space delimited by L and containing O. There must be at least one vertex of P in triangle AOc and outside $CIRCLE(O, r)$ different from c (in particular on $\overline{OA}$), otherwise we contradict Lemma 3.1. Of the vertices of P in triangle AOc, take the one that is the furthest in the direction of W_2 and call it B. Call B' its orthogonal projection on W_2. Then we can place a wedge $WEDGE(B', \overrightarrow{B'O}, \omega)$ at B' with $d(B', B) < r$, so there is a placement of the wedge that brings O closer to the set of vertices of P. This contradicts our assumption and proves the theorem for all angles $\omega \leq \pi$. ∎

Lemma 3.14 *The closest vertex to arc $\widehat{ab}$ is a vertex adjacent to an edge of $c(\widehat{ab})$.*

Proof The closest visible vertex must be on the inner chain by Theorem 3.13. Consider the Voronoi regions of the edges of P. We noted in Chapter 2 that they are delimited by the set of normals stemming from the adjacent vertices. By definition, $\widehat{ab}$ intersects the Voronoi regions of the edges (and adjacent vertices) in $c(\widehat{ab})$, and no other regions. Therefore the only vertices necessary to consider are those adjacent to edges of $c(\widehat{ab})$. ∎

This means that $cv(q)$, $q \in \widehat{ab}$ can reach outside of $c(\widehat{ab})$, but not beyond another vertex. To obtain an algorithm for CWP-cs, we modify the preceding algorithm by replacing the application of DAE on the current edge by an application of DAV on the vertices of the current edge.

Theorem 3.15 *Problem CWP-cs for a point set S with n points that are the vertices of $CH(S)$ can be solved in $O(n)$ time.*

Proof The preceding lemma guaranties the correctness of algorithm CWP-cs and the running time remains linear. ∎

Chapter 4

The CWP-p and -s problems

We will now consider the general version of the CWP problems and present algorithmic solutions. The precise statements of the problems follow.

> **Problem CWP-p (Closest Wedge Placement for a simple polygon).** Given a simple polygon P with n vertices and a wedge W with fixed angle ω, apex O, determine the position of W relative to P, allowing translations and rotations, such that $d(O,P)$ is minimized and P is contained inside W. Determine also the points of P that realize the minimal distance to O

> **Problem CWP-s (for a Set of points).** Given a set S of n points and a wedge W with fixed angle ω, apex O, determine the position of W relative to S, allowing translations and rotations, such that $d(O,S)$ is minimized and S is contained inside W. Determine also the points of S that realize the minimal distance to O.

We will assume that the only extreme points of S are the vertices of $CH(S)$, and that P is in standard form. These assumptions will simplify the proofs below, but are not necessary for achieving an $O(n \log n)$ time bound.

We first note that the wedge lines will be tangent to the convex hull of P or S, so the results concerning convex polygons of Chapter 3 apply here to $CH(P)$ and $CH(S)$, i.e. $CLOUD(P,\omega) = CLOUD(CH(P),\omega)$ and similarly for S.

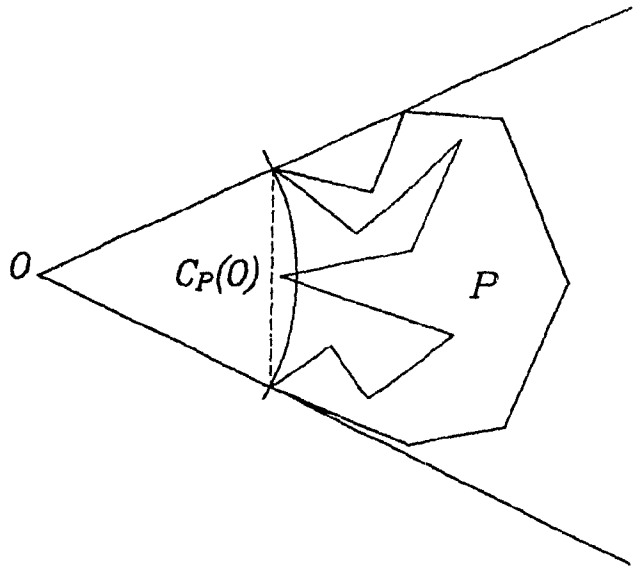

Figure 4.1: Closest point of P behind a hull edge.

To solve the first problem, which the case of a simple polygon P, we cannot merely scan the edges of P and obtain the successive $c(a)$ for cloud arc endpoints a, since the polygonal chain can wander anywhere inside $CH(P)$. As an example, we note the case where the optimal placement is such that $c(O)$ is inside $CH(P)$ as in Figure 4.1. A similar situation occurs in the case of a point set.

For a given arc of $CLOUD(P,\omega)$, we will use the Voronoi diagram of the set of edges and vertices of P to determine which edges of P to examine when searching for the closest point on P. Similarly, we will use the Voronoi diagram of S for the second problem.

The convex hulls $CH(P)$ and $CH(S)$ mentioned above can be obtained in linear time [MA79,Le83,GY83] and $O(n \log n)$ time [Gr72], respectively. However, the convex hull of S can be obtained from its Voronoi diagram in $O(n)$ time, as described in [PS85, p. 215]. The algorithm in [PS85] traverses the unbounded regions of $VOR(S)$, finding the convex hull vertices as illustrated in Figure 4.2. By Lemma 2.4, we can adapt this algorithm to construct the Voronoi diagram of the exterior of a simple polygon, since there exists a similar correspondence between convex hull

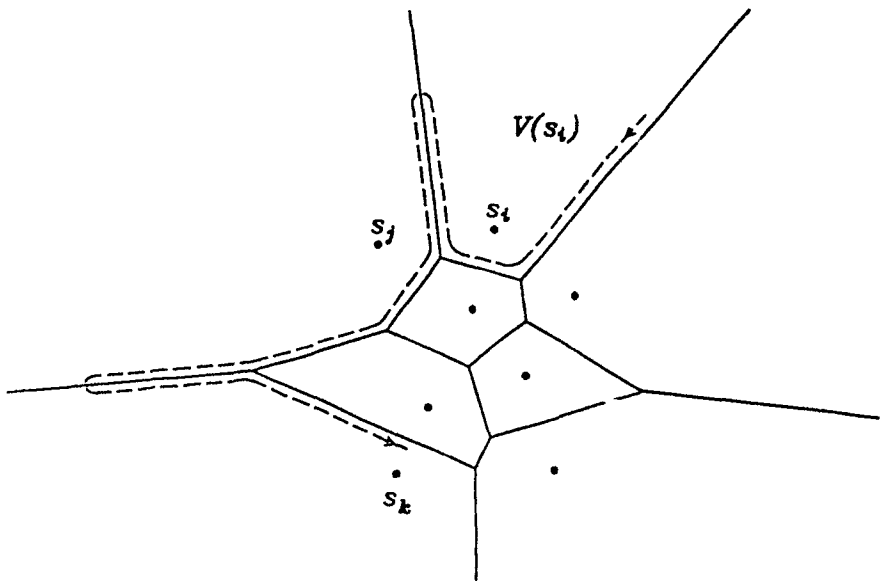

Figure 4.2: Obtaining the convex hull from the Voronoi diagram.

elements and unbounded Voronoi regions.

4.1 Properties of Voronoi diagrams and $CLOUD(P,\omega)$

Let H be $CH(P)$ or $CH(S)$ according to the problem considered. We will need a result similar to Lemma 2.6 for sets of points.

Lemma 4.1 *Let e an edge of $CH(S)$. The e-promontory of the Voronoi diagram of S is contained in the strip corresponding to e.*

Proof Identical to that of Lemma 2.6. ∎

We can now present a result that applies to both kind of Voronoi diagrams. It will enable us to efficiently search the diagrams for intersections with $CLOUD(H,\omega)$.

Below, we will assume w.l.o.g. that e is parallel to the x axis, and we define the 'vertical' direction to be in the direction of the y-axis. We also assume that P and S are below e (their points have smaller y coordinates).

Theorem 4.2 *Let $e = \overline{v_i v_j}$ be an edge of H. If the interior of the e-strip is not empty, then the e-promontory is an embedding of a tree in the plane such that (a) each node is a Voronoi edge, (b) a tree node N_i is a child of tree node N_j if their corresponding Voronoi edges E_i and E_j are adjacent, and if E_i is below E_j. The root of this tree is the node corresponding to $B(v_i, v_j)$, a half-line, and the leaves correspond to the (parts of) Voronoi edges ending with cutpoints.*

Proof Lemmas 2.6 and 4.1 guarantee that the Voronoi edges intersect the strip boundary only at e. First, we note that a Voronoi diagram is a planar graph. We now show it has no cycle outside of H. There are no Voronoi elements outside H, and since the only way to get a cycle is to have an entire Voronoi region (along with its element) in the exterior of H, the e-promontory must be a tree. By Lemma 2.2, $B(v_i, v_j)$ is the only infinite bisector in the interior of the strip since, sufficiently far from H, for $x \in B(v_i, v_j)$, $CIRCLE(x, d(x, v_i))$ is empty and intersects only v_i and v_j.

$B(v_i, v_j)$ will be taken as the root of the tree. We now show that any path from $B(v_i, v_j)$ is monotone with respect to the vertical direction and this will establish the theorem.

Assume the contrary and refer to Figure 4.3(a). Then there is a local minimum in the y coordinate, say at M. The set of points visible from M is above the horizontal line L through M. Since the path is the boundary of two Voronoi regions whose elements are below L, we get a contradiction with Lemma 2.1. ∎

In fact, since the set of elements is connected, the only kind of cycle in the exterior of P is one that contains either a vertex or an edge of P.

We can use the terms leftmost and rightmost child of a Voronoi edge, as well as the father of a Voronoi edge.

Let a *v-path* be a path from the root of an e-promontory to a cutpoint.

We will need some additional properties of $CLOUD(H, \omega)$. Since the contact points of W are on H, the inner chain of Chapter 3 will be a chain of H.

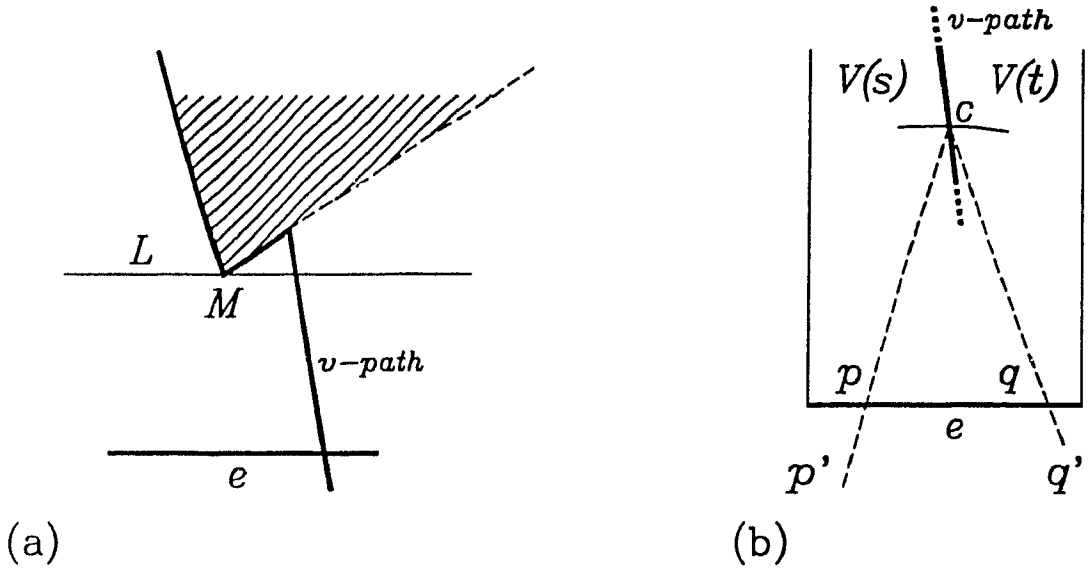

Figure 4.3: Showing monotonicity of a v-path and its relationship with cloud arcs.

Lemma 4.3 *The interior of a circle C defined by two contact points A and B as in Lemma 3.2, contains no arc of $CLOUD(H,\omega)$ in the half-plane delimited by line AB and containing the inner chain.*

Proof Let $\overset{\frown}{ab}$ be an arc of $CLOUD(H,\omega)$ such that $\overset{\frown}{ab} \subset INT(C)$. Let c be any point of $\overset{\frown}{ab}$. Then $\angle(\overline{cA},\overline{cB}) > \omega$. If $\overline{cA}$ and $\overline{cB}$ do not intersect $INT(H)$ we have a wedge of angle greater than ω tangent to and containing H. Otherwise, to make $\overline{cA}$ and $\overline{cB}$ tangent to H, we must translate A or B along the inner chain corresponding to C. However this increases $\angle(\overline{cA},\overline{cB})$ and we again get a wedge of angle greater than ω. ∎

Lemma 4.4 *A v-path does not intersect a circle (or arc) more than once.*

Proof Refer to Figure 4.3(b). Let c be any point of the v-path and let $B(s,t)$ be the bisector c is on. Then c is visible from a point p' in $V(s) \cap INT(H)$ and from q' in $V(t) \cap INT(H)$ by Lemma 2.1. Let e be the convex hull edge determining the strip in which the v-path is contained. Then by Lemmas 2.6 and 4.1, $\overline{cp'}$ and $\overline{cq'}$ cannot intersect the normals of e, so they must intersect e itself, say at p and q respectively. Hence pqc is a triangle containing the section of the v-path below c.

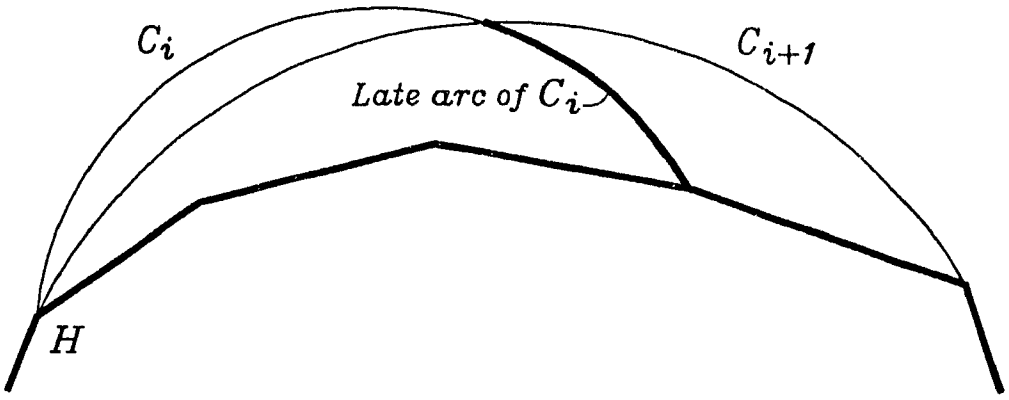

Figure 4.4: A late arc of a circle is contained inside the next circle.

We can place c on an arc $\widehat{ab}$ of $CLOUD(H,\omega)$ that is on a circle C defined by two contact vertices. Then by Lemma 3.4, $\overline{pq}$ is inside C. Hence pqc is inside C, and the v-path does not intersect C below c.

Assume now that the v-path intersects C above c, say at c'. Then applying the preceding discussion to c' instead of c produces a contradiction. Therefore the v-path cannot intersect the circle above c. ∎

Corollary 4.5 *A v-path does not intersect $CLOUD(H,\omega)$ more than once.*

Proof By Lemma 4.3, an arc $\widehat{ab}$ of $CLOUD(H,\omega)$ that is on circle C is contained in the interior of no other circle of the cloud, and no other arc of the cloud is inside C. Hence a v-path cannot cross $CLOUD(H,\omega)$ twice. ∎

> Define a *late* arc as the largest subarc of the circle, C, that is clockwise of its cloud arc, and up to the nearest intersection with the polygon, as in Figure 4.4.

The theorem below will establish the structure of the set of circles and show the correctness of step 5 of the algorithm in the next section.

Theorem 4.6 *When traversing a v-path up towards the root, the order in which late arcs are encountered is the same as the order in which they are encountered when traversing $CLOUD(H,\omega)$ clockwise.*

Proof By looking at algorithm ROTATING-WEDGE, we notice that successive stable placements have both contact points moving clockwise on the convex polygon. Any pair of circles can intersect at only two points, and successive circles C_i, C_{i+1} of $CLOUD(H,\omega)$ intersect already on $CLOUD(H,\omega)$. Both circles contain in their interior the last clockwise edge of the inner chain of C_i (unless the cloud intersects H on C_i, in which case we are done.) Therefore C_{i+1} cannot intersect C_i on the late arc of C_i (except at the endpoints of the late arc); hence the late arc of C_i is contained inside C_{i+1}. This can be seen in Figure 4.4. Finally, Lemmas 4.2 and 4.4 guarantee the ordering on the v-path. ∎

4.2 Algorithms CWP-p and CWP-s

In this section we present an algorithm for solving the CWP-p problem, and then show how a minor variation of it also solves CWP-s. As in Chapter 3, the algorithm for CWP-p will do a traversal of $CLOUD(P,\omega)$. For each Voronoi region encountered it considers the distance from the current arc to the element (edge or vertex of P) corresponding to the region.

> Given a vertex h of H, let the *left normal* at h, denoted $LNORMAL(h)$, be the normal at h that is orthogonal to the edge of H adjacent to and clockwise from h. For an arc A on circle C, let $LCARC(A)$ be the union of the cloud arc and the late arc on C.

We will assume that all infinite Voronoi edges join at infinity as in [PS85, p. 251], and that they have a dummy parent that is 'above' every Voronoi region. This is to simplify the description of steps 5.3 and 5.4, which find the next infinite Voronoi edge clockwise from the current edge. Alternately, the next infinite Voronoi edge can be found through a counterclockwise scan of the edges of the Voronoi region located to the right of the current infinite (directed) edge.

This can be accomplished in linear amortized time without altering the complexity of the algorithm. This is similar to the scan pictured in Figure 4.2.

Below we present a formal description of the algorithm followed by a detailed explanation and analysis.

Algorithm CWP-p

Input: Simple polygon P, angle ω.
Output: CWP-p optimal placement of O and $c_P(O)$.

1. Obtain $VOR(P)$.
2. Let $H = CH(P)$.
3. Obtain $CLOUD(H,\omega)$ and the list (a_i, u_i, v_i), $i = 0,\ldots,t-1$, of contact vertices using ROTATING-WEDGE algorithm.
4. [*Some initialization*]
   4.1. [*Good placement list*] $\mathcal{R} \leftarrow \emptyset$.
   4.2. [*Initial arc*] Let $A_i = a_i\widehat{a_{i+1}}$ be the arcs of $CLOUD(H,\omega)$ in clockwise order. Let h_0 be any vertex of H and traverse $CLOUD(H,\omega)$ to find k such that A_k intersects $LNORMAL(h_0)$. Let $i \leftarrow k$.
   4.3. [*Initial Voronoi element*] Let $z \leftarrow h_0$.
   4.4. [*Initial Voronoi edge*] Let E be the infinite Voronoi edge most counterclockwise in $V(h_0)$.
5. [*Do traversal*] Repeat:
   5.1. [*Follow leftmost v-path down to $LCARC(A_i)$*] While $E \cap LCARC(A_i) = \emptyset$ do: $E \leftarrow$ leftmost child of E.
   5.2. [*Find intersection of v-path and cloud*]
      5.2.1. [*Check distance*] Apply DAE or DAV, as appropriate for z, on A_i and z and get closest point pair (O,p). Add (O,p,u,v) to $\mathcal{R}$.
      5.2.2. If $E \cap A_i = \emptyset$ then:
         a. [*Go to next cloud arc*] $i \leftarrow (i+1) \bmod t$.
         b. While $E \cap LCARC(A_i) = \emptyset$ do: $E \leftarrow$ father of E.
         c. Repeat step 5.2.
   5.3. [*Advance to next Voronoi region*] Let z be the element corresponding to the newly entered Voronoi region on the other side of E.
   5.4. [*Find next initial Voronoi edge*] While E is not above $V(z)$ do: $E \leftarrow$ father of E.

5.5. $E \leftarrow$ next leftmost unvisited child of E.

5.6. If z is a vertex of H then [*skip to next left normal*]

repeat

5.6.1. [*Check distance*] Same as step 5.2.1.

5.6.2. [*Go to next cloud arc*] $i \leftarrow (i+1) \bmod t$.

until $A_i \cap LNORMAL(z) \neq \emptyset$.

until i reaches k after having been modified in step 5.

6. Find the smallest distance between points pairs (O,p) of the elements (O,p,u,v) of $\mathcal{R}$, and report the corresponding placement of O and $p = c_P(O)$.

More informally, the algorithm proceeds as follows. To begin we find $VOR(P)$ which takes $O(n \log n)$ time, $H = CH(P)$ which takes $O(n)$ time, and $CLOUD(H,\omega)$ which takes $O(n)$ time as discussed in Chapters 2 and 3. The sizes of $VOR(P)$, H and $CLOUD(H,\omega)$ are all linear in terms of n. Next we choose an arbitrary left normal (denoted $LNORMAL(h_0)$ above), and find in linear time its intersection with the cloud, by testing for intersection with each of its arcs. For each strip, the arc intersecting the left normal will be a starting arc. For ease of discussion, we will assume below that the left normal is vertical.

In step 4 some initialization is done. We will maintain a current Voronoi element z, which is a vertex of H at the start, and we initialize the current Voronoi edge to be the infinite Voronoi edge most counterclockwise in the current Voronoi region, $V(z)$. This can again be found in at most linear time given any edge of $V(z)$, which can be obtained from the data structure implementing the Voronoi diagram.

Step 5 performs the clockwise traversal of the cloud, and at the same time performs a variant of a depth-first search of the Voronoi diagram. Step 5 will be repeated until a full cycle around the cloud is made. Figure 4.5 illustrates this process. For each cloud arc A_i encountered, we will apply algorithm DAE if z is an edge, DAV otherwise, and find the closest point pair (O,p), $O \in A_i$, $p \in P$ (in steps 5.2.1 and 5.6.1).

First, the current Voronoi edge descends down the left-most v-path of the promontory until the late arc of the current cloud arc or the cloud arc itself is crossed, say at point q (step 5.1). This must occur by Lemma 3.4. Note that in the process we may

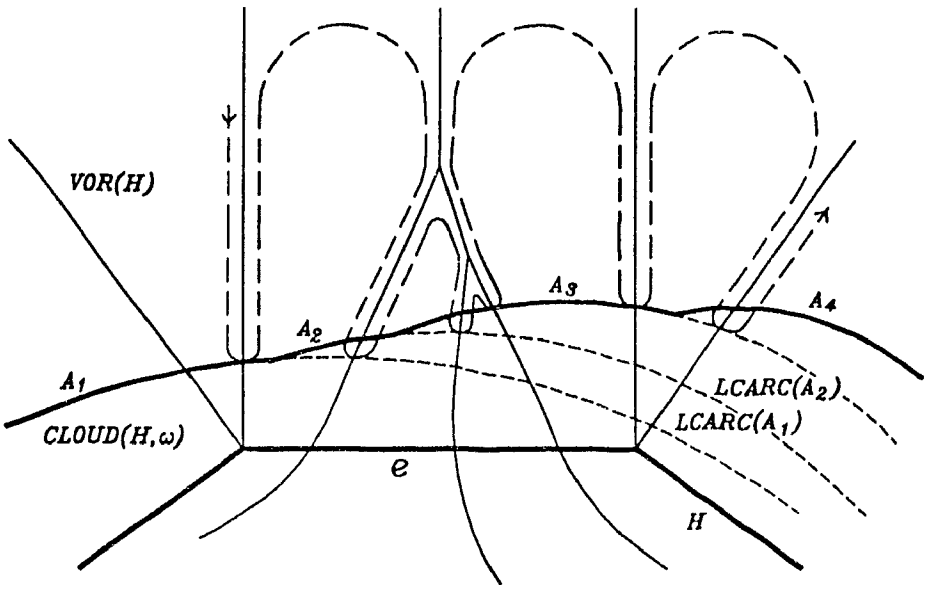

Figure 4.5: Finding the intersections of $CLOUD(H,\omega)$ and $VOR(P)$.

have crossed many late arcs corresponding to cloud arcs clockwise of the current one. If we have reached the corresponding cloud arc, we are done. Otherwise the intersection must occur at some cloud arc after (clockwise of) this one.

In step 5.2, we find the next intersection of $CLOUD(H,\omega)$ with $VOR(P)$ as follows. We scan the cloud clockwise and the v-path upwards from q. By Theorem 4.6, each late arc will intersect the v-path higher as we progress along the cloud until we reach the cloud arc that intersects the v-path. For each new $LCARC(\cdot)$, we climb the v-path until the point it crosses the v-path is reached (step 5.2.2), then we can advance the current arc. Thus the current arc A_i and the current Voronoi edge E must meet and this will be the only intersection by Corollary 4.5. Therefore step 5.2 must terminate and we never cross a Voronoi region twice.

At this point, the current cloud arc has crossed over to the next Voronoi region which is determined in step 5.3. Again, this can be done using the data structures described in Chapter 2. In steps 5.4 and 5.5, the current Voronoi edge is placed so that it is the highest Voronoi edge of the v-path that bounds the new Voronoi region on the right. We used the assumption that all infinite Voronoi edges join at infinity

to simplify the description of these steps. We are ready to start step 5 again and traverse this new region, unless we just reached an infinite region corresponding to a vertex v of P that is also on the convex hull. In this case step 5.6 brings the current arc to the left normal of the next strip. This is easy since we are guaranteed not to have any Voronoi edges in the interior of $V(v)$. Thus we will be ready to search the next promontory.

At each iteration of step 5, one or both of the following events occur: (a) the current arc crosses over to a new Voronoi region; (b) the current arc advances. Step 5 must therefore terminate. Also no cloud arc is traversed more than once. There is a linear number of regions and a linear number of arcs in $CLOUD(H,\omega)$. Moreover there is a linear number of Voronoi edges, and each is visited at most twice[1]. Step 5 thus runs in linear time.

Finally step 6 finds the solution to CWP-p among the linear number of good placements in $\mathcal{R}$ and also runs in linear time.

Theorem 4.7 *Problem CWP-p for a simple polygon P with n vertices can be solved in $O(n \log n)$ time. If we are given VOR(P), then CWP-p can be solved in $O(n)$ time.*

Proof By the discussion above, steps 2 to 6 run in linear time.

To obtain an algorithm for problem CWP-s for a set S of points, replace P by S in the above algorithm and its analysis. The discussion on algorithm CWP-p also applies to CWP-s since the facts on which it is based are valid for both types of Voronoi diagrams. However DAE will never be called since the only elements are points. In particular the comments pertaining to step 5.6 remain true: this step is just an aesthetic aid in the polygonal case, since a late arc coming out of a strip must intersect the left normal of the next clockwise strip (which is also a Voronoi edge). Step 5.6 now becomes essential in the case of point sets, since the leftmost v-path of the next strip does not necessarily intersect this late arc.

Thus follows our final theorem.

[1] Except the edge above each Voronoi region encountered, which is visited $s+1$ times, where s is the number of children of that edge. This number is linear over the whole diagram.

Theorem 4.8 *Problem CWP-s for a finite set S of n points can be solved in $O(n \log n)$ time. If we are given $VOR(S)$ then CWP-s can be solved in $O(n)$ time.*

Chapter 5

Conclusion and Further Research

We have presented efficient $O(n \log n)$ algorithms to obtain the optimal placement of a containing wedge for various types of sets. We take advantage of convexity to obtain the result in $O(n)$ time. For the CWP problems, the optimal placement was defined to be that which minimized the smallest Euclidean distance between the wedge apex and the set. We will mention other definitions for which the optimization problems are still open.

First, in the MiniMax-WP problem, we define the optimal placement as the placement minimizing the radius of the smallest circle centered at the wedge apex, and containing the set. In this problem, it is clear that only the vertices of the convex hull of the set are relevant. This problem is quite close to the smallest enclosing circle problem, which is also known as the *minimax facility location problem* [Sh77,BT85].

Following Shamos, we define the *furthest-point Voronoi diagram* (FPVD) of a point set $S = \{s_i : i = 1, \ldots, n\}$ to be the partition of the plane into unbounded convex 'polygons' $V(s_i)$, such that $V(s_i)$ is the locus of points further from s_i than from any other point in S. The size of this diagram is linear in terms of n.

Furthermore, we notice that given a general placement of the wedge, the point of the set that determines the radius is the point associated with the FPVD region

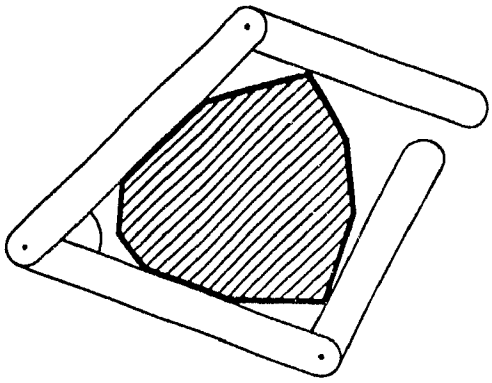

Figure 5.1: A type of robot hand.

containing the apex. It may be possible to take advantage of this fact to obtain an algorithm more efficient than the following naive $O(n^2)$ algorithm. For each arc of the cloud, examine the edges of the FPVD to determine which regions are intersected by the arc, and check the distance between the arc and the points associated with those regions.

Another problem arises when we define the quantity to be minimized as the area or perimeter of a triangle containing the set, and formed by the wedge half-lines and a segment whose endpoints are on the half-lines. For a right-angled wedge at a fixed position, these problems are solved in [Ch86]. This is a version of the smallest triangle enclosure problem that asks for the minimum triangular superset [KL85,Ch86], but with one fixed angle. It would be interesting to see if the optimal placements for the two wedge problems are related.

We may also want to measure the distance from the wedge apex to the contact points. The minimum or maximum of these is easily obtained in linear time. For some types of robot hands, this might be indicative of the size of the hand necessary to handle a given object (see Figure 5.1).

Finally, it would be interesting to obtain efficient algorithms for the two problems defined in this chapter, and extend the concepts of this thesis to three dimensions.

Bibliography

[ACGOY88] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, C. Yap, *Parallel Computational Geometry*, Algorithmica, pp. 293–327, 1988.

[BT85] B.K. Bhattacharya, G.T. Toussaint, *On Geometric Algorithms that use the Furthest-Point Voronoi Diagram.*, Computational Geometry, ed. G.T. Toussaint, North-Holland, pp. 43–61, 1985.

[Ba75] B.G. Baumgart, *A Polyhedron Representation for Computer Vision*, 1975 National Computer Conference. AFIPS Conference Proceedings, Vol. 44. AFIPS Press, Arlington Va., pp. 589–596, 1975.

[BDDG82] J.E. Boyce, D P Dobkin, R.L. Drysdale III, L.J. Guibas, *Finding Extremal Polygons*, 14th STOC, pp. 282–289, 1982.

[Ch86] J.S. Chang, *Polygon Optimization Problems*, TR. No. 240, NYU, Courant Inst. Math. Sci., New York, N.Y., Aug. 1986.

[CY84] J.S. Chang, C.K. Yap, *A Polynomial Solution to Potato Peeling and Other Polygon Inclusion and Enclosure Problems*, 25th FOCS, pp. 408–416, May 1984

[Ch83] B. Chazelle, *The Polygon Containment Problem*, Advances in Computing Research, F P Preparata ed., V1, pp. 1–33, 1983.

[DH73] R.O. Duda, P E. Hart, *Pattern Classification and Scene Analysis*, John Wiley and Sons, pp. 356–362, 1973.

[Eu] Euclid, *The Elements. c.* 300 B.C.

[Fo85] S. Fortune, *A Fast Algorithm for Polygon Containment by Translation*, Automata, Languages and Programming, Springer Lecture Notes in Comp. Sci 194, pp. 189-198, 1985.

[Fo86] S. Fortune, *A Sweepline Algorithm for Voronoi Diagrams*, Proc. 2nd. Ann ACM Sympos. Comput. Geom, pp. 313–322, 1986.

[Gr72] R.L. Graham, *An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set*, Information Processing Letters, Vol. 1, pp. 132–133, 1972.

[GY83] R.L. Graham, F.F. Yao, *Finding the Convex Hull of a Simple Polygon*, J. Algorithms, Vol. 4, No. 4, pp. 324–331, 1983.

[GS85] L. Guibas, G. Stolfi, *Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi diagrams*, ACM Transactions on Graphics, Vol. 4, No. 2, pp. 74–123, April 1985.

[HW84] J. Hopcroft and G. Wilfong, *On the Motion of Objects in Contact*, TR 84–602, Dept. of Comp. Sci., Cornell Univ., May 1984.

[KL85] V. Klee, M.C. Laskowski, *Finding the Smallest Triangles Containing a Given Convex Polygon*, Journal of Algorithms, Vol. 6, pp. 359–375, 1985.

[Kn73] D.E. Knuth, *The Art of Computer Programming. Volume III: Sorting and Searching*, Addison-Wesley, Mass., 1973.

[Le82] D.T. Lee, *Medial Axis Transformation of a Planar Shape*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol PAMI-4, No. 4, pp. 363–369, July 1982.

[Le83] D.T. Lee, *On Finding the Convex Hull of a Simple Polygon*, Int'l J Computer and Information Sciences, Vol. 12, No. 2, pp. 87–98, 1983.

[LD81] D.T. Lee, R.L. Drysdale, *Generalization of Voronoi diagrams in the plane*, SIAM J. Computing, Vol. 10, No. 1, pp. 73–87, Feb. 1981.

[MY86] S. Madilla, C.K. Yap, *Moving a Polygon Around a Corner in a Corridor*. Proc. 2nd. Ann. ACM Sympos Comput. Geom., pp. 187–192, 1986.

[MA79] D. McCallum, D. Avis, *A Linear Time Algorithm for Finding the Convex Hull of a Simple Polygon*, Information Processing Letters, Vol. 9, pp. 201–206, 1979.

[MP78] D.E. Mueller, F.P. Preparata, *Finding the Intersections of Two Convex Polyhedra*, Theoretical Computer Science, Vol 7, No. 2, pp, 217–236, Oct. 1978.

[OR87] J. O'Rourke, *Art Gallery Theorems and Algorithms*, Oxford University Press, New York, 1987.

[OAMB84] J. O'Rourke, A. Aggarwal, S Maddila, M. Baldwin, *An Optimal Algorithm for Finding Minimal Enclosing Triangles*, TR JHU/EECS-84/08, Dept. of Elec. Eng. and Comp. Sci., The Johns Hopkins University, May 1984.

[PS85] F.P. Preparata, M.I. Shamos, *Computational Geometry, an Introduction*, Springer-Verlag, New York, 1985.

[Sh77] M.I. Shamos, *Computational Geometry*, Ph.D. Dissertation, Yale University, 1977.

[SH75] M.I. Shamos, D. Hoey, *Closest-point Problems*, Proc. 16th IEEE Symposium on Foundations of Computer Science, pp. 151–162, Oct. 1975.

[St82] G. Strang, *The Width of a Chair.* The American Math. Monthly, Vol. 89, no. 8, pp. 529–535, Oct. 1982.

[Te88] M. Teichmann, *Shoving a Table Into a Corner*, Snapshots of Computational and Discrete Geometry, ed. G.T. Toussaint, TR. SOCS 88.11, McGill University, Montréal, pp. 99–118, June 1988.

[Te89] M. Teichmann, *Shoving a Table Into a Corner*, First Canadian Conference on Computational Geometry, McGill University, Montréal, August 1989.

[To83a] G.T. Toussaint, *Solving Geometric Problems with Rotating Calipers.* Proc. IEEE MELECON '83, Athens, Greece, May 1983.

[To83b] G.T. Toussaint, *Computing Largest Empty Circles with Location Constraints* International Journal of Computer and Information Sciences, Vol 12, No. 5, pp. 347–358, Oct. 1983.

[Ya86a] C. K. Yap, *How to Move a Chair Through a Door,* TR. No. 238, NYU, Courant Inst. Math. Sci., New York, N Y., Aug. 1986.

[Ya86b] C. K. Yap, *Algorithmic motion planning,* Advances in Robotics, Volume 1: Algorithmic and Geometric Aspects, ed. J.T. Schwartz and C.K. Yap, Lawrence Erlbaum Assoc., N.J., 1986.