INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

**FACILITY LOCATION
CONSTRAINED TO A SIMPLE POLYGON**

by

QINGDA WANG

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

School of Computer Science
Carleton University
Ottawa, Ontario

February 27, 2001

©Copyright
2001, Qingda Wang

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-61034-9

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

FACILITY LOCATION
CONSTRAINED TO A SIMPLE POLYGON

submitted by Qingda Wang
in partial fulfillment of the requirements for
the degree of Master of Science
Information and Systems Science

Dr. Frank Denhe

(Director, School of Computer Science)

Dr. Prosenjit Bose

(Thesis Supervisor)

Carleton University

April 27, 2001

Abstract

In this thesis we first present a brief survey on the research on facility location problems, including reviews on some recent papers. Then we present two efficient algorithms for locating a single obnoxious or desirable facility in a simple polygonal region. Many realistic facility location problems require the facilities to be constrained to lie in a simple polygonal region. Given a set S of m demand points and a simple polygon R of n vertices, we show how to compute the location of an obnoxious facility or a desirable facility constrained to lie in R , both in $O((m+n)\log m + m\log n)$ time. The algorithms are exact, efficient, and straightforward to implement, improving best known algorithms in literature. The algorithms are also extended to the case where the constraining region consists of multiple simple polygons.

Acknowledgments

I would like to express my deep appreciation to my advisor, Prof. Prosenjit Bose. I have learned a lot from him, both on scientific knowledge and research skills, as well as scientific integrity as an individual and a researcher, during and beyond the course of the thesis. Thank him for proposing the wonderful problems tackled in this report and for his valuable comments and discussions regarding this work.

Table of Content

<i>Abstract</i>	<i>iii</i>
<i>Acknowledgments</i>	<i>iv</i>
<i>Table of Content</i>	<i>v</i>
<i>List of Figures</i>	<i>viii</i>
Chapter 1 Introduction	1
1.1 Model of Computation and Growth of Functions	1
1.2 A Brief Survey of Facility Location Problems	3
1.3 Some Frequently-Used Data Structures and Algorithms in Facility Location	5
1.3.1 Voronoi Diagrams	5
1.3.2 Point Location	8
1.3.3 Ray Shooting	11
1.4 Reviews of Research Papers on Facility Location	12
1.4.1 A Review of Toussaint's Paper: "Computing Largest Empty Circles with Location Constraints"	12

1.4.2	A Review of Katz, Kedem and Segal's Paper: "Improved Algorithms for Placing Undesirable Facilities"	15
1.4.3	A Review of Ben-Moshe, Katz and Segal's Paper: "Obnoxious Facility Location: Complete Service with Minimal Harm"	17
1.4.4	A Review of Bose and Toussaint's Paper: "Computing the constrained Euclidean, geodesic and link centers of a simple polygon with applications"	19
1.4.5	A Review of Hurtado, Sacristán, and Toussaint's Paper: "Constrained Facility Location"	24
1.5	Outline of This Report	30
 Chapter 2 Locating an Obnoxious Facility in a Simple Polygon		31
2.1	Related Work	31
2.2	Preliminaries	32
2.2.1	Improved Characterization	33
2.2.2	Characterization of the Location of the Obnoxious Facility	39
2.3	The Algorithm	42
2.4	Extension to Multiple Polygons	45
 Chapter 3 Euclidean Center Constrained to a Simple Polygon		48
3.1	Related Work	48

3.2	Preliminaries	50
3.2.1	Improving Part 3 of Lemma 3.1	52
3.2.2	Improving Part 4 of Lemma 3.1	54
3.2.3	Characterization of the Constrained Euclidean Center	63
3.3	The Algorithm	64
3.4	Extension to Multiple Polygons	68
Chapter 4	Conclusion	69
References		71

List of Figures

Figure 1. A Polygonal Subdivision. Arrows denote line segments going to infinity	8
Figure 2. Computing the median y_m and applying the oracle of Lemma 1.5 to $y = y_m$	27
Figure 3. Discarding a point for a horizontal bisector.....	27
Figure 4. Computing the median x_m and applying the oracle of Lemma 1.5 to $x = x_m$	28
Figure 5. Illustration for proof of Lemma 2.2	36
Figure 6. Constructing additional simple polygons for general ray shooting query	37
Figure 7. Illustration for proof of Lemma 3.2	53
Figure 8. Illustration for proof of Lemma 3.3.	56
Figure 9. Finding the farthest point on the convex hull of S to a given line l	59

Chapter 1 Introduction

In this chapter we first introduce the model of computation we use in this thesis. We then present a brief survey of facility location problems, and describe several techniques, algorithms and data structures from computational geometry that are frequently used in the research of facility location problems. Finally, as a background of the research presented in this thesis, we review several relevant papers on facility location.

1.1 Model of Computation and Growth of Functions

To analyze an algorithm, we must have a *model of computation* that is used, including a model for the resources and their costs [12]. The model of computation is a formal, abstract definition of a computer. Using a model one can more easily analyze the intrinsic execution time or memory space of an algorithm while ignoring many implementation issues. There are many models of computation which differ in computing power and the cost of various operations.

In this thesis, we assume a generic one-processor, *random-access machine* (*RAM*) model of computation. In the RAM model, instructions are executed sequentially, with no concurrent operations.

Most often we want to measure the *computation time* when analyzing an algorithm. In general, the time taken by an algorithm grows with the size of the input, so we generally describe the *running time* of an algorithm in terms of its *input size*. The most natural measure of the input size is the number of items in the input, and the running time of an algorithm on a particular input is the number of primitive operations or “steps” executed. The notion of step is defined in such a way that it is as machine-independent as possible.

When the input size is large enough, the *order of growth* of an algorithm characterizes the algorithm’s efficiency, and we are no longer concerned with the different costs of different types of primitive operations. Such a view allows us to study the *asymptotic* efficiency of an algorithm.

Several asymptotic notations exist to describe the asymptotic running time of an algorithm. The O -notation asymptotically bounds a function from above. Informally, saying some equation $f(n) = O(g(n))$ means it is less than some constant multiple of $g(n)$. More formally it means there are positive constants c and n_0 , such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$.

Just as O -notation provides an asymptotic upper bound on a function, the Ω -notation asymptotically bounds a function from below, and the Θ -notation asymptotically bounds a function from both above and below. Furthermore, the o -notation and the ω -notation denote an upper bound and lower bound that is not asymptotically tight, respectively [12].

1.2 A Brief Survey of Facility Location Problems

Facility location is a classical branch of problems of operations research that has also been examined by the computational geometry community. Most of facility location problems described in the literature deal with “desirable” facilities, such as hospitals, supermarkets, post-offices and warehouses [13, 25]. The customers try to have the facilities as close as possible to their own location. Thus the goal is to minimize a distance function between the facilities and the demand sites (i.e., customers).

However, just as important are the problems dealing with “undesirable” or obnoxious facilities, such as nuclear power plants, garbage dump sites, mega-airports, and chemical plants. In this case the customers try to stay away from the facilities. Research concerning obnoxious facility location began in the mid 1970’s, when the environmental impact was considered in the industrial

societies [6, 11, 17]. The goal there is to maximize instead of minimizing the distance function between the facility and demand points. Notice that for the problem to be meaningful and practical, some constraints on the location of the facility should be specified, e.g., forcing it to lie in some bounded region.

There are a number of elements to consider in the facility location problems [13]. The most important components are listed as follow:

- (i) The location of the facilities can be continuous, discrete or on a network.
- (ii) The location of the facilities may be restricted to a region.
- (iii) The number of facilities to be located can be one, or more.
- (iv) For desirable facility location, the decision-maker's objective can be *minimax* (minimize the maximum distance between each demand point and the facility) or *minisum* (minimize the sum of the distances between each demand point and the facility). Similarly, for obnoxious facility location, the decision-maker's objective can be *maximin* (maximize the minimum distance between each demand point and the facility) or *maxisum* (maximize the sum of the distances between each demand point and the facility). In obnoxious facility location problems *maximin* is more commonly used than *maxisum* for fairness to the customers.
- (v) The distance measure can be rectilinear (L_1 or L_∞), Euclidean (L_2), geodesic, or other metrics.

- (vi) The objective function distances can be weighted or unweighted.
- (vii) Various other constraints may be specified, e.g. distances between the facilities may be required to be greater than a given d , or each of a number of neighborhoods in the region must be served by at least one facility.

1.3 Some Frequently-Used Data Structures and Algorithms in Facility Location

In this section we describe some of the techniques, algorithms and data structures from computational geometry that are frequently used in the research of facility location problems. These are also the data structures and algorithms we have used in developing the algorithms presented in Chapter 2 and 3 of this thesis.

1.3.1 Voronoi Diagrams

Voronoi diagram in its most basic setting is defined as the following. Let $S = \{s_1, s_2, \dots, s_n\}$ be a set of n given point sites in the plane. The Voronoi diagram of S is the partition of the plane into n cells, one for each site in S , with the property that a point q lies in the cell corresponding to a cell s_i iff $d(q, s_i) < d(q, s_j)$ for each s_j in S with $j \neq i$, where d is the Euclidean distance [3]. We denote

the Voronoi diagram of S by $VD(S)$. The cell that corresponds to a site s_i is denoted $V(s_i)$.

The Voronoi diagram has the following properties [3]:

- i) The Voronoi diagram is a planar subdivision whose edges are straight line segments. Some edges are line segments and others are half-lines. Unless all sites are collinear there will be no edges that are full lines.
- ii) The Voronoi diagram of a set of n points in the plane has at most $2n-5$ vertices and $3n-6$ edges;
- iii) For the Voronoi diagram $VD(S)$ of a set of points S the following holds:
 - A point q is a vertex of $VD(S)$ iff its largest empty circle $C(q)$ contains three or more points on its boundary.
 - The bisector between sites s_i and s_j defines an edge of $VD(S)$ iff there is a point q in R^2 such that $C(q)$ contains both s_i and s_j on its boundary but no other site.

The Voronoi diagram can be computed in $O(n \log n)$ time with $O(n)$ storage using a sweep line algorithm [15].

Voronoi diagram can be generalized in many ways. One generalization is to partition the plane into regions according to the k closest sites, for some $1 \leq k \leq n - 1$. The diagrams obtained in this way are called *higher-order Voronoi diagram*. The order-1 Voronoi diagram is the standard Voronoi diagram. The order- $(n - 1)$ Voronoi diagram is also called the *farthest-point Voronoi diagram*, because the Voronoi cell of a point s_i is now the region of points for which s_i is the farthest site.

Formally, given a point $x \in R^2$, we let $\phi(x)$ denote the farthest neighbors of x in S , that is, the set of points in S such that $d(x, \phi(x)) = \max_{y \in S} d(x, y)$. The farthest-point Voronoi diagram of S , denoted by $FPVD(S)$, partitions the plane into n cells, $V(s_i)$, such that for any point $q \in V(s_i)$, $s_i \in \phi(q)$.

The farthest-point Voronoi diagram is determined only by points on the convex hull, therefore all cells in the farthest-point Voronoi diagram are unbounded. The diagram can also be computed in $O(n \log n)$ time, based on a divide-and-conquer scheme [27].

1.3.2 Point Location

Point location is one of the fundamental problems in computational geometry. Given a polygonal subdivision, where the Euclidian plane is subdivided into polygons by n line segments that intersect only at their endpoints (Figure 1), we are asked to determine which of the polygons (regions) contains a given query point. Solutions vary in preprocessing time P , space used S , and query time Q . Solutions achieve optimality when $S = O(n)$, $Q = O(\log(n))$ and $P = O(n)$.

Figure 1. A Polygonal Subdivision. Arrows denote line segments going to infinity.

To be able to search such a polygonal subdivision efficiently, one may attempt to decompose each region into polygons for which the search operation is relatively simple. The success of point location depends on the ability to narrow down quickly the set of components to be searched. A potentially

successful point-location method should exhibit the following features: the planar subdivision is efficiently transformed into a new one, each of whose constituent polygons intersects a small and fixed number (possibly just one) of original regions, and can be organized so that binary search is applicable.

The Lipton-Tarjan method [23] is based on the planar separator theorem, and is the earliest optimal solution for this search problem. But it so far has been only of theoretical interest because its implied constant is so large as to make implementation impractical.

Other optimal approaches include Kirkpatrick's hierarchy [22], Edelsbrunner, Guibas, and Stolfi's chain method [14], and Sarnak and Tarjan's persistent search tree [28]. All methods create new geometric objects (components) that permit efficient searching. So, Kirkpatrick's method [22], by retriangulating portions of the subdivision, creates a suitable small set of new *triangles* that are easily searched. Edelsbrunner, Guibas, and Stolfi's method [14] is based on extracting from the planar subdivision a set of polygonal chains that partition the plane into *monotone polygons*, which can be easily searched. Finally, Sarnak and Tarjan's method [28] is based on partitioning the subdivision into *slabs* which permit binary searching. All the methods also build a structure of directed acyclic graph that allows indexing into the target component quickly.

All of the three methods are optimal with respect to query time ($O(\log n)$) and space used ($O(n)$). Sarnak and Tarjan's persistent search tree method [28] works on general planar subdivision and needs $O(n \log n)$ preprocessing time. Kirkpatrick's hierarchical tree method [22] works on triangulated subdivisions. To apply it to general planar subdivisions, we can use a linear-time algorithm for triangulating simple polygons [8]. Thus Kirkpatrick's method needs only $O(n)$ preprocessing time. Edelsbrunner, Guibas, and Stolfi's chain method [14] works on monotone subdivision. An $O(n \log n)$ time line sweeping algorithm can be used to partition a general planar subdivision into monotone pieces [14], then the chain method needs $O(n \log n)$ preprocessing time. However, Chazelle's linear-time algorithm for triangulating simple polygons can also be applied here thus the preprocessing time of the chain method can be reduced to $O(n)$ time.

Based on the slab method, Mulmuley [26] devised a randomized incremental approach to trapezoidal decomposition, and achieved optimal bounds on storage ($O(n)$) and query time ($O(\log n)$). The preprocessing takes $O(n \log n)$ time. The algorithm is the simplest compared to all the above methods, but the optimality is achieved only in the expected case.

1.3.3 Ray Shooting

The ray shooting problem in a simple polygon P asks for a data structure representing P that can efficiently answer queries of the form “find the first intersection between a ray and the boundary of P .” A ray is a directed half-line, specified by a point of origin p and a direction vector.

Optimal logarithmic query time for the ray shooting problem for a simple polygon was first achieved by Chazelle and Guibas [10]. Another optimal algorithm, which is simpler, was proposed by Chazelle et al. [9]. Their algorithm is based on a balanced hierarchical decomposition of the input polygon into geodesic triangles (a geodesic triangle is a region of a simple polygon bounded by shortest paths between three points.), where a ray intersects $O(\log n)$ geodesic triangles. However, the algorithm is still quite complicated.

Hershberger and Suri [20] developed the geodesic triangulation approach in [9] further to obtain a Steiner triangulation that decomposes the polygon into ordinary triangles that still has the property that a ray intersects only $O(\log n)$ triangles. The query algorithm, after locating the triangle containing the origin of the ray, walks along the ray, advancing from one triangle to a neighboring one until the polygon boundary is reached. The Steiner triangulation is computed in linear time. Therefore the algorithm is both optimal and simple.

Based on the technique of using a spanning tree of small stabbing number [9], the triangulation procedure can be extended to the case of k disjoint polygons with a total of n vertices, so that a ray intersects at most $O(\sqrt{k} \log n)$ triangles before hitting the boundary of one of the polygons. The preprocessing time for building such a triangulation is $O(n\sqrt{k} + n \log n + k^{3/2} \log k)$.

1.4 Reviews of Research Papers on Facility Location

In this section we review some of the research papers on facility location problems. In Section 1.4.1 through 1.4.3, we review three papers on obnoxious facility location. In Section 1.4.4 and 1.4.5, we review two papers on locating a desirable facility constrained to a simple polygon and a convex polygon, respectively.

1.4.1 A Review of Toussaint's Paper: "Computing Largest Empty Circles with Location Constraints"

The problem of locating a single obnoxious facility is also known as the largest empty circle (LEC) problem. Let Q be a set of n points on the plane. The LEC problem consists in finding the largest circle C with center in the convex hull of Q (denoted by $CH(Q)$) such that no point $q_i \in Q$ lies inside C . The problem has a lower bound of $\Omega(n \log n)$ by reducing to the MAX-GAP problem.

In this paper, Toussaint [31] showed that an $\Theta(n \log n)$ algorithm can be obtained if the center of C is constrained to lie in an arbitrary convex n -gon. And an $O(n \log n + k \log n)$ algorithm is given for solving this problem when the center is constrained to lie in an arbitrary simple n -gon P , where k denotes the number of intersections occurring between edges of P and edges of the Voronoi diagram of Q .

Toussaint first pointed out an error in Shamos's $O(n \log n)$ algorithm [29] for finding the center of the LEC constrained to lie in $CH(Q)$. Then he proposed a new algorithm to solve a more general problem, i.e. finding the center of the LEC constrained to lie in an arbitrary simple n -gon P . The algorithm is based on the following observation:

Lemma 1.1 *Given a set of points $Q = \{q_1, q_2, \dots, q_n\}$ and a simple polygon $P = \{p_1, p_2, \dots, p_n\}$, the largest circle C with center O such that $O \in P$ and no point in Q lies in the interior of C must be a circle centered at either (a) a vertex of the Voronoi diagram of Q (denoted by $VD(Q)$), or (b) an intersection point of $VD(Q)$ and the boundary of P , or (c) a vertex of P .*

Based on Lemma 1.1, the algorithm is straightforward:

Algorithm LEC-2

Input: A set of points $Q = \{q_1, q_2, \dots, q_n\}$ and a simple polygon $P = \{p_1, p_2, \dots, p_n\}$.

Output: The largest empty circle with center in P .

1. Compute $VD(Q)$.
2. Preprocess P for point inclusion test.
3. Determine which Voronoi vertices of $VD(Q)$ lie in P and compute their empty circles.
4. Compute the intersections of $VD(Q)$ and the boundary of P , and for each intersection its empty circle.
5. Preprocess $VD(Q)$ for point location test.
6. For each vertex of P determine the nearest neighbour $q_i \in Q$ and compute its empty circle.
7. Select and output the largest circle found in step 3, 4 and 6.

$VD(Q)$ can be computed in $O(n \log n)$ time [29]. Step 2 and 5 can be done in $O(n \log n)$ time using Kirkpatrick's algorithm such that each point location test in step 3 and 6 takes $O(\log n)$ time [22]. The best known algorithm at the time of writing for computing the intersection points in step 4 takes $O(n \log n + k \log n)$ time, where k is the number of intersections [2]. Thus the total complexity of the algorithm is dominated by step 4 and is $O(n \log n + k \log n)$. In the worst case this is $O(n^2 \log n)$ time, since k can be $\Omega(n^2)$.

It should be pointed out that a new trapezoidal sweep algorithm for reporting red/blue segment intersections by Chan [7] can be used in step 4. Then the complexity of the algorithm can be improved to $O(n \log n + k)$, or $O(n^2)$ in the worst case.

When the polygon P is convex, the number of intersections between $VD(Q)$ and the boundary of P is $O(n)$, then the algorithm runs in $\Theta(n \log n)$ time.

1.4.2 A Review of Katz, Kedem and Segal's Paper: "Improved Algorithms for Placing Undesirable Facilities"

Katz, Kedem and Segal [Katz, 1999] proposed improved algorithms for the following *multiple* obnoxious facility location problem under L_∞ norm: Given a set P of n points in a rectangular region R , and given a distance r and another distance d , locate k undesirable facilities such that the smallest distance between each demand point and the facilities is at least r , and the distance between any pair of facilities is at least d .

First the problem is reformulated. Let U be the union of squares of size r around each point $p_i \in P$, and let $V = R - U$. The decision problem: "does there exist a set of k locations in V with pairwise distance at least d ?", is then solved

in $O(n \log n)$ time for $k = 2$ or 3 . The case where $k = 2$ is straightforward. In the case where $k = 3$, first it is proved that at least one of the three sites is a vertex of the boundary of V . Now, for each vertex $v \in V$, assuming one facility is located on v , the authors solve the two facility problem constrained to the region $V - Q_v$, where Q_v is the square with side size $2d$ centered at v , in $O(\log n)$ time. The technique includes preprocessing the vertices of V for orthogonal range searching so that all the candidate locations of the two facilities can be searched in total $O(n \log n)$ time.

In the case where $k \geq 4$, a straightforward recursive algorithm is proposed since at least one of the sites is a vertex of V . Thus the algorithm runs in time $O(n^{k-2} \log n)$ for $k \geq 4$, a significant improvement over Brimberg and Mehrez's $O(n^{2k})$ algorithm [5].

The optimization scheme employs the technique of sorted matrices by Frederickson and Johnson [16] and adds a multiplicative $O(\log n)$ factor to the running time of the decision algorithm.

1.4.3 A Review of Ben-Moshe, Katz and Segal's Paper: "Obnoxious Facility Location: Complete Service with Minimal Harm"

Ben-Moshe, Katz and Segal presented efficient algorithms for several instances of the following obnoxious facility location problem [1]: Let P be a set of n demand points, and let R be a set of m , $m < n$, rectangular regions (called *neighborhoods*). Find k sites $c_1, \dots, c_k$ for the k obnoxious facilities, such that (i) $C = \{c_1, \dots, c_k\}$ is a *piercing set* for R , that is, each of the neighborhoods in R is served by at least one facility that is located in the neighborhood. (ii) The minimal distance between a demand point in P and a site in C is maximized.

First the following subproblem, which is called the reception problem, is solved efficiently: Given a set D of n unit disks, determine whether a constant-size query polygonal region Q is fully covered by the disks in D . Making use of Voronoi diagram, simplex range searching, and ray shooting, the set D can be preprocessed in $O(n^{1+\epsilon})$ time so that the query can be answered in $O(n^{1/2+\epsilon})$ time. In the special case where the query regions are axis-parallel rectangles, the preprocessing time can be reduced to $O(n \log n)$ and query time to $O(\log n)$.

The authors then proceed to solve several instances of the obnoxious facility location problems stated above. In the case where $k = 1$ under the L_∞

metric, the problem is solved in $\Theta(n \log n)$ time using the L_∞ Voronoi diagram of P restricted to the non-empty rectangle $R = \cap \mathcal{R}$.

For the case where $k = 2$ under the L_∞ metric, the decision problem is: “does there exist a piercing pair $\{p_1, p_2\}$ for $\mathcal{R}$ (each region in $\mathcal{R}$ is served by at least one facility in $\{p_1, p_2\}$), such that the distance between any point of P and either p_1 or p_2 is at least d ? (d is the parameter of the problem.)”. Let U be the union of the squares of edge length $2d$ centered at the points of P . An algorithm is suggested which in $O(n \log n)$ time determines whether there is a partition of $\mathcal{R}$ into $\mathcal{R}_1$ and $\mathcal{R}_2$ that allows a piercing pair $\{p_1, p_2\}$, $p_1 \in \mathcal{R}_1$, $p_2 \in \mathcal{R}_2$ such that neither p_1 nor p_2 lies in U , by employing a technique from Sharir [30] that resembles searching in monotone matrices. The optimization scheme uses the technique of Frederickson and Johnson [16] and adds a multiplicative $O(\log n)$ factor to the running time.

Similar algorithms are proposed under Euclidean distance in the cases where $k = 1$ or 2 .

1.4.4 A Review of Bose and Toussaint's Paper: "Computing the constrained Euclidean, geodesic and link centers of a simple polygon with applications"

The paper [4] is motivated by the following problem in the manufacturing industry: finding a suitable location for the pin gate, which is the point from which liquid is poured or injected into a mold. Experience has shown that a suitable pin gate location possesses several desirable geometric characteristics, namely, (i) the distance from the pin gate to any point in the mold should be small, and (ii) the number of turns on the path from a point in the mold to the pin gate should be small. In the paper, Bose and Toussaint solved the pin gate location problem, when viewed from a pure geometric perspective, for molds modeled as simple polygons.

The pin gate location with property (i) mentioned above can be interpreted as the *Euclidean center* constrained to lie inside or on the boundary of a simple polygon (the latter case arises if the pin gate is required to lie on the boundary of the mold), when the distance is measured in the Euclidean metric. The distance can also be measured by the geodesic metric, i.e. the minimum distance the liquid must travel inside the mold to reach a destination. In this case, the pin gate location problem can be interpreted as computing the *geodesic center* constrained to lie inside or on the boundary of a simple polygon. Finally, the pin gate location with property (ii) can be interpreted as the *link center* constrained

to lie inside or on the boundary of a simple polygon, where distance is measured in link metric. The link metric measures the number of turns in a path between two points.

The Euclidean center problem has a long history which is summarized in [27]. Megiddo found an optimal $O(n)$ algorithm to solve the unconstrained version of the problem, based on a prune-and-search strategy [24].

Given a simple polygon M with n vertices, Bose and Toussaint solved the Euclidean center problem with the center constrained to lie in the interior of M , or on the boundary of M , in $O(n \log n + k)$ time, where k is the number of intersections between M and the furthest point Voronoi diagram of the vertices of M . Furthermore, they showed how to compute the geodesic center of M constrained to the boundary in $O(n \log n)$ time and the geodesic center of M constrained to lie in a polygonal region in $O(n(n + k))$ time where k is the number of intersections of the geodesic furthest point Voronoi diagram of M with the polygon. Finally, they showed how to compute the link center of M constrained to the boundary of M in $O(n \log n)$ time [4].

Here we review only their algorithms on computing the Euclidean center constrained to lie inside or on the boundary of a simple polygon. They solved a slightly more general problem than the one mentioned above. Given a set $S =$

$\{s_1, s_2, \dots, s_m\}$ of m points in the plane, and an n vertex simple polygon P , we wish to find the point c in P whose maximum distance to any point in S is minimized. c is referred to as the Euclidean center of S constrained to P and denote by $EC_P(S)$.

The algorithm makes use of the farthest point Voronoi diagram of the set S , denoted as $FPVD(S)$. Given a point $x \in R^2$, we let $\phi(x)$ denote the farthest neighbors of x in S , that is, the set of points in S such that $d(x, \phi(x)) = \max_{y \in S} d(x, y)$ where d is the Euclidean distance. The $FPVD(S)$ partitions the plane into unbounded convex cells, $V(s_i)$, such that for any point $p \in V(s_i)$, $s_i \in \phi(p)$.

The authors first proved the following lemma:

Lemma 1.4 *If $EC_P(S)$ does not lie on the midpoint of the diameter of S , then it lies on one of the following points that yields the smallest enclosing circle:*

1. *A vertex of $FPVD(S)$ contained in P ;*
2. *A vertex of the polygon P ;*
3. *A proper intersection point of $FPVD(S)$ and the boundary of P ;*
4. *A point x on an edge e of P with the property that $\forall y \in e$, if $\phi(y) = \phi(x)$ then $d(y, \phi(x)) \geq d(x, \phi(x))$.*

Based on Lemma 1.4, the following algorithm is outlined to compute $EC_P(S)$:

Algorithm *Euclidean center of S constrained to lie in P*

Input: A set of demand points $S = \{s_1, s_2, \dots, s_m\}$, and a simple polygon P of n vertices.

Output: Euclidean center of S constrained to lie in P .

1. Compute $FPVD(S)$.
2. Compute $DIAM(S)$.
3. Compute the circle C having $DIAM(S)$ as diameter.
4. Preprocess P for point inclusion test [22].
5. If the midpoint of $DIAM(S)$ is contained in P and all points of S are contained in C then exit with the midpoint of $DIAM(S)$.
6. Compute the set of vertices of $FPVD(S)$ contained in P . Let V_c represent this set.
7. Compute the set of intersections $I_c = \{i_1, i_2, \dots, i_k\}$ of P with $FPVD(S)$.
8. Partition each edge e_i of P such that for every pair of points $x, y \in e_i$, we have that $\phi(x) = \phi(y)$. Denote the j^{th} partition of e_i by e_{ij} .
9. For each e_{ij} , compute the point on e_{ij} closest to $\phi(e_{ij})$. If this point is not an endpoint of e_{ij} , place it in the set E_c .
10. Let P_c represent the vertices of P . For each point in c in V_c, I_c, P_c , and E_c , compute the smallest spanning circle with center c . Let SP represent this set.

11. Select the smallest circles in SP , and output their centers and the radius.

It has been shown that the steps 7-9 dominate the running time of the algorithm. Step 7 can be done in $O((m + n) \log(m + n) + k)$ time, where k is the number of intersections between P and $FPVD(S)$, using the algorithm of Chan [7]. Once the intersection points have been computed, step 8 and 9 can be achieved in $O(n + k)$ time. Thus the total running time of the algorithm is $O((m + n) \log(m + n) + k)$.

For simple polygons, k can be $\Omega(mn)$. For convex polygons, however, k is $O(m)$, thus the algorithm runs in $O((m + n) \log(m + n))$ time when P is convex. In the next section (Section 1.4.5) we are going to review a paper ([21]) where it is shown how to remove the $\log(m + n)$ factor of the running time for the case where P is convex.

To compute the Euclidean center constrained to lie on the boundary of P , a lemma similar to Lemma 1.4 is derived. The modifications to the algorithm are straightforward and the running time remains the same.

1.4.5 A Review of Hurtado, Sacristán, and Toussaint's Paper: "Constrained Facility Location"

In this paper [21], an $\Theta(n+m)$ time algorithm is proposed for the Euclidean center problem with the center constrained to satisfy m linear constraints. As a corollary, the problem with the center constrained to lie in an m -vertex convex polygon is solved in optimal linear time.

The algorithm employs a prune-and-search approach and makes use of the technique of Megiddo [24] for solving the Euclidean center problem constrained to a straight line. The basic idea of the algorithm is the following:

1. Solve the problem in one less dimension (with the constraint of having the center lying on a given line s), determine whether or not the solution to the original problem lies on the line or which side of the line it belongs to.
2. Apply this result recursively to reduce the size of the original problem.

For step 1, the following algorithm is proposed:

- (1) Detect if s and the constraint polygon intersect. If they do not intersect, determine on which side of s the polygon lies. Otherwise, find the line segment s' which is the portion of s that is inside the polygon, and proceed to the next step. This is shown to be done in linear time.

- (2) Solve the s -constrained problem, then solve the s' -constrained problem in a straightforward way.
- (3) Find the point(s) p_i that determines the maximum distance to the s' -optimum. Depending on the location of p_i relative to s , it is decided on which side of s the solution to the original problem lies.

In step (2), the center constrained to lie on s can be found in $O(n)$ time using Megiddo's algorithm. The rest of the algorithm can be done in $O(n+m)$ time. Therefore,

Lemma 1.5 *Given a line s , by solving the problem restricted to s it is possible to find the center of the minimum enclosing circle of a set of n points in the plane, constrained to belong to a set of m halfplanes, if it lies on s , or to decide on which side of s it lies, in optimal $\theta(n+m)$ time.*

To solve the original problem of finding the Euclidean center of a set of n points $\{p_1, \dots, p_n\}$ subject to a set of m linear constraints, from Lemma 1.5 we know we can use as an oracle a linear time algorithm that can decide on which side of a given line the solution to the problem lies. Thus the following algorithm is proposed.

1. Pair up the n points. Consider the orthogonal bisectors of the segments they form. Perform a coordinate change such that the bisectors with positive slopes and those with negative slopes are evenly divided. Pair each negative slope bisector with one of positive slope. Each pair of bisectors intersects in a point c_i . The rest of the bisectors are horizontal and vertical ones.
2. Pair up the m halfplanes determined by the linear constraints. Always pair up an upper halfplane with another non parallel upper halfplane, and a lower one with a lower one similarly. Compute the intersection point d_i of all the pairs of linear restrictions.
3. Compute the median y_m of all the y -coordinates of the horizontal bisectors, intersection points of the pairs of bisectors and intersection points of the pairs of linear constraints. Apply the oracle of Lemma 1.5 to the horizontal line $y = y_m$, determining whether the center lies on, above or below the line. If it is on the line, the oracle will find it and we are done. Otherwise, suppose it is determined that the solution lies below the line (Figure 2).

Figure 2. Computing the median y_m and applying the oracle of Lemma 1.5 to $y = y_m$

4. We can now discard a point p_i for each horizontal bisector belonging to the halfplane opposite to the solution (Figure 3).

Figure 3. Discarding a point for a horizontal bisector

5. Restrict our attention to the halfplane opposite to the solution (the upper one). Compute the median x_m of the x -coordinates of the vertical bisectors, intersection points of the pairs of bisectors and intersection points of the pairs of linear constraints that lie on the upper halfplane, and apply the oracle to the vertical line $x = x_m$. This determines if the solution lies on the

line, in which case we are done. Otherwise, the oracle determines on which of the four quadrants the solution lies. Suppose it lies in the lower right quadrant, denoted by Q (Figure 4).

**Figure 4. Computing the median x_m and applying the oracle
of Lemma 1.5 to $x = x_m$**

- Then we can discard a point p_i for each vertical bisector and each intersection point of bisectors that belongs to the quadrant Q' opposite to the solution. Moreover, we can discard at least one linear restriction l_i for each intersection point of linear constraints that belongs to Q' , using the arguments: (1) if l_i cannot intersect Q , it is irrelevant; (2) if the intersection of the halfplane determined by l_i with Q contains the intersection of a

halfplane determined by another linear constraint l_j with Q , then l_i can be discarded.

Thus, in this iteration of the algorithm we discard one point for each horizontal bisector lying in the upper halfplane, another one for each vertical bisector lying in the left halfplane, and another one for each intersection point c_i lying in Q' . As for the linear restrictions, the algorithm discards at least one for each intersection point d_i lying in Q' . In this process we discard at least $1/16$ of the input elements, either points or constraints, before applying the algorithm again to the reduced input.

Since the oracle runs in linear time, the cost of each application is linear in the size of the input because it only requires computing the median. We have $T(n+m) = O(n+m) + T(15/16(n+m))$. The linear running time follows.

The authors pointed out the essence of the algorithm is to find an oracle that decides in linear time on which side of a given line the solution lies. They showed the generality of the strategy by providing two more examples.

The authors' main contribution is that they go one step further from Megiddo's algorithm, applying the prune-and-search strategy to the linear constraints as well as to the points. The two are integrated and a linear time

algorithm is thus obtained. A good point to note is that the authors treat the points and linear constraints uniformly in a sense. In one iteration of the algorithm, they prune a fraction (at least $1/16$) of the total input, regardless of whether the input is a point or a linear constraint. It is possible that all the elements pruned are just points, or are just linear constraints.

1.5 Outline of This Report

Most facility location problems dealt with in the literature assumed convex feasible regions (including rectangular regions). However, this assumption is usually not very realistic since the feasible regions are often not convex due to geographical reasons and zoning laws. Therefore in this paper we study facility location problems under the assumption of simple polygonal feasible region instead.

In Chapter 2 and 3 we present improved algorithms for locating an *obnoxious* facility and a *desirable* facility in a simple polygonal region, respectively, under Euclidean distance. Both our algorithms run in subquadratic time. In Chapter 4 we summarize and conclude.

Chapter 2 Locating an Obnoxious Facility in a Simple Polygon

2.1 Related Work

The problem of locating a single obnoxious facility for a set of demand points is also known as the largest empty circle problem. As far as we know, Toussaint is the only one that has studied the obnoxious facility location problem constrained to a simple polygonal region ([31], also see Section 1.4.1). A *simple polygonal region* is a region enclosed by a single closed polygonal chain that does not intersect itself.

Toussaint's algorithm makes use of the Voronoi diagram of the demand points. Given a set of n points in S , we define the *Voronoi diagram* of S , denoted by $VD(S)$, as the subdivision of the plane into n cells, one for each point in S , with the property that a point q lies in the cell corresponding to a point p_i if and only if $d(q, p_i) = \min_{y \in S} d(q, y)$ where d is the Euclidean distance.

Given n demand points, Toussaint proposed an $O(n \log n)$ algorithm if the facility is constrained to lie in a convex n -gon, and an $O(n \log n + k \log n)$

algorithm if the facility is constrained to lie in a simple n -gon P , where k denotes the number of intersections occurring between edges of P and edges of the Voronoi diagram of the demand points. Notice k could be $\Omega(n^2)$. So in the worst case the algorithm runs in $O(n^2 \log n)$ time.

In this section we improve Toussaint's algorithm. Our algorithm runs in $O(n \log n)$ time if the facility is constrained to lie in a simple polygonal region, eliminating the dependence of the running time on k (the number of intersections between the polygon P and the Voronoi diagram of the demand points).

2.2 Preliminaries

The problem is stated as the following:

Problem 1: *Given a set $S = \{s_1, s_2, \dots, s_m\}$ of m demand points and an n vertex simple polygon region R , locate a single obnoxious facility within R such that the smallest Euclidean distance between each demand point and the facility is maximized.*

It is assumed that the demand points are in general position, i.e. no three points are collinear and no four points are co-circular.

In [31] Toussaint proved the following Lemma (restatement of Lemma 1.1 in Section 1.4.1):

Lemma 2.1 *The location of the single obnoxious facility under the constraints above lies on one of the following points that yields the largest empty circle:*

1. *A vertex of the Voronoi diagram of S (denoted as $VD(S)$) contained in R .*
2. *A vertex of the polygon R .*
3. *A proper intersection point of $VD(S)$ and the boundary of R .*

The quadratic runtime occurs in computing the third part of the above. We are going to improve on the characterization of Lemma 2.1, by removing the dependency on the number of intersections between the polygon R and the Voronoi diagram of the demand points. This will allow us to improve Toussaint's algorithm,

2.2.1 Improved Characterization

For convenience, i.e. to avoid problems caused by unbounded edges of $VD(S)$, let B be a large rectangle that contains S , R and all vertices of $VD(S)$. Denote by $BVD(S)$ the graph induced by $B \cap VD(S)$. First, $VD(S)$ can be computed in $O(m \log m)$ time and stored in a data structure such as a doubly-

connected edge list that allows us to walk along the edges of a given cell, or find in constant time the neighboring cell sharing an edge of a given cell [27]. Secondly, B can be computed in $O(m+n)$ time by finding the maximum and minimum of the x -coordinates and y -coordinates of the vertices of S , R and $VD(S)$, noticing the number of vertices in $VD(S)$ is $O(m)$. Finally, given B and $VD(S)$, we note that $BVD(S)$ can be computed in linear time, as explained in the following.

Let BI be the set of intersection points between B and $VD(S)$, i.e. intersection points between B and the unbounded edges of $VD(S)$ since B is large enough. To compute BI in $O(m)$ time, we need only show that we can find all the unbounded edges of $VD(S)$ in linear time. Notice that if a Voronoi cell $V(s)$ has unbounded edges, then s is a vertex of the convex hull of S (denoted by $CH(S)$), and $V(s)$ has exactly two unbounded edges. Now, starting from the leftmost demand point s_l , which must be a vertex of $CH(S)$, we find unbounded edges e_1 and e_2 of $V(s_l)$ by going through all edges of $V(s_l)$. We then proceed to the neighboring cell $V(s_i)$ sharing e_2 with $V(s_l)$ and find the other unbounded edge e_3 of $V(s_i)$. We then proceed to the neighboring cell of $V(s_i)$ which shares e_3 with $V(s_i)$, and so on. Ultimately we will come back and find e_1 and we stop. Since we go through each edge at most once to check whether it is unbounded, we can find all unbounded edges of $VD(S)$ in linear time. It takes constant time to compute the intersection point of each unbounded edge with B . Thus BI can

be computed in $O(m)$ time and has $O(m)$ points, therefore $BVD(S)$ can be computed in $O(m)$ time.

Lemma 2.2: *If the location of the single obnoxious facility under the constraints stipulated in Problem 1, lies on an intersection point c between an edge $e = [pq]$ of $BVD(S)$ and the boundary of R , then either $[cp]$ or $[cq]$ does not contain any intersection point between $BVD(S)$ and the boundary of R , i.e. among all intersection points between e and the boundary of R , c is the closest to one of the two endpoints of e .*

Proof: See Figure 5. For the sake of contradiction, suppose both $[cp]$ and $[cq]$ contains at least one intersection point between $BVD(S)$ and the boundary of R , denoted by x and y , respectively. Let $V(b)$ be one of the two Voronoi cells incident on e . We have either $\angle bcp$ or $\angle bcq$ is $\geq \pi/2$. Without loss of generality suppose $\angle bcp \geq \pi/2$. Now the circle centered at x with radius $d(x, b)$ is empty. However, $d(x, b) > d(c, b)$. Hence, we have a contradiction. $\square$

Figure 5. Illustration for proof of Lemma 2.2

Lemma 2.2 indicates that we need not consider all the proper intersection points between $VD(S)$ and R . Rather, for all intersection points occurring between a given edge $e = [pq]$ of $BVD(S)$ and the boundary of R , we need consider at most two points, i.e. the points closest to p and q .

The intersection points with this property can be found efficiently by ray shooting queries over R [20]. Before we proceed, notice that the solution in [20] for ray shooting problems over a simple polygon R of n vertices works regardless of whether the ray-origin lies inside or outside R , as explained in the following. Here we refer to the first intersection point of a ray and a polygon as a *hit point*.

In general, to deal with the cases where the ray-origin lies outside of R , we construct two additional simple polygons as follows. Let B' be a box large enough to contain all vertices of R and all possible points to be origins of query rays. Let y_{max} and y_{min} be the largest and smallest y -coordinates of the vertices of R . Let l be a horizontal line $y = y_0$ where $y_{max} > y_0 > y_{min}$. Let E and H be the intersection points of l with the boundary of B' , and let F and G be the intersection points of l with the boundary of R that are the closest to E and H , respectively. By adding the two line segments $a = [EF]$ and $b = [GH]$, we obtain two additional simple polygons R_1 and R_2 of $O(n)$ vertices as shown in Figure 6. This can be done in $O(n)$ time.

**Figure 6. Constructing additional simple polygons for
general ray shooting query**

The following steps are then carried out to perform the ray shooting query:

- (i) Preprocess R , R_1 and R_2 for point inclusion test using the algorithms of Kirkpatrick [22] or Edelsbrunner, Guibas and Stolfi [14].
- (ii) Preprocess R , R_1 and R_2 in $O(n)$ time for ray shooting queries using the algorithm of Hershberger and Suri [20].
- (iii) Given a point x and a ray r emanating from x ,
 - Determine in $O(\log n)$ time whether x is in R , R_1 or R_2 [14, or 22]. If x is on a or b and r is co-linear with a (and b), it is easy to determine whether the hit point exists and find it if so. Otherwise, denote by Q the polygon (R , R_1 or R_2) in which x lies (if x is on a or b , let Q be the polygon r points to). Perform the ray shooting query over Q with the ray r emanating from x , using the algorithm of [20]. Let p be the hit point.
 - If $Q = R$, return p and we are done. Otherwise test if p is on a , b or the boundary of B' . If not then return p and we are done again. Otherwise, if p is on the boundary of B' , then the hit point does not exist; and
 - If p is on a or b , perform another ray shooting query over the neighboring polygon of Q with the ray emanating from p and in the same direction as r . Let p' be the new hit point.
 - Test if p' is on the boundary of B' (p' cannot be on a or b this time). If so the hit point does not exist. Otherwise return p' .

In step (i), we can use Kirkpatrick's hierarchy method [22] or Edelsbrunner, Guibas and Stolfi's chain method [14], combined with Chazelle's linear time algorithm for triangulating simple polygons [8], thus preprocessing of R , R_1 and R_2 for point inclusion test can be done in $O(n)$ time. Taken together, regardless of whether the ray-origin is inside or outside R , we have a general algorithm to perform a ray shooting query over a simple n -gon R , with $O(\log n)$ query time, by preprocessing R in $O(n)$ time.

Now, to find the intersection points with the property mentioned in Lemma 2.2, for each vertex v of $BVD(S)$ and each edge e incident on v , we shoot a ray emanating from v and containing e . Let p be the hit point of the boundary of R by the ray. We then test if p is on e . If p is not on e , then e does not intersect R and p is discarded. Otherwise, p is the intersection point of e and R that is the closest to v , a candidate with the property mentioned in lemma 2.2.

2.2.2 Characterization of the Location of the Obnoxious Facility

Using the technique above allows us to avoid the expensive computation of all intersection points between $VD(S)$ and R . To summarize, the following lemma characterizes the desired location of the obnoxious facility.

Theorem 2.3: *The location of the single obnoxious facility under the constraints stipulated in Problem 1 lies on one of the following points that yields the largest empty circle:*

1. *A vertex of $VD(S)$ contained in R ;*
2. *A vertex of the polygon R ;*
3. *A point of the boundary of R hit by a ray emanating from a vertex v of $BVD(S)$ and containing an edge of $BVD(S)$ incident on v .*

Proof: A point in R can only be one of the following:

- (i) It is in the interior of R and in the interior of a cell of $VD(S)$;
- (ii) It is in the interior of an edge of R and in the interior of a cell of $VD(S)$;
- (iii) It is in the interior of an edge of $VD(S)$, and in the interior of R ;
- (iv) It is a point in the interior of an edge e_v of $VD(S)$, and in the interior of an edge e_r of R such that e_v and e_r intersect but not properly;
- (v) It is a proper intersection point between an edge e of $VD(S)$ and the boundary of R , but it is not the closest to either endpoint of e among all the intersection points between e and the boundary of R ;
- (vi) It is a proper intersection point between an edge e of $VD(S)$ and the boundary of R , and it is the closest to either endpoint of e among all the intersection points between e and the boundary of R ;
- (vii) It is a vertex of R ;
- (viii) It is a vertex of $VD(S)$ contained in R .

Notice the points mentioned in part 1, 2 and 3 of the lemma are the same as in case (viii), (vii) and (vi). Therefore to prove the lemma, we need only prove that the desired location of the facility cannot be any of the cases (i) to (v). We show that each of these cases leads to a contradiction. For simplicity of exposition, let c be the desired location of the facility.

Case (i) c is a point in the interior of R and in the interior of a cell of $VD(S)$.

Let $V(b)$ be the cell containing c . If $c = b$, let $x \neq c$ be any point in $R \cap V(b)$. Otherwise the line through c and b must intersect the boundary of R and $V(b)$, let x be the intersection point farthest to b within $V(b)$. Then the circle centered at x with radius $d(x, b)$ is empty. However, $d(x, b) > d(c, b)$ by construction. Hence, we have a contradiction.

Case (ii) c is a point in the interior of an edge $[pq]$ of R and in the interior of a cell of $VD(S)$. Let $V(b)$ be the cell containing c . If $c = b$, let $x \neq c$ be any point in $R \cap V(b)$. Otherwise either $\angle bcp$ or $\angle bcq$ is $\geq \pi/2$. Suppose $\angle bcp$ (resp. $\angle bcq$) $\geq \pi/2$. Let x be any point on line segment $[cp]$ (resp. $[cq]$) within $V(b)$. Now the circle centered at x with radius $d(x, b)$ is empty. However, $d(x, b) > d(c, b)$ by construction. Hence, we have a contradiction.

- Case (iii)** c is a point in the interior of an edge $e = [pq]$ of $BVD(S)$ and in the interior of R . Let $V(b)$ be either of the two cells incident on e . We have either $\angle bcp$ or $\angle bcq$ is $\geq \pi/2$. Suppose $\angle bcp$ (resp. $\angle bcq$) $\geq \pi/2$. Let x be any point on line segment $[cp]$ (resp. $[cq]$) within R . Now the circle centered at x with radius $d(x, b)$ is empty. However, $d(x, b) > d(c, b)$ by construction. Hence, we have a contradiction.
- Case (iv)** c is a point in the interior of an edge e_v of $VD(S)$, and in the interior of an edge e_r of R such that e_v and e_r intersect but not properly. Same argument as Case (iii).
- Case (v)** c is a proper intersection point between an edge e of $VD(S)$ and the boundary of R , but it is not the closest to either endpoint of e among all the intersection points between e and the boundary of R . Contradiction follows from Lemma 2.2. $\square$

2.3 The Algorithm

Based on Theorem 2.3, we outline the following algorithm to compute the location of the obnoxious facility constrained to a simply polygon.

Algorithm 1: *Single obnoxious facility location (S, R)*

Input: A set of demand points $S = \{s_1, s_2, \dots, s_m\}$, and a simple polygon R of n vertices.

Output: Point(s) for the location of a single obnoxious facility within R that maximizes the minimal distance between the facility and the demand points in S .

1. Compute $VD(S)$ and $BVD(S)$.
2. Preprocess R for point inclusion test using the algorithms of Kirkpatrick [22], or Edelsbrunner, Guibas and Stolfi [14].
3. Compute the set of vertices of $VD(S)$ contained in R using the point location structure of R , and compute their largest empty circles.
4. Preprocess R in $O(n)$ time for ray shooting query using the algorithm of Hershberger and Suri [20].
5. For each vertex v of $BVD(S)$ and each edge e of $BVD(S)$ incident on v , perform a ray shooting query over R , with a ray emanating from v and containing e [20]. If the hit point of the boundary of R by the ray is on e , compute its largest empty circle.
6. Preprocess $VD(S)$ for point inclusion test using the algorithms of [14] or [22].
7. For each vertex v of R , determine which Voronoi cell it is in using the point location structure of $VD(S)$, and compute its largest empty circle.
8. Select and output the largest empty circles found in steps 3, 5 and 7.

Theorem 2.4: *Given a set of m demand points $S = \{s_1, s_2, \dots, s_m\}$, and an n -vertex simple polygon R , we can compute the location of a single obnoxious facility constrained to lie in R , under Euclidean distance, in time $O((m+n)\log m + m\log n)$.*

Proof: The correctness of the algorithm follows from Theorem 2.3.

Let us analyze the complexity of the algorithm:

1. Step 1 of the algorithm can be computed in $O(m\log m)$ time because $VD(S)$ can be computed in $O(m\log m)$ time [27] and $BVD(S)$ can then be computed in $O(m)$ time.
2. Step 2 can be done in $O(n)$ time, as discussed in Section 2.2.1.
3. In step 3, for each vertex of $VD(S)$, we spent $O(\log n)$ time to determine if it is inside R [14, or 22], and constant time to compute its largest empty circle. There are $O(m)$ vertices in $VD(S)$, therefore step 3 takes $O(m\log n)$ time.
4. Step 4 is computed in $O(n)$ time [20].
5. In step 5, each ray shooting query takes $O(\log n)$ time [20], and computing the largest empty circle of a hit point takes constant time. Since the number of edges in $BVD(S)$ is $O(m)$, and we perform at most two ray shooting queries for each edge, the total time spent in step 5 is $O(m\log n)$.

6. $VD(S)$ is a planar subdivision, therefore we can use algorithms of [14] or [22] to preprocess $VD(S)$ for point inclusion test, thus step 6 takes $O(m)$ time.
7. In step 7, for each vertex of R , we spent $O(\log m)$ time to determine which Voronoi cell it is in using the point location structure of $VD(S)$, then constant time to compute its largest empty circle. There are n vertices in R , therefore step 7 takes $O(n \log m)$ time.
8. Step 8 can be computed in $O(m+n)$.

Therefore, the total complexity of the algorithm is $O((m+n) \log m + m \log n)$. $\square$

The largest empty circle problem has a known lower bound of $\Omega(m \log m)$ by reducing it to the MAX-GAP problem [27]. Thus Algorithm 1 runs in optimal $\Theta(m \log m)$ time when m is $\Omega(n)$, and in optimal $\Theta(n)$ time when m is $O(1)$. It remains an open problem whether the algorithm is optimal when m is $o(n)$ and $\omega(1)$.

2.4 Extension to Multiple Polygons

Algorithm 1 can be extended to the case where the constraining region R is a set of r disjoint simple polygons $\{R_1, R_2, \dots, R_r\}$ with a total of n vertices.

Extending Theorem 2.3, the following corollary characterizes the location of the obnoxious facility in this case.

Corollary 2.5: *The location of a single obnoxious facility in the case where the constraining region R consists of r disjoint simple polygons $\{R_1, R_2, \dots, R_r\}$ lies on one of the following points that yields the largest empty circle:*

1. *A vertex of $VD(S)$ contained in R ;*
2. *A vertex of a polygon $R_i \in R$;*
3. *A point of the boundary of a polygon $R_i \in R$ hit by a ray emanating from a vertex v of $BVD(S)$ and containing an edge of $BVD(S)$ incident on v .*

Thus, to compute the location of the obnoxious facility, all we need are a point location structure and a ray shooting structure for the region R of multiple polygons.

We can use a large rectangle B'' to enclose the region R and all vertices of $VD(S)$. Using Edelsbrunner, Guibas, and Stolfi's algorithm [14], the planar subdivision induced by $B'' \cap R$ can be preprocessed in $O(n \log n)$ time, so that it can be determined in $O(\log n)$ time whether a vertex of $VD(S)$ is inside R .

As mentioned in Section 1.3.3, we can build a ray shooting structure for R in time $O(n\sqrt{r} + n\log n + r^{3/2}\log r)$, so that a ray shooting query over R can be answered in time $O(\sqrt{r} \log n)$ [20].

Therefore, extending Algorithm 1 and Theorem 2.4, we obtain the following corollary:

Corollary 2.6: *Given a set of m demand points $S = \{s_1, s_2, \dots, s_m\}$, and a region R consists of r disjoint simple polygons $\{R_1, R_2, \dots, R_r\}$ with a total of n vertices, we can compute the location of a single obnoxious facility constrained to lie in R , under Euclidean distance, in time $O((m+n)\log m + (m\sqrt{r} + n)\log n + n\sqrt{r} + r^{3/2}\log r)$.*

Chapter 3 Euclidean Center Constrained to a Simple Polygon

3.1 Related Work

The problem of locating a single desirable facility for a set of demand points is also known as the Euclidean center problem. The problem of computing the unconstrained Euclidean center has a rich history, which is summarized in [27].

In [4] Bose and Toussaint solved the problem of computing the Euclidean center constrained to a simple polygonal region (see Section 1.4.4). Their algorithm finds the Euclidean center of a set of m points constrained to lie in a simple polygon P of n vertices in $O((m+n)\log(m+n))$ time when P is convex, and $O((m+n)\log(m+n) + k)$ time when P is not convex, where k denotes the number of intersections occurring between edges of P and edges of the farthest point Voronoi diagram of the demand points. Notice k could be $\Omega(mn)$.

An optimal $O(n)$ algorithm based on a prune-and-search strategy was given by Megiddo to solve the unconstrained Euclidean center problem [24].

Extending the technique of Megiddo [24], Hurtado, Sacristan and Toussaint obtained an $O(m + n)$ time algorithm for the problem when the Euclidean center is constrained to lie in a convex polygon ([21], also see Section 1.4.5), reducing a logarithmic factor from the algorithm presented in [4].

It can be shown that the Euclidean center problem constrained to a simple polygonal region is equivalent to the problem in the presence of polygonal obstacles. Halperin, Sharir and Goldberg studied the Euclidean 2-center problem in the presence of polygonal obstacles and obtained efficient algorithms [19]. In the paper they also mentioned that Halperin and Linhart have solved the 1-center problem with obstacles in $O((m+n)\log(mn))$ time [18]. However, their algorithm with the above mentioned time bound solved the decision problem only, and an approximation approach based on binary search is then used. Thus the algorithm is not exact. Moreover, the algorithm involves dynamic maintenance of the intersection of disks and is quite involved.

In this section we present an algorithm that is based on Bose and Toussaint's approach [4] to solve the Euclidean center problem constrained to a simple polygon. Our algorithm runs in $O(m\log(mn) + n\log m)$ time even if the constraining polygon is not convex, eliminating the dependency of the running time on k (the number of intersections between edges of P and edges of the farthest point Voronoi diagram of the demand points). Although the algorithm

in [18] has the advantage that it can be generalized to solve Euclidean 2-center problem, the merits of our algorithm compared to [18] are the following: (i) it is exact; ii) it is more efficient, considering the approximation scheme in [18] adds a multiplicative logarithmic factor to the running time. (iii) it is simpler and straightforward to implement, since it involves well-known structures only, including convex hull, farthest-point Voronoi diagram, point location structure and structure for ray shooting query.

3.2 Preliminaries

The problem is stated as the following:

Problem 2: *Given a set $S = \{s_1, s_2, \dots, s_m\}$ of m demand points and an n vertex simple polygon region R , locate a single desirable facility within R such that the largest Euclidean distance between each demand point and the facility is minimized.*

Such a location is called the Euclidean center of the demand points. It is assumed that the demand points are in general position, i.e. no three points are collinear and no four points are co-circular.

Bose and Toussaint's algorithm [4] makes use of the farthest point Voronoi diagram of the set S , denoted as $FPVD(S)$. Given a point $x \in R^2$, we let $\phi(x)$ denote the farthest neighbors of x in S , that is, the set of points in S such that $d(x, \phi(x)) = \max_{y \in S} d(x, y)$ where d is the Euclidean distance. The $FPVD(S)$ partitions the plane into unbounded convex cells, $V(s_i)$, such that for any point $p \in V(s_i)$, $s_i \in \phi(p)$.

The following lemma is adopted from [4] characterizing the location of the Euclidean center (restatement of Lemma 1.4):

Lemma 3.1 *The Euclidean center under the constraints stipulated in Problem 2 is either the midpoint of the diameter of S provided that the diametrical circle contains the set S and the midpoint is contained in R , or lies on one of the following points that yields the smallest enclosing circle:*

1. *A vertex of $FPVD(S)$ contained in R ;*
2. *A vertex of the polygon R ;*
3. *A proper intersection point of $FPVD(S)$ and the boundary of R ;*
4. *A point x on an edge e of R with the property that $\forall y \in e$, if $\phi(y) = \phi(x)$ then $d(y, \phi(x)) \geq d(x, \phi(x))$.*

The quadratic runtime occurs in computing part 3 and part 4 of the above. In the following we are going to prove several lemmas, Lemma 3.2 to 3.6, which will allow us to improve Bose and Toussaint's algorithm, reducing the quadratic runtime, by improving the characterization of the Euclidean center.

3.2.1 Improving Part 3 of Lemma 3.1

A similar approach as in Section 2.2.1 is used to improve part 3 of Lemma 3.1.

Lemma 3.2: *If the Euclidean center c under the constraints above lies on a proper intersection point c between an edge e of $FPVD(S)$ and the boundary of R , and if c is not the midpoint of $DIAM(S)$, then among all intersection points between e and the boundary of R , c is the closest to an endpoint of e .*

Proof: See Figure 7. Let $V(a)$ and $V(b)$ be the two cells incident on e . The edge e must lie on the bisector of line segment $[ab]$, for the points on e are equidistant from a and b . The points a , b , and c must form a triangle because otherwise $[ab]$ would be the diameter. Thus neither $[ca]$ nor $[cb]$ is perpendicular to e . If e is unbounded, let p be the unique endpoint of e . Otherwise, let p be the endpoint of e that is closer to a (and b). Then we have $\angle acp < \pi/2$. For the sake of contradiction, suppose c is not the closest to p among all intersection points

between e and the boundary of R . Let x be the closest to p . Now the circle centered at x with radius $d(x, a)$ encloses the set S . However, $d(x, a) < d(c, a)$. Hence, we have a contradiction. $\square$

Figure 7. Illustration for proof of Lemma 3.2

Lemma 3.2 indicates that we need not consider all the proper intersection points between $FPVD(S)$ and R . Rather, for all intersection points occurring between a given edge e of $FPVD(S)$ and the boundary of R , we need consider the points closest to an endpoint of e , of which there are at most two.

The intersection points with this property can be found by a ray shooting query over R ([20], also see Section 2.2.1). For each vertex v of $FPVD(S)$ and each edge e incident on v , we shoot a ray emanating from v and containing e . Let p be the hit point of the boundary of R by the ray. We then test if p is on e .

If p is not on e , then e does not intersect R and p is discarded. If p is on e , then p is the intersection point of e and the boundary of R that is the closest to v , a candidate with the property mentioned in Lemma 3.2. It is possible that some of these points are redundant, however, this does not affect the correctness nor the complexity of the algorithm we propose (see below).

Using the technique above allows us to avoid the expensive computation of all intersection points between edges of $FPVD(S)$ and the boundary of R in part 3 of Lemma 3.1. Next we develop another technique to efficiently compute part 4 of Lemma 3.1.

3.2.2 Improving Part 4 of Lemma 3.1

Lemma 3.3: *If the Euclidean center x under the constraints stipulated in Problem 2 lies on an edge e of R and has the following properties:*

- i) $\forall y \in e$, if $\phi(y) = \phi(x)$ then $d(y, \phi(x)) \geq d(x, \phi(x))$
- ii) x is not the midpoint of $DIAM(S)$
- iii) x is not a vertex of R
- iv) x is not an intersection point between the boundary of R and edges of $FPVD(S)$,

then the line segment $[\phi(x), x]$ is perpendicular to e , and $\phi(x)$ is the unique farthest point to the line containing e among all points in S .

Proof: Let $a = \phi(x)$, or equivalently, $x \in V(a)$. Let $[pq]$ be the portion of e that lies in $V(a)$ (Notice then p and q are either vertices of R or intersection points of e and $V(a)$), and $\forall x' \in e, \phi(x') = \phi(x)$ if and only if $x' \in [pq]$.

To prove the first part of the lemma, for the sake of contradiction suppose $[ax]$ is not perpendicular to e , then either $\angle axp$ or $\angle axq$ is $< \pi/2$. Without loss of generality suppose $\angle axp < \pi/2$. Let y be the projection point of a on e , if it exists, otherwise let $y = p$. Now the circle centered at y with radius $d(y, a)$ encloses the set S . However, $d(y, a) < d(x, a)$. Hence, we have a contradiction. Therefore $[ax]$ is perpendicular to e .

Let l denote the line containing e . To prove the second part (see Figure 8), for the sake of contradiction suppose there exists a point b of S , such that the distance between b and l is no less than that between a and l . Let y be the projection point of b on l . Given $d(y, b) \geq d(x, a)$, in the following we will prove $d(x, b) > d(x, a)$, which contradicts that $x \in V(a)$.

Figure 8. Illustration for proof of Lemma 3.3.

First, if i) $d(y, b) > d(x, a)$, or ii) $d(y, b) = d(x, a)$ and $x \neq y$, then $d(x, b) > d(x, a)$ because $d(x, b) \geq d(y, b)$ in the first case and $d(x, b) > d(y, b)$ in the second. Next, we will show it is impossible that $d(y, b) = d(x, a)$ and $x = y$. In this case b is the mirror image of a with respect to l and x is the midpoint of $[ab]$. Since both a and b are the farthest points of x , x lies on the common edge of $V(a)$ and $V(b)$. In other words, $[ab]$ crosses and is perpendicular to an edge of $FPVD(S)$. Thus $[ab]$ is $DIAM(S)$ and x is its midpoint. However, we know x is not a midpoint of $DIAM(S)$. Hence we have a contradiction. $\square$

Lemma 3.3 indicates that for each edge e of R , there is at most one point that can be qualified as a candidate for the constrained Euclidean center with the property mentioned in the lemma, despite the fact that e may span many cells of $FPVD(S)$. When there are two points in S that are both the farthest and equidistant to l , there is no point on e that can be qualified. It is worth pointing out that the converse of Lemma 3.3 is not true, i.e. if $a \in S$ is the unique farthest point from l and the projection point p of a is on e , it is not necessarily true that p has the property mentioned in Lemma 3.3, because it is possible that $p \notin V(a)$.

To find the point in S that is the farthest to a line, we make use of the convex hull of S . The convex hull $CH(S)$ of a set of points S is the smallest convex set that contains S . Or alternatively, it is the unique convex polygon whose vertices are points from S and that contains all points of S [27]. We observe the following:

Lemma 3.4: *The point in a given set of points S that is the farthest from a given line l is a vertex of the convex hull of S .*

Proof: Let p be the point of S that is the farthest to l . For the sake of contradiction suppose p is not a vertex of $CH(S)$. Then p must be in the interior of $CH(S)$. Let h be the half-plane containing p determined by l . The intersection of h and the interior of $CH(S)$ is also a convex polygon. Denote it by C . We

know p is in the interior C . Then there must be a vertex of C that is on the opposite side of l with respect to p . Let x be the vertex. Then x is further from l than p . We have a contradiction. $\square$

By Lemma 3.4, given a line, we need only compute the farthest point in $CH(S)$ to find out the farthest point in S . Let h be the number of edges of $CH(S)$. By preprocessing $CH(S)$ in $O(h)$ time we can find the unique farthest point of S to a given line in $O(\log h)$ time, as explained in the following.

Let $ang(e)$ be the angle of an edge e of $CH(S)$, measured as the angle between a right oriented horizontal ray x and e . The edges are directed in such a way that a walk along the directed edges constitutes a counterclockwise traversal of the boundary of $CH(S)$ (Figure 9). We store $ang(e)$ for each edge e of $CH(S)$ in sorted order in a set A . This can be done in $O(h)$ time given that edges in $CH(S)$ are in sorted order.

Figure 9. Finding the farthest point on the convex hull of S to a given line l .

Given a line l , to answer the query of the unique farthest point in $CH(S)$, we first compute the angle of l for both directions of l , denoted as $ang(l_\alpha)$ and $ang(l_\beta)$. For each of $ang(l_k)$ ($k = \alpha$ or β), we do a binary search in A to find out the two angles in A in between which $ang(l_k)$ falls. The two angles correspond to two neighboring edges in $CH(S)$, whose common vertex is then obtained. Denote by u and v the vertices obtained corresponding to $ang(l_\alpha)$ and $ang(l_\beta)$, respectively (Figure 9).

We note that u and v are actually an antipodal pair of $CH(S)$ [27], and the farthest point on $CH(S)$ to l has to be either of them. The proof is the following. Suppose x is the farthest point on $CH(S)$ to l . Let l' be the line through x and parallel to l . It can be shown by contradiction that $CH(S)$ must lie entirely on one side of l' . Notice u and v are the only two vertices on $CH(S)$ that have the

property that $CH(S)$ lies entirely on one side of the line through either of them and parallel to l .

Compute their respective distance to l . The vertex corresponding to the larger distance is then returned. If the two distances happen to be equal, by Lemma 3.3 we need not return a vertex.

During the binary search, if $ang(l_k)$ ($k = \alpha$ or β) happens to be equal to one of angles in A , which means there are two vertices in $CH(S)$ on the same side of l that are both the farthest to l , then again we need not return a vertex corresponding to $ang(l_k)$ according to Lemma 3.3.

To analyze the complexity, the binary search takes $O(\log h)$ time and all other computation takes constant time, thus query time is $O(\log h)$. Obviously $h = O(m)$, thus to compute Part 4 of Lemma 3.1, after preprocessing $CH(S)$ in $O(m)$ time, for each edge e of R , in $O(\log m)$ time we can find the unique point $p \in S$ that is the farthest from the line containing e , and compute the projection point of p on e . Therefore,

Lemma 3.5: *Given $CH(S)$, part 4 of Lemma 3.1 can be computed in $O(m + n \log m)$ time.*

There is an alternative approach to computing Part 4 of Lemma 3.1, which takes totally $O(m + n \log n)$ time after computing $CH(S)$, as explained in the following. The set A of angles of the edges of $CH(S)$ are computed as above. We then define two directed edges $e_{i\alpha}$ and $e_{i\beta}$ for each edge e_i of R , such that a walk through $e_{1\alpha}, e_{2\alpha}, \dots, e_{n\alpha}$ constitutes a counterclockwise traversal of the boundary of R , and a walk through $e_{1\beta}, e_{2\beta}, \dots, e_{n\beta}$ constitutes a clockwise traversal of the boundary of R . We sort the edges $e_{i\alpha}$, $1 \leq i \leq n$, in non-decreasing order of their angles (measured the same as above, i.e. the angle between a right oriented horizontal ray x and $e_{i\alpha}$). The edges $e_{i\beta}$ are sorted similarly as well. Suppose now the edges $\{e_{1\alpha}, e_{2\alpha}, \dots, e_{n\alpha}\}$ and $\{e_{1\beta}, e_{2\beta}, \dots, e_{n\beta}\}$ are in non-decreasing order of their angles, respectively. Now,

1. Find the two angles a_{j-1} and a_j in A such that $a_{j-1} < \text{ang}(e_{1\alpha}) < a_j$. Then a_{j-1} and a_j correspond to two neighboring edges in $CH(S)$, whose common vertex is a candidate of the farthest point to e_1 .

2. For $i = 2$ to n

Begin;

3. Start from a_j and go through consecutive angles in A , until we find the first angle a_x in A such that $\text{ang}(e_{i\alpha}) < a_x$. With a_{x-1} and a_x we obtain the point that is a candidate of the farthest point to e_i .

4. $j := x;$

End;

Going through all directed edges $e_{i\beta}$, $1 \leq i \leq n$, of R similarly, we can find the other candidates of the farthest points to each edge. Now for each edge e , we compare the two candidates of the farthest points (an antipodal pair of $CH(S)$), pick the farther one, and compute its projection point on e . Thus Part 4 of Lemma 3.1 is computed.

It should be noted that whenever we find an angle in A that is equal to $ang(e_{ik})$, $k = \alpha$ or β , by Lemma 3.3 we do not compute the farthest point to e_{ik} and proceed to the next edge. Also if the two farthest points on each side of e_i are equidistant from e_i , we do not compute the projection point.

To analyze the complexity, step 1 takes $O(h)$ time, where h is the number of edges of $CH(S)$. In the loop, in step 3 we make $k_i + 1$ comparisons, where $k_i = x - j$ is the number of times we cross over an angle in A . Notice that $\sum_i k_i$ is no greater than $h - 1$, because both the angles of the edges of R and the angles in A are in non-decreasing order, thus each angle in A can be crossed over at most once. Therefore we make

$$\sum_{i=2}^n (k_i + 1) = \sum_{i=2}^n k_i + (n-1) \leq (h-1) + (n-1) = O(m+n)$$

comparisons. Step 4 takes constant time for each edge. Therefore the total time spent in the loop is $O(m+n)$. The running time of the algorithm is thus $O(m+n)$. Considering we need $O(n \log n)$ time to sort the edges of R in order of their angles, the above discussion leads to the following lemma:

Lemma 3.6: *Given $CH(S)$, part 4 of Lemma 3.1 can be computed in $O(m+n \log n)$ time.*

3.2.3 Characterization of the Constrained Euclidean Center

To summarize, the following lemma characterizes the location of the Euclidean center constrained to a simple polygon.

Theorem 3.7: *The Euclidean center under the constraints above is either the midpoint of the diameter of S provided that the diametrical circle contains the set S and the midpoint is contained in R , or lies on one of the following points that yields the smallest enclosing circle:*

1. *A vertex of $FPVD(S)$ contained in R ;*
2. *A vertex of R ;*

3. *A point of the boundary of R hit by a ray emanating from a vertex v of $FPVD(S)$ and containing an edge of $FPVD(S)$ incident on v ;*
4. *A projection point on an edge e of R of a point in S that is the farthest to e .*

Proof: The correctness of the lemma follows from Lemma 3.1, 3.2 and 3.3. $\square$

3.3 The Algorithm

Based on Theorem 3.7, we outline the following algorithm to compute the Euclidean center constrained to a simply polygon.

Algorithm 2: *Euclidean Center (S, R)*

Input: A set of demand points $S = \{s_1, s_2, \dots, s_m\}$, and a simple polygon R of n vertices.

Output: Euclidean center of S constrained to lie in R .

1. Compute $CH(S)$.
2. Compute $DIAM(S)$.
3. Compute the circle C having $DIAM(S)$ as diameter.
4. Preprocess R for point inclusion test using the algorithms of Kirkpatrick [22], or Edelsbrunner, Guibas and Stolfi [14].
5. If the midpoint of $DIAM(S)$ is contained in R and all points of S are contained in C then exit with the midpoint of $DIAM(S)$ and radius of C .

6. Compute $FPVD(S)$.
7. Compute the set of vertices of $FPVD(S)$ contained in R using the point location structure of R , and compute their smallest enclosing circles.
8. Preprocess $FPVD(S)$ for point inclusion test using the algorithms of [14] or [22].
9. For each vertex v of R , determine which Voronoi cell it is in using the point location structure of $FPVD(S)$, and compute its smallest enclosing circle.
10. Preprocess R in $O(n)$ time for ray shooting query over R [20].
11. For each vertex v of $FPVD(S)$ and each edge e of $FPVD(S)$ incident on v , perform a ray shooting query over R , with a ray emanating from v and containing e [20]. If the hit point of the boundary of R by the ray is on e , compute its smallest enclosing circle.
12. Store in A in sorted order the set of angles of each edge of $CH(S)$. For each edge e of R , find the unique farthest point p of $CH(S)$ to e , and compute the projection point of p on e , using the techniques of Lemma 3.5. Determine which Voronoi cell p is in using the point location structure of $FPVD(S)$, and compute its smallest enclosing circle.
13. Select and output the smallest enclosing circles found in steps 7, 9, 11 and 12.

Theorem 3.8: *Given a set of demand points $S = \{s_1, s_2, \dots, s_m\}$, and an n -vertex simple polygon R , we can compute the Euclidean center of S constrained to lie in R in time $O((m+n)\log m + m\log n)$.*

Proof: The correctness of the algorithm follows from Theorem 3.7.

Let us analyze the complexity of the algorithm.

1. Step 1 can be computed in $O(m\log m)$ time [27].
2. Given $CH(S)$, step 2 can be computed in $O(m)$ time by computing the diameter of $CH(S)$ [27].
3. Step 3 is done in constant time.
4. Step 4 can be done in $O(n)$ time as discussed in Section 2.2.1.
5. Step 5 can be done in $O(\log n + m)$ time using the point inclusion test as well as going over all points of S to test if they are contained in C .
6. Step 6 of the algorithm can be computed in $O(m\log m)$ time [27].
7. In step 7, for each vertex of $FPVD(S)$, we spent $O(\log n)$ time to determine if it is inside R [14, or 22], and constant time to compute its smallest enclosing circle. There are $O(m)$ vertices in $FPVD(S)$, therefore step 7 takes $O(m\log n)$ time.
8. $FPVD(S)$ is a planar subdivision, therefore we can use algorithms of [14] or [22] to preprocess $FPVD(S)$ for point inclusion test, thus step 8 takes $O(m)$ time.

9. In step 9, for each vertex of R , we spent $O(\log m)$ time to determine which Voronoi cell it is in using the point location structure of $FPVD(S)$, then constant time to compute its smallest enclosing circle. There are n vertices in R , therefore step 9 takes $O(n \log m)$ time.
10. Step 10 is computed in $O(n)$ time [20].
11. In step 11, each ray shooting query takes $O(\log n)$ time [20], and computing the smallest enclosing circle of a hit point takes constant time. Since the number of edges in $FPVD(S)$ is $O(m)$, and we perform at most two ray shooting queries for each edge, the total time spent in step 5 is $O(m \log n)$.
12. Given $CH(S)$, step 12 can be computed in $O(m + n \log m)$ by Lemma 3.5.
13. Step 13 can be computed in $O(m+n)$.

Therefore, the total running time of the algorithm is $O((m+n) \log m + m \log n)$. $\square$

Alternatively, in step 12 we can use the technique of Lemma 3.6, which takes $O(m + n \log n)$ time, then the algorithm runs in $O((m+n) \log(m+n))$ time, which is not as efficient.

When m is $O(1)$, Algorithm 2 runs in optimal $\Theta(n)$ time. It remain open problems whether Algorithm 2 is optimal when m is $\omega(1)$ or whether there exists linear time solution to the problem.

3.4 Extension to Multiple Polygons

Similar to Section 2.4, Algorithm 2 can be extended to the case where the constraining region R is a set of r disjoint simple polygons $\{R_1, R_2, \dots, R_r\}$ with a total of n vertices. All we need are a point location structure and a ray shooting structure for the region R of multiple polygons. As in Section 2.4, the former can be built in $O(n \log n)$ time for logarithmic time point location test [14], and the latter can be built in $O(n\sqrt{r} + n \log n + r^{3/2} \log r)$ time, so that a ray shooting query over R can be answered in time $O(\sqrt{r} \log n)$ [20].

Therefore, extending Algorithm 2 and Theorem 3.8, we obtain the following corollary.

Corollary 3.9: *Given a set S of m demand points, and a region R consists of r disjoint simple polygons with a total of n vertices, we can compute the location of a single desirable facility constrained to lie in R , under Euclidean distance, in time $O((m+n) \log m + (m\sqrt{r} + n) \log n + n\sqrt{r} + r^{3/2} \log r)$.*

Chapter 4 Conclusion

We have presented two efficient algorithms, one for locating a single obnoxious facility constrained to a simple polygonal region, and the other for locating a single desirable facility constrained to a simple polygonal region. The algorithms make use of the Voronoi diagram or the farthest-point Voronoi diagram. They are based on and are an improvement of the algorithms in [31] and [4]. Given a set S of m demand points and a simple polygon R of n vertices, we have shown how to compute the location of either an obnoxious or a desirable facility to lie in R , both in $O((m+n)\log m + m\log n)$ time. We have also extended both algorithms to the case where the constraining region consists of multiple simple polygons.

The improved algorithms are a result of a deeper perception of the structures of the problem. The innovation in the techniques developed in the work includes using ray shooting queries instead of computing all the intersection points between the Voronoi diagram (or farthest point Voronoi diagram) and a simple polygon. The technique may be applicable to other problems involving simple polygons.

It is worth noticing that both algorithms can be applied to problems with some seemingly different constraints, for example, to locate the facility in the presence of polygonal obstacles, or in certain simple polygonal region with holes. As an illustration, let the feasible region P be a rectangle B in the presence of a simple polygonal obstacle R with n vertices, that is, $P = B - R$. Referring back to Figure 6, in $O(n)$ time we can divide P into two simple polygonal regions R_1 and R_2 of $O(n)$ vertices. By solving the problem of locating the facility in R_1 and R_2 separately using the algorithms in this report, and pick the better of the two solutions, we solve the original problem of locating a facility constrained to P .

References

- [1] B. Ben-Moshe, M. J. Katz, and M. Segal, Obnoxious Facility Location: Complete Service with Minimal Harm, *Proc. 11th Canadian Conf. on Computational Geometry*, pp. 76-79, 1999.
- [2] J. L. Bentley and T. A. Ottmann, Algorithms for Reporting and Counting Geometric Intersections, *IEEE Transactions on Computers*, 9(C-28):643-647, 1979.
- [3] M. d. Berg, M. V. Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*. Springer, Berlin, 2000.
- [4] P. Bose and G. T. Toussaint, Computing the constrained Euclidean, geodesic and link centers of a simple polygon with applications, *Proceedings of Computer Graphics International (CGI'96)*, Pohang, Korea, 1996.
- [5] J. Brimberg and A. Mehrez, Multi-Facility Location Using a Maximin Criterion And Rectangular Obstacles, *Location Science*, 2(1):11-19, 1994.
- [6] P. Cappanera, A Survey on Obnoxious Facility Location Problems, *Technical Report*, Dipartment di Informatica, Universita Di Pisa, Number TR-99-11, April 19, 1999.

- [7] T. M. Chan, A Simple Trapezoid Sweep Algorithm for Reporting Red/Blue Segment Intersections, *Canadian Conference in Computational Geometry*, pp. 263-268, 1998.
- [8] B. Chazelle, Triangulating a simple polygon in linear time, *Discrete Comput. Geom.*, 6:485-524, 1991.
- [9] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink, Ray shooting in polygons using geodesic triangulations, in *Proc. of the 18th ICALP*: Springer-Verlag, 1991, pp. 661-673.
- [10] B. Chazelle and L. Guibas, Visibility and intersection problems in plane geometry, *Discr. Comput. Geom.*(4):551-581, 1989.
- [11] R. L. Church and R. S. Garfinkel, Locating an Obnoxious Facility on a Network, *Transportation Science*, 12:107-118, 1978.
- [12] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. MIT Press, Cambridge, 1998.
- [13] Z. Drezner, *Facility Location, a Survey of Applications and Methods*. Springer Series in Operations Research, Springer, 1995.
- [14] H. Edelsbrunner, L. J. Guibas, and J. Stolfi, Optimal Point Location in a Monotone Subdivision, *SIAM J. Comput.*, 15:317-340, 1986.

- [15] S. Fortune, A Sweepingline Algorithm for Voronoi Diagrams, *Proceedings of the second annual ACM SIGACT/SIGGRAPH symposium on Computational geometry*, pp. 313-322, 1986.
- [16] G. N. Frederickson and D. B. Johnson, Generalized Selection and Ranking: Sorted Matrices, *SIAM J. Comput.*, 13:14-30, 1984.
- [17] A. J. Goldman and P. M. Dearing, Concepts of Optimal Location for Partially Noxious Facilities, *ORSA Bulletin*, 23(1):B-31, 1975.
- [18] D. Halperin and C. Linhart, The Smallest Enclosing Disks with Obstacles, *Manuscript*, 1999. Website: <http://www.math.tau.ac.il/~halperin>.
- [19] D. Halperin, M. Sharir, and K. Goldberg, The 2-Center Problem with Obstacles, *Proceedings of the sixteenth annual symposium on Computational geometry*, pp. 80-90, 2000.
- [20] J. Hershberger and S. Suri, A Pedestrian Approach to Ray Shooting: Shoot a Ray, Take a Walk, *Journal of Algorithms*, 18(3):403-431, 1995.
- [21] F. Hurtado, V. Sacristán, and G. Toussaint, Constrained Facility Location, *Proceedings of the 10th Canadian Conference on Computational Geometry*, pp. 24-25, 1998.
- [22] D. Kirkpatrick, Optimal Search in Planar Subdivisions, *SIAM J. Comput.*, 12(1):28-35, 1983.

- [23] R. J. Lipton and R. E. Tarjan, Applications of a planar separator theorem, *SIAM J. Comput.*, 9:615-627, 1980.
- [24] N. Megiddo, Linear-time Algorithms for Linear Programming in R^3 and Related Problems, *SIAM J. Comput.*, 12:759-776, 1983.
- [25] P. Mirchandani and R. Francis, *Discrete Location Theory*. John Wiley and Sons, Inc., New York, 1990.
- [26] K. Mulmuley, A Fast Planar Partition Algorithm, I, *Journal of Symbolic Computation*, 10:253-280, 1990.
- [27] F. P. Preparata and M. I. Shamos, *Computational Geometry: an Introduction*. Springer-Verlag, New York, 1985.
- [28] N. Sarnak and R. E. Tarjan, Planar Point Location Using Persistent Search Tree, *Commun. ACM*(29):669-679, 1986.
- [29] M. I. Shamos, Computational Geometry, Ph.D Thesis, Yale University, 1978.
- [30] M. Sharir, A Near-Linear Algorithm for the Planar 2-center Problem, *Discrete Comput. Geom.*, 18:125-134, 1997.
- [31] G. T. Toussaint, Computing Largest Empty Circles with Location Constraints, *International Journal of Computer and Information Science*, 12(5):347-358, 1983.