FACILITY LOCATION IN THE PRESENCE OF RECTANGULAR OBSTACLES

by

Rizwan M.H. Merchant

B.C.S. University of Pune, 1998

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
May 2003

APPROVAL

Name:	Rizwan M.H. Merchant
Degree:	Master of Science
Title of thesis:	Facility Location in the Presence of Rectangular Obstacles.

Examining Committee: Dr. Ke Wang
Chair

Dr. Binay Bhattacharya, Senior Supervisor.

Dr. Ramesh Krishnamurti, Supervisor.

Dr. Daya Ram Gaur, External Examiner.

Date Approved: May 1, 2003

Facility Location in the Presence of Rectangular Obstacles

Abstract

The area of facility location has applications in a wide range of topics such as operations research, urban planning, telecommunications, VLSI circuit design, robot motion, supply chain management, etc. We study three facility location problems in the presence of obstacles. The obstacles are disjoint, rectangular and axis-parallel. The distance between any two points in the plane is measured in L_1 geodesic metric, i.e., the shortest path in rectilinear metric avoiding obstacles.

In this thesis we study three seemingly unrelated problems. We identify commonalities between these problems and improve upon the existing results by using geometric techniques.

In the 1-median problem, we are given m obstacles and n source points in the plane, and the objective is to find a point t in the plane such that the sum of its distances to all the n points is minimized. We present efficient algorithms to solve the 1-median problem based on the relative orders of n and m.

In the shop floor layout problem, we are given a layout consisting of several demand facilities and a single supply facility. The supply facility serves each of the demand facilities through an I/O point or "door" on each of the demand facilities. The optimal placement of the supply facility and the doors on the demand facilities is vital in order to minimize the total transportation costs between them. We study three different versions of this problem: (i) the supply facility is a fixed point and its location is known, (ii) the supply facility is a point and can lie anywhere in the given layout, and (iii) the supply facility is a point and can lie anywhere within a given convex rectilinear region. In this thesis, we present efficient algorithms to solve the above versions of the shop-floor layout problem.

Finally, we study the clustering problem. The objective is to partition a given set of data points into clusters such that data points within the same cluster are "similar". We measure the similarity between data points in terms of distance functions. In this thesis, we present an efficient implementation of the iterative k-median clustering algorithm.

Acknowledgments

It is a pleasure to take this opportunity to extend my gratitude to the people without whom this work would be impossible. Firstly, I would like to thank my mentor and supervisor, Dr.Binay Bhattacharya, for his continuous guidance and support. Despite the many, many questions he showed tremendous patience and provided me with invaluable teachings and direction. I am extremely glad to have decided to work with Binay and am indebted to him. I am thankful to my colleague and friend, Robert, who was always willing to share his knowledge and ideas during the entire course of my research.

On a personal note, I feel that all accomplishments would be incomplete if I did not mention my parents, who have made and continue to make innumerable sacrifices to ensure that their children are provided with every opportunity to prosper. I thank them deeply for the love, encouragement and support they have always shown. If I could do for them even a fraction of what they have done for me, I would consider myself successful. To Rahael, who has been my greatest source of motivation - thanks for always being there to instill in me hope and optimism when I needed it the most. I would like to mention Hussein, who has been such a good friend and room-mate for the last seven years. I am also grateful to my close friend Vineet for helping me out in times of utmost need. Finally, I would like to thank all my family and friends for their support and encouragement.

To my parents...

"Obstacles are those frightful things you see when you fail to focus on your goals"

— Henry Ford

Contents

List of Figures

Chapter 1

Introduction

1.1 A note on Facility Location Problems

Facility location has been an intriguing and attractive field of study for many researchers for decades. This is not surprisingly so. The problems of facility location find applications in a wide range of real-life applications. Operations research, urban planning, telecommunications [11], VLSI circuit design [25, 28], robot motion [26, 5], supply chain management [11], etc are only some of the areas that borrow techniques and concepts from the field of facility location to achieve their goals.

So what is facility location?

Despite the amount of work and effort that goes into facility location, the objective of such a problem is in fact quite simple. We are given several facilities or resources which are to be utilized by a set of clients. The goal is to place these resources in an optimal fashion such that the total cost involved for the clients to access these resources is minimized. The cost can be measured in terms of minimum/maximum distance, monetary gain, etc and depends on the application under consideration. Simply put, facility location is a resource allocation optimization problem.

Consider a classic application of facility location: urban planning. It is often required to place certain facilities in and around the city, keeping in mind the access criteria to these facilities. For example, a fire station in the downtown area should ideally be located such that the travel time to any of its service locations is the same, i.e., at the center of its service area. The quality of solutions to such facility location problems are usually measured in terms of a travel metric, and the goal is to optimally place the given facilities so as to minimize the distances travelled or material transportation costs involved. Depending on the application, a suitable metric is chosen to calculate the distances. Euclidean and Manhattan are the two common distance metrics used.

When tackling real-life facility location problems, it is commonly seen that some restrictions apply on how and where facilities can be placed in a given region or space. The presence of obstacles results in forbidden regions where the facilities cannot be placed. The obstacles also pose a barrier to travel. These obstacles could appear in the form of rivers, lakes, parks, buildings, airports, etc. In such cases, it is imperative that the distance function be modified to acknowledge the presence of these obstacles when calculating distances between points in the space. Some applications where such a problem may arise are:

- Motion planning in the field of robotics where a robot tries to find its way through a maze of obstacles.
- Urban planning where the goal is to place services or facilities such as mail boxes, fire stations, etc. A typical obstacle could be a golf course inside which a mail box cannot be placed.
- In an industrial plant where a supply unit needs to be placed to service requests from demand facilities. In this case, the demand facilities pose as obstacles and travel cannot occur through these obstacles.
- In the design of VLSI circuits.

Drezner and Hamacher [11] present many facility location problems and the underlying theory behind solving them.

Concepts from computational geometry provide efficient algorithms and elegant data structures to solve problems arising in facility location. For instance, Preparata and Shamos [29] used the properties of the convex hull to produce an algorithm to solve the "smallest enclosing circle of a set of points" problem in $O(n \log n)$ time, where n is the number of points in the set. The Voronoi diagram, which is one of the fundamental structures described in computational geometry, has been repeatedly used to solve many facility location problems such as the "nearest neighbor query problem for the rectilinear metric in the presence of obstacles" [9].

In this thesis, we study three facility location problems: (i) the rectilinear 1-median problem as described by Choi *et. al.* [8], (ii) the "shop floor layout problem" as described by Wang *et. al.* [35], and (iii) the iterative k-median clustering problem in the presence of obstacles [33, 36].

The three problems mentioned above are facility location problems. Apart from this commonality, these problems share a strong relationship with each other in terms of our work and objectives. We started our research studying the clustering problem in the presence of obstacles. We realized that the k-means clustering approach falters in the presence of obstacles as it can place facilities (cluster centres) in forbidden regions, such as in the interior of an obstacle. This led us to pursue a different approach, and we started studying the 1-median problem in the presence of obstacles. Not only could we use the 1-median technique for our clustering process, but we were also able to improve the currently published algorithms solving the 1-median problem. As we will see later, the shop floor layout problem is very similar to the 1-median problem in the presence of obstacles. The objective of both these problems is to minimize sum of distances from a central point to other points in the region. We consider our obstacles to be axis parallel rectangular objects that occupy space and are a barrier to travel. We use the obstacle avoiding (geodesic) rectilinear (or Manhattan) L_1 metric to compute distances between pairs of points in the region.

The first problem we study is the rectilinear 1-median problem in the presence of obstacles. Here we are given n source points in the plane and a set of m rectangular obstacles. As before, these obstacles pose as barriers to travel and are axis-parallel. The objective is to find a point p such that the sum of the geodesic distances from each source point to p is minimized. The point p is called the 1-median point. This problem has been studied by Choi *et. al.* [8] who gave an $O(n(n+m)\log m)$ algorithm to locate a 1-median point. In this thesis, we present an $O(n(n+m))$ time algorithm for locating a 1-median point. We further improve the running time to $O(mn \log n)$ under the assumption that the number of source points is much larger than the number of obstacles in the plane, i.e., $n \gg m$. Also, our algorithm requires linear storage space, which is an improvement over the $O(mn)$ storage space requirement for the algorithm in [8].

Next we study the shop floor layout problem introduced in [35]. We are given a central supply facility and several demand facilities in the 2-D plane. The facilities are rectangular in shape and of finite size. They also pose as barriers to travel in the layout. The supply facility acts as a server of goods or resources to the demand facilities. The goods can be received by a demand facility at an I/O point or "door", which is placed somewhere on its perimeter. A door is also placed on the perimeter of the supply facility from where the goods are to be dispatched. Several versions of this problem have been studied. For instance, the supply facility can be a fixed point in the plane (also known as a supply point). In this case, the door to the supply facility would be the point itself. In another version, the location of the supply facility is unknown, and is computed as part of the solution set. Yet another version requires the supply facility to lie within a given convex rectilinear region, and the optimal location inside this region also needs to be determined.

Given the problem definition, the goal is to find an I/O point on the supply facility and an I/O point on each of the demand facilities. These I/O points need to located such that the total cost of transportation from the supply facility to the demand facilities is minimized, i.e., the sum of the distances from the I/O point of the supply facility to the I/O points of the demand facilities is minimum. The facilities are a barrier to travel and any travel path cannot intersect the interior region of a facility, but the paths are allowed to run along the perimeters of the facilities. We assume that the facilities are rectangular objects which are axis-parallel, i.e., their sides are parallel to the X and Y axes. All distances are measured using the rectilinear metric. We consider the different versions of this problem, and provide algorithms which have significantly better running times than those described by Wang *et. al.* [35].

Finally we examine the clustering problem. We are given a set of n data points, a set of m disjoint, rectangular, axis-parallel obstacles, a value k and an objective function. The goal is to partition the set of data points into k disjoint clusters, such that the objective function is minimized. Each cluster is represented by a cluster centre and all points in a cluster are closer to its cluster centre that to any other cluster centre. The distances are measured using the geodesic rectilinear metric.

This thesis is organized as follows. The rest of this chapter discusses the preliminaries, which contains several definitions and results that we use through the rest of the study to solve the above problems. In chapter 2, we discuss the rectilinear 1-median problem and present algorithms to solve the same. In chapter 3, we discuss in detail the shop floor layout problem and present algorithms to solve different versions of the problem. Chapter 4 examines the iterative k-median clustering problem, and an efficient algorithm is presented to solve this problem in the presence of obstacles. In chapter 5, we present our conclusions and discuss possible future work in the area.

Choice of Metric

In this thesis, we calculate distances between points using geodesic (obstacle avoiding) Manhattan metric, which is also known as the rectilinear metric. Even in the absence of obstacles, there is no exact algorithm for the 1-median problem in the plane when distances are computed using the Euclidean metric. This is due to the presence of square roots in the Euclidean metric, which could be irrational numbers [3]. Thus, there is no published result on the k-median problem for points in the plane. In contrast, objective functions that comprise of Manhattan distances are easier to solve.

1.2 Preliminaries

Given the Euclidean space $\mathcal{R}^2$, which have co-ordinate axes X and Y, a point p in the space is denoted by (p_x, p_y), where p_x is the X-coordinate and p_y is the Y-coordinate.

We define the distance between any two points p and q, $p, q \in \mathcal{R}^2$, as the L_1 or Manhattan distance between them, i.e., $||p - q|| = |p_x - q_x| + |p_y - q_y|$.

Also given are m obstacles (in 2-D) denoted by $\mathcal{O} = \{O_1, O_2, O_3, \ldots, O_m\}$, such that none of the obstacles intersect each other. The obstacles are rectangular and axis-parallel, i.e., their sides are parallel to the X and Y axes. We define the border of any region $Q \in \mathcal{R}^2$ as the boundary of that region, denoted by $B(Q)$. Thus the border of an obstacle O_i is its perimeter. We also define the free space $\mathcal{F}$ to be the region in $\mathcal{R}^2$ minus the set of obstacles $\mathcal{O}$, plus the border of each of the obstacles in $\mathcal{O}$, i.e., $\mathcal{F} = \mathcal{R}^2 - \bigcup\{O_i - B(O_i) | O_i \in \mathcal{O}\}$. We consider only rectilinear paths between the points. A rectilinear path is a chain of axis-parallel segments lying in the free space $\mathcal{F}$. Note that the borders of the obstacles of $\mathcal{O}$ are included in the free space and thus a rectilinear path can consists of segments in $B(O_i), O_i \in \mathcal{O}$. The length of such a path is the sum of the lengths of its segments. The shortest rectilinear path between two points p and q ($p, q \in \mathcal{F}$) can now be defined as the rectilinear path having minimum length, amongst geodesic rectilinear possible paths between p and q, and its length is denoted by $d(p, q)$. We define the x-distance between the points p and q as the sum of the lengths of the segments of the path from p to q that are parallel to the X-axis, denoted by $d_x(p, q)$. The y-distance between p and q is similarly defined. We denote the top-left, top-right, bottom-left and bottom-right corners of an obstacle $O_i \in \mathcal{O}$ by tl(O_i), tr(O_i), bl(O_i) and br(O_i) respectively.

We denote a path P by a sequence of end points of its segments, i.e., $P = p_0, p_1, \ldots p_k$, such that the coordinates of p_i and p_{i+1} differ in exactly one coordinate (X or Y). In other words, any two consecutive points in the path have either the same x-coordinate value or the same y-coordinate value. A path is said to be X-monotone (or Y-monotone) if the coordinates of its points continually increase or decrease in the X-direction (resp., Y-direction). An X-monotone (resp., Y-monotone) path has all of its horizontal (resp., vertical) segments directed either rightwards or leftwards(resp., either upwards or downwards). A path that is both X-monotone and Y-monotone is called an XY-monotone path, also known as a *stair-case path.* The length of a stair-case path is simply the L_1 distance between the two end points p and q, i.e., $d(p,q) = |p_x - q_x| + |p_y - q_y|$.

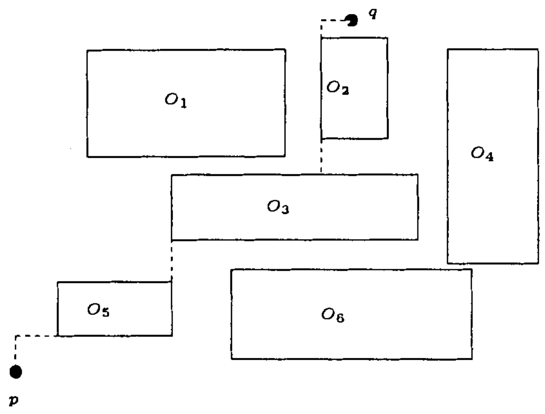

Figure 1.1: Example of a XY-monotone, rectilinear path.

In figure 1.1, the path from point p to the point q is rectilinear, and is monotone in both, the X and Y directions, i.e., it is XY-monotone. Notice that the path can run along the perimeter of some of the obstacles.

We now present some important properties of rectilinear paths between points which are used extensively later.

Property 1.2.1. *[4] Given two points p and q ($p, q \in \mathcal{F}$), the shortest path between them has to be either X-monotone or Y-monotone (or both).*

Property 1.2.2. *[4] Given two points p and q ($p, q \in \mathcal{F}$), if there exists an X-monotone path and a Y-monotone path between them, then there also exists an XY-monotone path between them.*

Property 1.2.3. *[4] If there exists an XY-monotone path between two points p and q, then that path is the shortest path between p and q. Further, any other shortest path between p and q has to be XY-monotone.*

Definition 1. We define an +x(+y) path $P = (p_0, p_1, \dots p_k)$ starting at a point p as follows. Starting from the point p, the path moves in the +x-direction, i.e., the first segment (p_0, p_1) of the path P is a horizontal line segment with $y = y(p)$. The path continues to move in the +x-direction until it hits an obstacle $O_i \in \mathcal{O}$. The path then moves along the edge of the obstacle O_i in the +y-direction till it reaches the top-left corner of O_i, after which it starts to move in the +x-direction again. This continues until the last segment of the path (p_{k-1}, p_k), which goes to infinity. $+x(-y)$, $-x(+y)$ and $-x(-y)$ paths are similarly defined.

As seen in figure 1.2, the paths defined above divide the free space $\mathcal{F}$ into four separate regions, $\mathcal{F}_{p(x)}$, $\mathcal{F}_{p(-x)}$, $\mathcal{F}_{p(y)}$ and $\mathcal{F}_{p(-y)}$. $\mathcal{F}_{p(x)}$ is the region of $\mathcal{F}$ that is enclosed by the xy and the $x(-y)$ paths.

The division of the plane into the four regions with respect to p as mentioned above imposes some properties on the distance measurements between p and the regions. The shortest path between p and any point inside $\mathcal{F}_{p(x)}$ and $\mathcal{F}_{p(-x)}$ is non-y-monotone, i.e., the path is x-monotone but not y-monotone. The shortest path between p and any point inside $\mathcal{F}_{p(y)}$ and $\mathcal{F}_{p(-y)}$ is either xy-monotone or just y-monotone. Analogously, we can create regions by drawing y(+x), y(-x), -y(+x) and -y(-x) paths so that the shortest path between p and any point inside $\mathcal{F}_{p(y)}$ and $\mathcal{F}_{p(-y)}$ is non-x-monotone, and the shortest path between p and any point inside $\mathcal{F}_{p(x)}$ and $\mathcal{F}_{p(-x)}$ is xy-monotone or just x-monotone (see figure 1.3).

Lemma 1.2.1. *[30] Given a point p and the obstacle set $\mathcal{O}$, the free space $\mathcal{F}$ can be divided into the four regions as described above in $O(n \log n)$ time and using $O(n)$ space.*

Lemma 1.2.2. *[8] Given two points p and q on a line segment L lying in the free space $\mathcal{F}$, and another point $s \in \mathcal{F}$, then (i) $d_x(s,p) = d_x(s,q)$, if L is a vertical line, and (ii) $d_y(s,p) = d_y(s,q)$, if L is a horizontal line.*

Figure 1.4 illustrates lemma 1.2.2 when L is a vertical line. Points p and q lie on L. The x-distance from s to p is equal to the x-distance from s to q, irrespective of

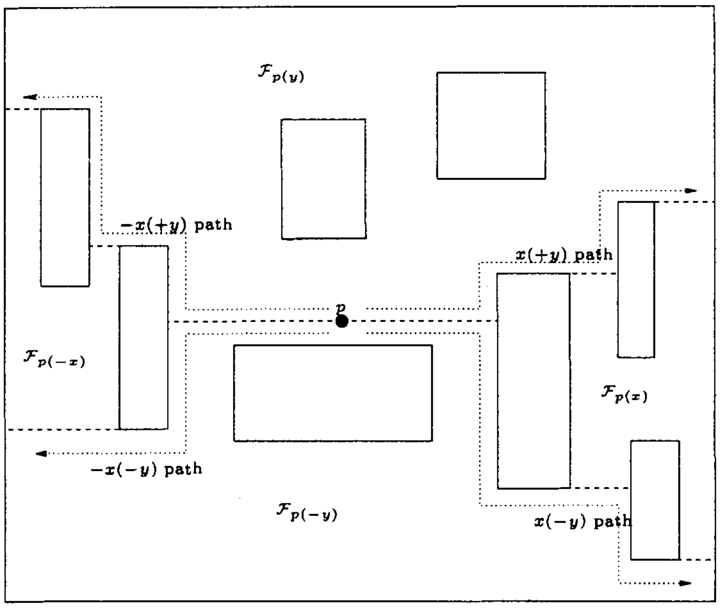

Figure 1.2: Partitioning of the free space into four regions - creating non-y-monotone regions with respect to a point.

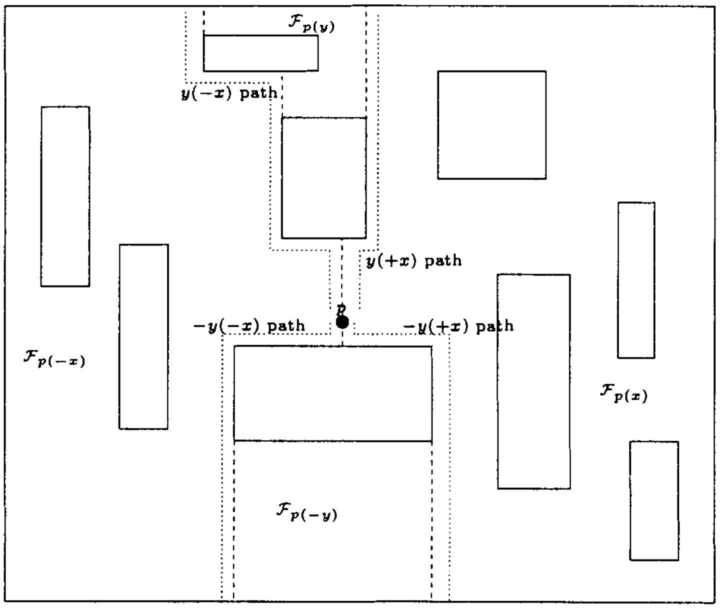

Figure 1.3: Partitioning of the free space into four regions - creating non-x-monotone regions with respect to a point.

where p and q lie on L.

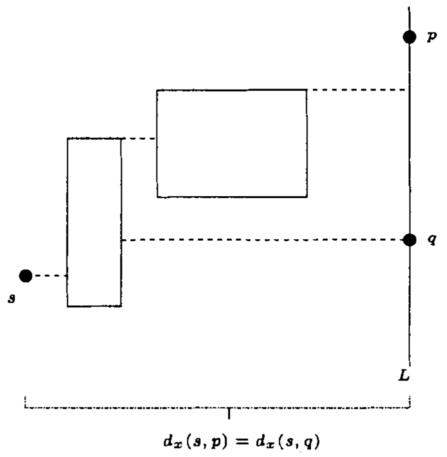

Figure 1.4: A diagram supporting lemma 1.2.2

1.3 Summary

We have presented several definitions and properties that we will use to solve the problems described in the subsequent chapters. In each chapter we provide a brief overview of the problem, followed by new and efficient algorithms which improve upon the previously published results. At the end of each chapter we provide a detailed analysis of our algorithms.

Chapter 2

The Constrained 1-Median Problem

2.1 Background and Problem Definition

The 1-median problem has been studied extensively and has applications in a wide range of areas, particularly in facility location [8, 15, 7, 20]. Given n source points in the plane and a set of m disjoint rectangular axis-parallel obstacles (as described in section 1.2), the goal of the 1-median problem is to find a point t that minimizes the distance function below:

$$D(t) = \sum_{i=1}^{n} d(s_i, t)$$

where s_i is the i^{th} source point and t is the median point such that t lies in the free space described by the plane and the obstacles (refer to section 1.2). The distances are geodesic and are measured using the rectilinear metric, as discussed in the previous chapter.

Before we proceed, we provide a brief history of the 1-median problem. Given the 1-dimensional case and no obstacles, the 1-median can be computed in linear time [10]. The same problem in the plane (2-Dimensions) can be split into two 1-Dimensional problems and solved again in time linear to the number of source points [15]. Chepoi and Dragan [7] studied the problem of computing a median point in a simple rectilinear polygon and presented an $O(p + n \log p)$ algorithm to solve this problem (p is the number of vertices of the polygon, and n is the number of source points lying inside the polygon).

Larson and Li [23] were amongst the first to study facility location problems in the presence of obstacles. They present an algorithm to find the shortest rectilinear path between origin-destination points in the presence of obstacles. With the help of these results, Larson and Sadiq [22] studied the general p-median problem in a two-dimensional Euclidean space in the presence of barriers. Kusakari and Nishizeki [20] presented an algorithm to find all 1-median points. This algorithm is output sensitive and runs in $O((k + m) \log m)$ time, where k is the number of polygonal vertices of the region found and m is the number of axis-parallel rectangular obstacles in the plane. The number of source points n does not affect the running time of the algorithm since the authors have assumed the value of n to be a constant. The storage space required for their algorithm is $O(k+m)$. If it is assumed that n is not constant, then their algorithm requires $O((k+nm)\log nm + n^2m)$ time. Choi *et. al.* [8] also worked on this problem and have presented an $O(n(n+m)\log m)$ time algorithm to find the 1-median point. They explain how to compute the median set (the set of all 1-median points) in $O(n(n+m)\log m + k)$ time, where k is the complexity of the median set. The 1-median set is a collection of rectilinear polygons, where line segments and individual points are degenerate cases of polygons. For example, it is possible that all points on a given line segment are 1-median points for a given set of source points.

Other facility location problems have also been studied in the presence of obstacles. Moshe, Katz and Mitchell [4] present an $O(mn\log(m+n))$ time algorithm to construct a data structure of size $O(mn)$ that can report the farthest point to a query point in $O(\log(m+n))$ time, where n is the number of source points and m is the number of obstacles in the plane. For the same setup, they also study the 1-center problem, where the objective is to find a point in the plane that minimizes the maximum distance to the n points. They present an $O(mn\log(m+n))$ time algorithm to find the 1-center point.

In this chapter, we present an efficient algorithm for the 1-median problem in the presence of obstacles. For a given set of n source points and m rectangular, axis-parallel obstacles, our algorithm runs in $O(n(n+m))$ time, and improves the current best published result of $O(n(n+m)\log m)$ in [8]. We further improve this running time to $O(nm\log n)$, motivated by the fact that in most cases $n \gg m$. We also reduce the storage space requirement of $O(mn)$ for the algorithm in [8] to $O(m+n)$ (linear) space.

2.2 Proposed Algorithm

We denote the set of n source points by $S = \{s_1, s_2, \ldots, s_n\}$ and the set of m disjoint, axis-parallel rectangular obstacles in the plane by $\mathcal{O} = \{O_1, O_2, \ldots, O_m\}$. The obstacles and source points fall in a given rectangular region R in the 2-D plane. As discussed in section 1.2, we denote the length of the shortest obstacle-avoiding rectilinear distance between two points p and q in the free space by $d(p, q)$. We preprocess the given data in the following manner. We maintain four sorted lists for the set of obstacles. The first list stores the obstacles sorted by the left edges, the second stores the obstacles sorted by the right edges, and the third and fourth lists store obstacles sorted by the top and bottom edges respectively. Similarly, we also maintain sorted lists for the set of source points that store the points sorted by the x-coordinate and by the y-coordinate. .

Our 1-median algorithm starts by generating horizontal and vertical line decompositions of the given input data. The edges of each obstacle are extended until they hit another obstacle or the boundary of the region R. We draw a horizontal line and a vertical line through each source point $s \in S$ in a similar manner. The set of horizontal line segments created is denoted by H, and the set of vertical line segments created is denoted by V. We present a detailed description of how the horizontal and vertical line decompositions are computed in the next section.

Lemma 2.2.1. *[8] There exists an intersection point among the $O(m+n)^2$ intersection points created by the horizontal and vertical line segments in H and V respectively, which is the optimal location for the 1-median.*

According to lemma 2.2.1, the 1-median point lies on the intersection of the horizontal and vertical line segments in H and V respectively. By definition, the 1-median point is the one amongst these intersection points that has minimum total distance to the n source points in S. Let h_i and v_j denote the horizontal and vertical line segments respectively that intersect to create an intersection point p. The distance from

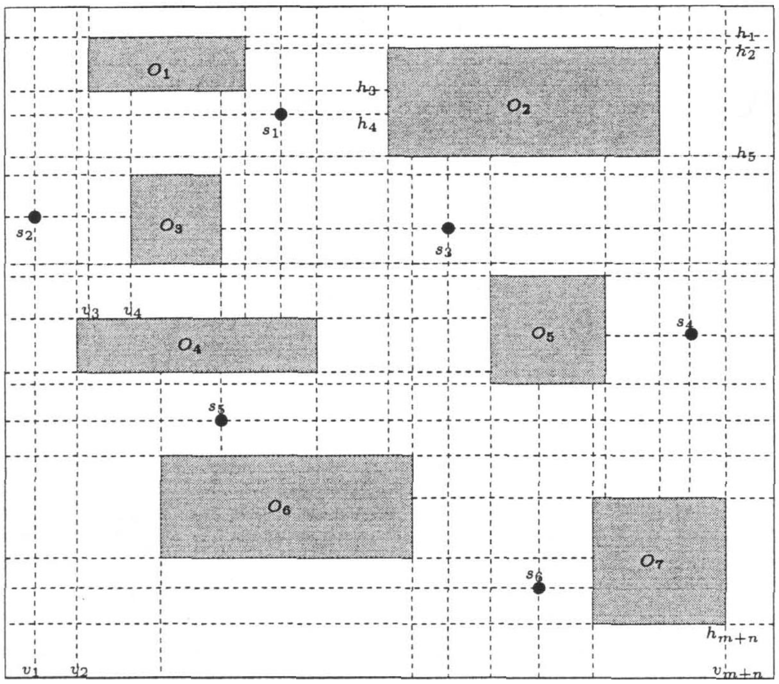

Figure 2.1: Preprocessing performed for the 1-median problem.

p to any source point s is the sum of the x-distance and y-distance from p to s. Thus the total distance from p to all the source points is the sum of all the x-distances and y-distances. Also, the total x-distance from the source points to p is the same as total x-distance to any other point on the vertical line segment v_j (lemma 1.2.2). Similarly, the total y-distance from the source points to p is the same as the total y-distance from the source points to any other point on the horizontal line segment h_i. Thus, knowing the total y-distances from the source points to all the horizontal line segments and the total x-distances from the source points to all the vertical line segments, we can find the total distance from any intersection point to all the source points in constant time.

The 1-median algorithm can be formally described as follows:

Algorithm 1-median
Input: A set of source points S, a set of obstacles $\mathcal{O}$.
Output: A 1-median point of the set S.

Step 1: Compute the horizontal line decomposition (HLD) and the vertical line decomposition (VLD) of the given input data.

Step 2: (i) Find the x-distances from the source points in S to the line segments in V.
(ii) Find the y-distances from the source points in S to the line segments in H.

Step 3: Find the intersection point, among all intersection points created by the line segments in H and V, that has minimum sum of x-distance and y-distance.

2.2.1 Finding Horizontal and Vertical Line Decompositions

We now present the technique used to obtain the HLD and the VLD. We use the sorted lists of obstacles and source points described in the beginning of this section. To compute the vertical line decomposition, we sweep a vertical line L that starts from the left edge of the region R and stops when it hits (i) the left edge of an obstacle, (ii) the right edge of an obstacle, or (iii) a source point. The sweep-line maintains a list of "active" obstacles, i.e., those whose left edge has been encountered but right edge has not yet been encountered. On encountering the left and right edges of an obstacle, we extend these edges in the +y and -y directions until they hit another obstacle or the boundary of the region R. When the line L encounters a source point, we shoot two rays out of the source point in +y and -y directions until the rays hit either an obstacle or the boundary of R. The obstacles lying directly above and directly below a source point or another obstacle can be determined using a list of active rectangles maintained with the sweep-line. A balanced binary tree (such as a red-black tree [10], AVL tree [10], splay tree [32]) can be used to maintain this active list in sorted order, and the VLD can be computed in $O((m+n)\log m)$ time. We obtain the horizontal line decomposition in a similar manner in $O((m+n)\log m)$ time by sweeping a horizontal line starting from the top edge of the region R. These procedures create $2m+n$ horizontal line segments and $2m+n$ vertical line segments in the region R. Let the set of horizontal line segments created be denoted by $H = \{h_1, h_2, \ldots, h_{2m+n}\}$ and let the set of vertical line segments created be denoted by $V = \{v_1, v_2, \ldots, v_{2m+n}\}$. See figure 2.1 for illustrations.

2.2.2 Finding Distances From Source Points to Line Segments in H and V

We now present Step 2 of our 1-median algorithm which again uses a sweep-line technique to calculate the total x-distances (y-distances) from the source points in S to each vertical line segment (horizontal line segment) in V (H). Our algorithm considers each source point $s \in S$ at a time and computes its x-distance (y-distance) contribution to all the vertical (horizontal) line segments created in Step 1 of the 1-median algorithm.

We explain how to calculate the y-distance from a source point s to all the horizontal line segments in H using a vertical sweep-line L. The same procedure can be applied to calculate the x-distances from s to the vertical line segments in V using a horizontal sweep-line. We denote by $ytotal(h)$ the total y-distance to the horizontal line segment $h \in H$ from all the source points processed so far by our algorithm.

Algorithm Find y-distances(currentp)
Input: Source Point *currentp*, set of source points S, set of obstacles $\mathcal{O}$, set of horizontal line segments H.
Output: y-distance contribution of *currentp* to all $h \in H$.

Step 1: Initialize a vertical sweep-line L passing through *currentp*.

Step 2: Sweep L in the +x direction, stopping at the two events mentioned below:

(a) /* *L passes through a source point s* */.
Compute the y-distance of the horizontal line segment h_s passing through s from the source point *currentp* and add this contribution to $ytotal(h_s)$.

(b) /* *L coincides with the left edge of an obstacle $o \in \mathcal{O}$* */.
Compute the y-distance of the horizontal line segments passing through the top and bottom edges (denoted by h_i and h_j respectively) of o from the source point *currentp* and add this y-distance contribution of *currentp* to $ytotal(h_i)$ and $ytotal(h_j)$.

Step 3: Sweep the line L (starting from *currentp*) in the -x direction, stopping at the two events mentioned below:

(a) /* *L passes through a source point s* */.
Compute the y-distance of the horizontal line segment h_s passing through s from the source point *currentp* and add this contribution to $ytotal(h_s)$.

(b) /* *L coincides with the right edge of an obstacle $o \in \mathcal{O}$* */.
Compute the y-distance of the horizontal line segments passing through the top and bottom edges (denoted by h_i and h_j respectively) of o from the source point *currentp* and add this y-distance contribution of *currentp* to $ytotal(h_i)$ and $ytotal(h_j)$.

Repeating the three steps described above for all the source points in S, we can calculate the total y-distance from all the source points to each of the horizontal line segments in H. We now present a detailed description of how the two events in Step 2 of the above algorithm are handled.

Event (a): The sweep-line hits a source point s during the forward sweep.
Let the vertical line passing through s be denoted by v_s and the horizontal line passing through s be denoted by h_s. h_s either intersects the vertical line passing through *currentp* or hits an obstacle that lies to the left of s (but to the right of *currentp*). In the first case, the y-distance from *currentp* to s is simply the $y-$constituent of the Manhattan distance between the two points, i.e.

$$d_y(currentp, s) = |currentp_y - s_y|$$

We know that the y-distance from the point *currentp* to s is the same as the y-distance from *currentp* to any point on the horizontal line segment passing through s, i.e. h_s. Thus,

$$ytotal(h_s) = ytotal(h_s) + d_y(currentp, s)$$

i.e., the contribution of *currentp* to h_s has been added to the total y-distance of h_s.

In the second case, suppose the line h_s hits an obstacle o. This obstacle has already been encountered by the sweep-line L. Thus the shortest distances from *currentp* to the corner vertices of o, i.e. $tl(o), tr(o), bl(o)$ and $br(o)$ have already been calculated. Suppose that h_s intersects the right edge of o at a point q (see figure 2.2). The shortest path from *currentp* to q passes through either $tr(o)$ or $br(o)$. Thus,

$$\begin{aligned} d_y(currentp, q) &= \mathbf{min}\{d_y(currentp, tr(o)) + d_y(tr(o), q), \\ &\quad d_y(currentp, br(o)) + d_y(br(o), q)\}. \end{aligned}$$

$$\begin{aligned} \therefore d_y(currentp, q) &= \mathbf{min}\{(d_y(currentp, tr(o)) + |tr(o)_y - q_y|), \\ &\quad (d_y(currentp, br(o)) + |br(o)_y - q_y|)\}. \end{aligned}$$

Since s and q lie on the same horizontal line h_s,

$$d_y(currentp, s) = d_y(currentp, q).$$

Thus,

$$ytotal(h_s) = ytotal(h_s) + d_y(currentp, s).$$

When computing the y-distance from a point p, the vertical sweep-line stops at $O(n)$ source points. Since the HLD has already been computed earlier, we can determine in constant time the end points of the horizontal line h passing through a source point s. If h intersects the vertical line passing through p then the y-distance between s and p can also be computed in constant time by taking the difference of the y-coordinate values of s and p. If the line h hits an obstacle o, we need to consider the y-distances stored at the right corners of o. This requires constant time also. Thus for each source point we require $O(n)$ time to determine its y-distances to other source points. Therefore, the running time of this event for all source points is $O(n^2)$.

Event (b): The sweep-line L hits the left edge of an obstacle o. Let h_o^1 and h_o^2 be the horizontal lines passing through the top and bottom edges of o respectively. Each of these horizontal lines either intersects the vertical line passing through *currentp* or hits an obstacle lying to the left of o (but to the right of *currentp*). Thus, the y-distances to the two horizontal lines h_o^1 and h_o^2 can be computed in the same way as explained for the horizontal line passing through a source point. We also record the y-distances to the corner vertices of o in the variables $y(tl(o))$, $y(tr(o))$, $y(bl(o))$ and $y(br(o))$ (which store the y-distance from p to the top-left, top-right, bottom-left and bottom-right corners of obstacle o respectively).

The running time for the case of obstacles is similar to that of source points, since obstacles are treated as two separate source points. There are $O(m)$ obstacles, and all distance computations are done in constant time. Thus for all source points the total time required is $O(nm)$.

To illustrate, consider figure 2.2. Obstacle O_1 creates two horizontal edges, namely $h_1(O_1)$ and $h_2(O_1)$. Both these edges intersect the vertical edge passing through *currentp*. Thus the y-distances to these two edges can be easily calculated by taking

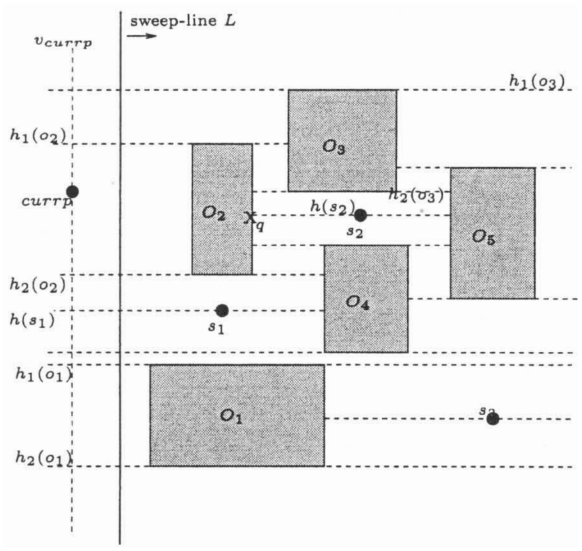

Figure 2.2: Illustration of the 1-median algorithm.

the absolute difference of the y-coordinates between each edge and *currentp*. The y-distance to the edge $h_1(O_1)$ is the same as the y-distance to the top corner points of O_1. The y-distances to the edges of O_2 can be similarly calculated. Now consider the source point s_2. The horizontal edge $h(s_2)$ hits the obstacle O_2 on the left at point q. We have already calculated the distances from *currentp* to the corners of this obstacle. The y-distance from *currentp* to q can now be easily calculated by considering the two paths, i.e. one that comes from the top-right corner of O_2 and the other that comes from the bottom-right of O_2 corner and choosing the one which minimizes the total distance. The distance from *currentp* to q is the same as the distance from *currentp* to the horizontal line $h(s_2)$ and also to the source point s_2. Suppose that the obstacles $O_1, \ldots, O_4$ have already been processed by the sweep-line, i.e., we know the distances from *currentp* to the corners of each of these obstacles. For the obstacle O_5, its two horizontal edges, $h_1(O_5)$ and $h_2(O_5)$ hit the obstacles O_3 and O_4 respectively. The distance to each of these edges can be computed as explained for the edge $h(s_2)$.

The method outlined above computes the shortest y-distances from the source points to the horizontal line segments that originate from the obstacles that lie to the right of the source point under consideration. The x-distances from the source points to the vertical line segments in V can be computed in a similar fashion using horizontal sweep lines.

Lemma 2.2.2. *Given a source point p, the y-distance from p to all line segments in H and the x-distance to all line segments in V can be computed in $O(m+n)$ (linear) time.*

2.2.3 Finding the 1-median point

We have calculated the total distances from the source points to the horizontal and vertical line segments in H and V respectively. As seen in fig. 2.3, the horizontal and vertical segments created intersect each other and according to lemma 2.2.1, the median point of the source points lies on one such intersection point. For any given intersection point p, the total distance from p to all the source points can be easily calculated by considering the total y-distance stored at the horizontal segment passing through p and the total x-distance stored at the vertical segment passing through p, i.e.

$$\sum_{s \in S} d(s,p) = ytotal(h_p) + xtotal(v_p)$$

where h_p and v_p are the horizontal and vertical segments passing through p respectively. Our goal is to find the intersection point that has the minimum total distance among all the intersection points.

A brute force algorithm to find all the intersections and then to find an intersection with the minimum total distance i.e. the 1-median point would require $O((n+m)^2)$ time. Instead, we use a sweep-line technique to solve this problem in $O((n+m)\log(n+m))$ time. We first sort the end points of the vertical line segments in V and the horizontal segments in H by their y-coordinates. Each vertical line segment in V has a total x-distance value associated with it and each horizontal line segment in H has a total y-distance value associated with it (as described in

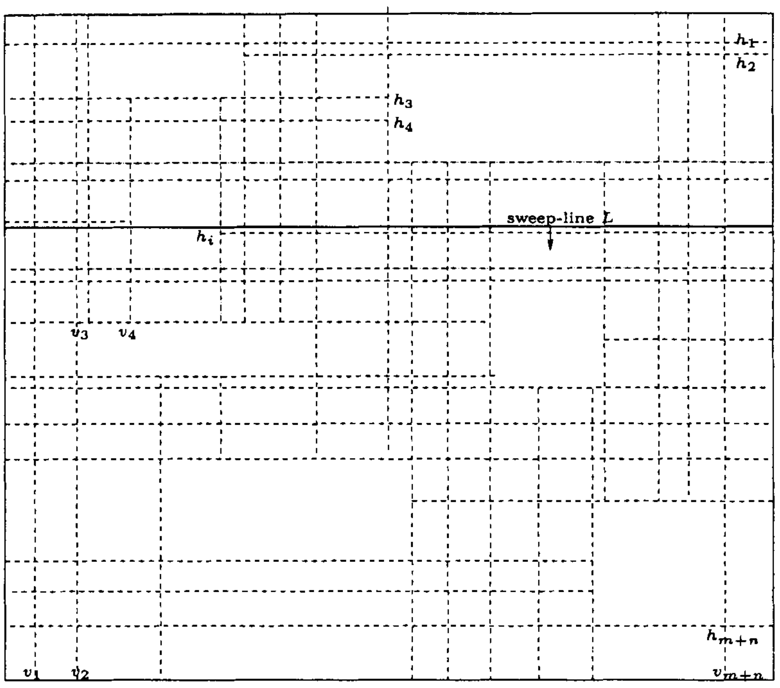

Figure 2.3: Intersection of segments in H and V.

the previous section). A horizontal line L sweeps the plane from top to bottom. As the sweep-line moves, we maintain the set of "active" vertical line segments that are currently intersecting L. Once a vertical line segment stops intersecting L it becomes non-active. The line L also stops when it coincides with a horizontal line segment $h \in H$. We choose the vertical line segment v from the active list that intersects h and has the minimum total x-distance. The intersection point of v and h is a candidate for the 1-median point. Such a candidate is created each time L stops at a horizontal line during the sweeping process and we pick the candidate with minimum total distance as the 1-median point of the set of source points S.

Algorithm: Locate 1-median point

Input: Set of horizontal line segments H, set of vertical line segments V.

Output: 1-median point of set of source points S.

Step 1: Start with *min_candidate* set to null, *min_candidate_value* set to ∞, and the horizontal sweep-line L set at y_∞.

```
Step 2: Generate the event queue $Q$ which contains the end points of line segments
in $V$, and the line segments in $H$, all sorted by the values of their y-coordinates in
decreasing order.

Step 2: $L$ stops at an event $e$ :
  (a) /* $e$ is the top of a vertical line segment $v \in V$ */
        Add $v$ to the set of active line segments denoted by $V_{active}$.

  (b) /* $e$ is the bottom of a vertical line segment $v \in V$ */
        Delete $v$ from the set of active line segments $V_{active}$.

  (c) /* $e$ is a horizontal line segment $h \in H$ */
        (i) Determine line segments in $V_{active}$ intersecting with $h$.
        (ii)Pick up the segment $v \in V_{active}$ that has minimum value of total
        x-distance.
              ($p$ is the intersection point of $h$ and $v$).
        (iii)If y-distance(h) + x-distance(v) $\leq$ min_candidate_value then
              min_candidate = $p$
              min_candidate_value = y-distance(h) + x-distance(v).

Step 3: Output median_point = min_candidate
               median_distance = min_candidate_value.
```

Implementation:

We maintain information for the sweep-line procedure mentioned above using a balanced binary tree denoted by T. The leaf nodes of this tree store the active vertical line segments (in increasing order of x-coordinate value) currently intersecting L. Each leaf node contains a record which contains two pieces of information, (i) the x-coordinate of the vertical line segment represented by that node, and (ii) the total x-distance associated with the vertical line segment. An internal node i stores a record made up of the splitting value (to guide the search) and a link to the vertical segment that has the minimum total x-distance among all the children of i. Each node also stores a link to its parent node.

When the line L hits the top of a vertical line segment, a new leaf node z is created and added at the appropriate location in the tree. The path from the node z to the root of T is updated to reflect changes (if any) caused by the addition of z (for example, z could represent a vertical line segment having minimum total x-distance) . Similarly, when L hits the bottom of a vertical line segment, the node corresponding to that line is deleted from the tree T and the tree is updated. When the line L and a horizontal line segment $h \in H$ coincide, we perform a 1-dimensional range query on the tree T using the x-coordinates (denoted by h_{x1} and h_{x2}) of the endpoints of h to determine the vertical line segments intersecting h. The query on h_{x1} gives a path p_1 from the root of the tree T to a leaf node l_1 and the query on h_{x2} gives a path p_2 from the root of the tree T to a leaf node l_2. All the leaf nodes stored between l_1 and l_2 represent the active vertical line segments intersecting h. To find the vertical line segment v_{min} with the minimum total x-distance, we simply traverse the paths from l_1 and l_2 towards the root of T. Let the internal node where these two paths meet be denoted by i. Each node hanging off the paths between i and l_1 and i and l_2 (and lying within the region defined by these two paths) stores a pointer to the vertical line segment having minimum total x-distance among all of its children. We traverse from i to l_1 and l_2, examining each such internal node and picking the one which points to the leaf node whose vertical line has minimum x-distance. Denote this vertical line segment by v_{min}. The intersection of v_{min} and h is a candidate point for the median point. Figure 2.4 illustrates the search for the 1-median point using balanced binary trees. The internal nodes hanging off the paths are shaded in grey.

The binary tree created above stores information about the active vertical line segments during the sweeping process. There are $O(m+n)$ vertical line segments and thus the size of this binary tree is $O(m+n)$ and the depth is $O(\log(m+n))$. The sweep-line stops at three events, the two end points of a vertical line segments and when it coincides with a horizontal line. Thus there are a total of $O(m+n)$ events. At the events corresponding to a vertical line segment, we either add or remove the line from the balanced binary tree, and each such operation requires $O(\log(n+m))$ time. The vertical line segment with the minimum total x-distance can be found in $O(\log(m+n))$ time. For the events corresponding to the horizontal line segments, we perform two queries on the BST, each also requiring $O(\log(n+m))$ time. Once the candidates have been generated for each horizontal line, we can find the one with the minimum total distance in $O(n+m)$ time.

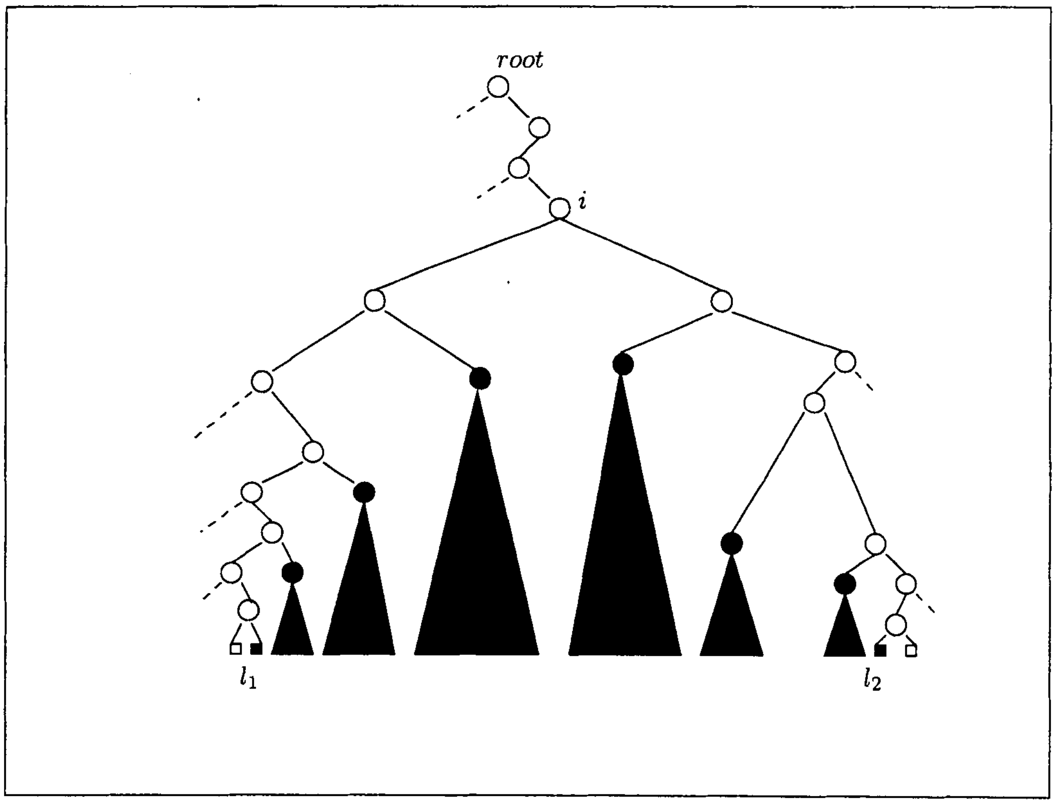

Figure 2.4: Searching for the 1-median point using balanced binary trees

2.3 Analysis

In the previous section we described an algorithm to find the 1-median point of a set of n source points in the presence of m rectangular axis-parallel obstacles. We conclude by presenting an analysis of the running time of our algorithm.

We first preprocess the given data, which involves the sorting of the obstacles and source points, and the creation of the horizontal and vertical line segments (the sets H and V). Since there are n source points and m obstacles, sorting operations can be performed $O((n + m)\log(n + m))$ time. In section 2.2.1, we have seen that the HLD and VLD can be computed in $O((n + m)\log m)$ time. Section 2.2.2 shows how the distances from source points to the line segments in H and V can be computed in $O(n(n+m))$ time. Finally, we have also seen in section 2.2.3 how to find the 1-median point in $O((n + m)\log(n + m))$ time.

Thus, the running time of our algorithm is

$$O(\underbrace{(n + m)\log(n + m)}_{Preprocessing\ time} + \underbrace{n(n + m)}_{Finding\, x-distances\, and\, y-distances} + \underbrace{(n + m)\log(n + m)}_{Finding\ 1-median\ point})$$

The expression above is dominated by the time required to find the x-distances and y-distances to the vertical and horizontal line segments respectively and hence the running time is $O(n(n + m))$.

In terms of storage space, the algorithm presented by Choi *et. al.* in [8] requires a total of $O(mn)$ space. During the process of locating the 1-median point, the authors create a height-balanced binary search tree for each obstacle. Each tree stores information about the cuts created by the source points and other obstacles ("cuts" are equivalent to the line decompositions in the approach presented here). This requires a total of $O(mn)$ storage space. In contrast, our algorithm requires only linear space. Only $O(m + n)$ horizontal and vertical line segments are created by the HLD and VLD processes. Locating the 1-median point requires construction of a balanced binary tree which has linear storage requirement. Thus the storage space requirement of the proposed algorithm is $O(m+n)$.

2.4 An Improved Algorithm

In many real-life applications, it is often the case that the number of source points is much greater than the number of obstacles that lie in the plane, i.e., $n \gg m$. For example, when a facility or service (such as a post office or police station) needs to be placed in a neighbourhood, the number of customers (houses or shops) far outnumber the number of obstacles lying in that neighbourhood. In cases where the relative orders of n and m exhibit such a relationship, we improve upon the running time of the previously mentioned 1-median algorithm by a factor of $O(\frac{n}{m \log n})$.

The basic idea remains the same, i.e., the intersections of the vertical and horizontal line segments in H and V are the candidates for the 1-median point. In the previous algorithm, we considered each source point at a time and computed its x-distance(y-distance) contribution to all the elements in $V(H)$. Instead, we now consider a group of points to calculate these contributions during the sweeping process. Consider the VLD shown in figure 2.5. The vertical line segments in V created by the left and right edges of obstacles partition the region into a number of rectangles. This set of rectangles created by the subset of line segments in V is denoted by $rect_V$. Figure 2.5 depicts the partitioning of the given region into rectangles. Each source point in S falls in one of these rectangles and each rectangle contains zero or more source points. Let the number of source points in r_i be denoted by n_i. We process rectangles one at a time and calculate the x-distance contribution of all the points contained inside to all the vertical line segments in V. Similarly, the HLD decomposes the input region R into another set of rectangles (we denote this set by $rect_H$) , and the rectangles are processed one at a time to calculate the y-distance contribution of all the points contained inside it to all the horizontal line segments in H. Locating the 1-median point from the candidates is then performed in the same manner as explained in section 2.2.3 .

Algorithm: 1-median_Improved
Input: Set of source points S, set of obstacles $\mathcal{O}$.
Output: 1-median point of set of source points S.

Step 1: Find HLD and VLD of the given input.

Step 2: Determine the points lying in each rectangle.

Step 3: For each rectangle $r \in rect_V$, compute the x-distance contribution of all the source points in r to the line segments in V.

Step 4: For each rectangle $r \in rect_H$, compute the y-distance contribution of all the source points in r to the line segments in H.

Step 5: Locate the 1-median point among all intersections of line segments in H and V.

In order to determine which rectangles the source points belong to, we can use a sweep-line method similar to the one described in 2.2.1. The sweep-line maintains information about the currently active obstacles. Each source point belongs to exactly one rectangle, and this rectangle is bounded from above and below by the edges of specific obstacles. When the sweep-line hits a source point, the point can be projected above and below to determine the two obstacles it falls on. Knowing the two obstacles, we can determine which rectangle the point belongs to. This entire step can be executed in $O((m+n)\log m)$ time.

2.4.1 Steps 3 and 4 : Computing Distances To Line Segments in H and V

Step 1 and Step 5 of the above algorithm are performed in exactly the same manner as in the algorithm described in section 2.2 to find the 1-median point. Step 2

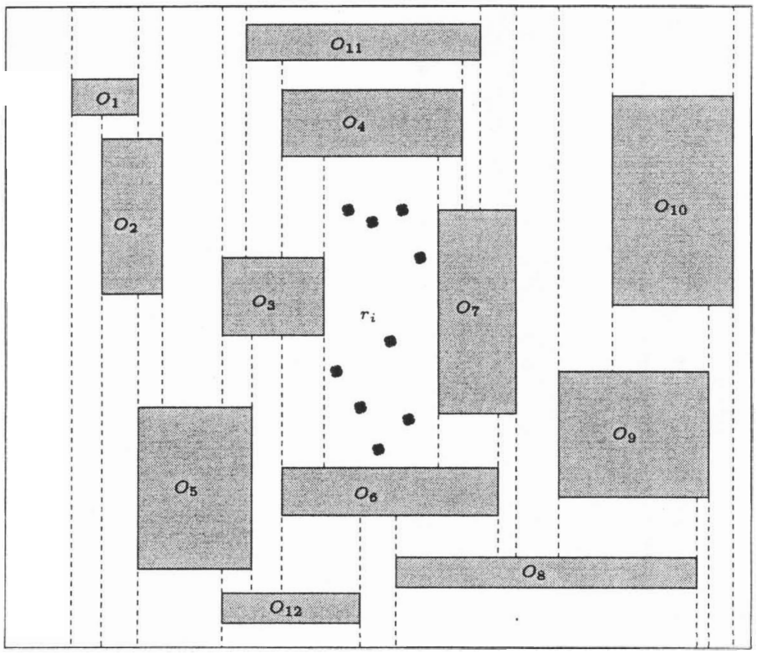

Figure 2.5: Division of the input region into rectangles

is performed as explained in the previous paragraph. We therefore focus on Step 3 and Step 4 which calculate the distance contributions of the source points to the line segments in H and V. We describe the processing of the rectangles created by the VLD (i.e., calculation of x-distances to line segments in V). Rectangles created by HLD are processed in a similar manner.

Let the set of rectangles created by the VLD (consider only vertical edges created by the obstacles) be denoted by $rect_V = \{r_1, r_2, \ldots, r_l\}$ (where l is $O(m)$). We explain the procedure for one rectangle, say r_i, and all the other rectangles are processed in the same fashion. Our objective is to compute the x-distance contribution of all the n_i source points in r_i to all the vertical line segments in V. We project all the source points in r_i onto a horizontal line segment AB, whose end points A and B lie on the left and right edges of r_i respectively. Denote the set of projected points by $S_i =< s_1, s_2, ..., s_{n_i} >$. The set S_i is sorted by the x-coordinate values of its elements. Note that moving these points in the y-direction within the rectangle does not change their x-distance to any object (obstacle or source point). For each projected source point $s_j \in S_i$, we store two values: (i) the sum of the x-distances from each projected source point to the left of s_j (including s_j) to the point A (denoted by $d_A(j)$), and (ii) the sum of the x-distances from each projected source point lying to the right of s_j (including s_j) to the point B (denoted by $d_B(j)$). Thus,

$$d_A(j) = \sum_{z=1}^{j} d_x(s_z, A)$$

$$d_B(j) = \sum_{z=j}^{n_i} d_x(s_z, B)$$

Thus, $d_A(n_i)$ denotes the sum of the x-distances of all the source points in S_i to A and the value is stored with the projected source point s_{n_i}. Similarly, $d_B(1)$ denotes the sum of the x-distances of all the source points in S_i to B. Clearly, this can be computed in $O(n_i)$ time.

Next, we compute the shortest distance from the point A to each vertical line segment $v \in V$ and store this value with v. Similarly, we also store at v the shortest distance from B to v. We are now ready to compute the x-distance contributions of the source points in S_i to the vertical line segments in V. For each vertical line segment $v \in V$ we know its x-distance to the points A and B, denoted by $d_x(A, v)$ and $d_x(B, v)$ respectively. We also know the length of the line segment AB. Compute the point s_{mid} on the line segment AB such that the x-distance from s_{mid} to v via A is equal to the x-distance from s_{mid} to v via B, i.e.,

$$d_x(s_{mid}, A) + d_x(A, v) = d_x(s_{mid}, B) + d_x(B, v)$$

It is easy to see that all projected source points in S_i on the left of s_{mid} have shortest distance to v through A, and all projected source points in S_i on the right of s_{mid} have shortest distance to v through B. It may happen that s_{mid} lies to the left of s_1, in which case all source points have shortest x-distance to v through B, or s_{mid} lies to the right of s_{n_i} in which case all source points have shortest path to v through A. Let the first projected source point to the immediate left of s_{mid} be s_j and the first projected right point to the immediate right of s_{mid} be s_{j+1}. Note that if s_{mid} coincides with a projected source point in S_i then the length of the shortest path from that source point to the vertical line segment v via A is the same as that of the shortest path via B. In this case, we assume that the shortest path goes via A and let s_j be the source point that coincides with s_{mid}. The point s_{mid} can be found in $O(\log n_i)$ time by performing a binary search on the set S_i. The x-distance contribution of the source points lying in r_i to the vertical line segments in V can now be computed as follows:

$$\sum_{s \in S_i} d_x(s,v) = \big(j * d(A,v)\big) + d_A(j) + \big((n_i - j) * d(B,v)\big) + d_B(j+1)$$

This x-distance contribution of the source points in r_i to v can be added to the total x-distance contribution from all source points in all rectangles.

$$xtotal(v) = xtotal(v) + \sum_{s \in S_i} d(s,v)$$

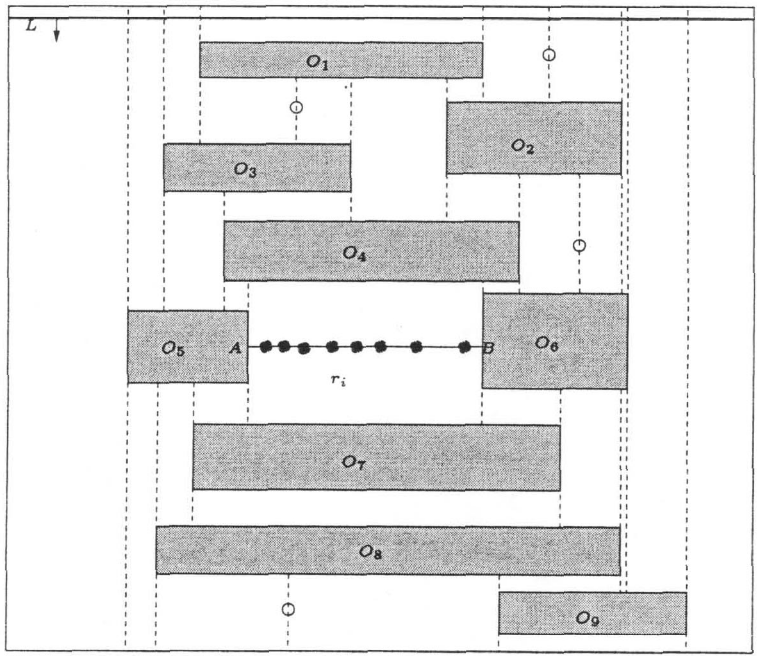

Figure 2.6: Processing rectangle r_i for improved 1-median problem

Each source point of S_i also makes x-distance contributions to the vertical line segments created by all other source points of S_i. Since we have already computed the values $d_A(j)$ and $d_B(j)$, $j = 1, \ldots, n_i$, we can compute in constant time the x-distance contributions of all source points in S_i to the vertical line segment of a given source point of S_i. Consider a projected source point $s_j \in S_i$. To compute the x-distance contributions from all projected source points lying to the right and left of s_j to the vertical line segment v_j created by s_j, we use the expression

$$\begin{aligned}\sum_{s \in S_i} d_x(s, v_j) &= d_B(1) - d_B(j) - \big((j-1) * d_x(s_j, B)\big) \\ &\quad + d_A(n_i) - d_A(j) - \big((n_i - j) * d_x(s_j, A)\big).\end{aligned}$$

For the rectangles created by the HLD, the same process can be applied to calculate the y-distance contributions of source points to the horizontal line segments in H. The intersections of the line segments in H and V are the candidates for the 1-median point, and this point can be located using the balanced binay tree procedure used in the previous 1-median algorithm (see section 2.2.3).

2.4.2 Analysis

In this section we present an analysis of the running time of the improved 1-median algorithm presented above. The steps to generate the HLD and the VLD remain the same, thus there is no change in the running time for these steps. Also, once the distances have been calculated, the location of the 1-median point is also performed in the same manner as in the previous 1-median algorithm. Thus, the only difference lies in the calculation of the distances to the line segments in H and V. This step dominated the running time of the 1-median algorithm described previously.

The m obstacles create $O(m)$ line segments in V that divide the region R into $O(m)$ rectangles. Considering each rectangle r_i at a time, we first determine (in constant time) two points A and B on the left and right edges of r_i. There are n_i source points of S that lie inside the rectangle r_i. For each vertical line segment in V, we find its shortest x-distance from the two points A and B. Section 2.2.2 explains how we can calculate the x-distance from a point to all vertical line segments in $O(m+n)$ time. We then calculate the values $d_A(j)$ and $d_B(j)$ ($j = 1, \ldots, n_i$), which can easily be done in time linear to the number of points, i.e. $O(n_i)$. For each vertical line segment, we then find the two consecutive points in S_i between which the point s_{mid} lies in $O(\log n_i)$ time. Thus, the entire process of calculating the x-distance contributions to the vertical line segments for one rectangle takes $O((m+n)+n_i+(m+n)\log n_i)$ time. The y-distance contributions to the horizontal line segments for a rectangle can be calculated similarly with the same complexity. Hence for $O(m)$ rectangles the running time is

$$O(m(m+n)+m\{\sum_{\forall r_i \in rect_V} n_i\}+m(m+n)\sum_{\forall r_i \in rect_V} \log n_i)$$
$$= O(m^2\log n + mn\log n)$$

Since $n \gg m$ the running time of our improved 1-median algorithm is $O(mn\log n)$.

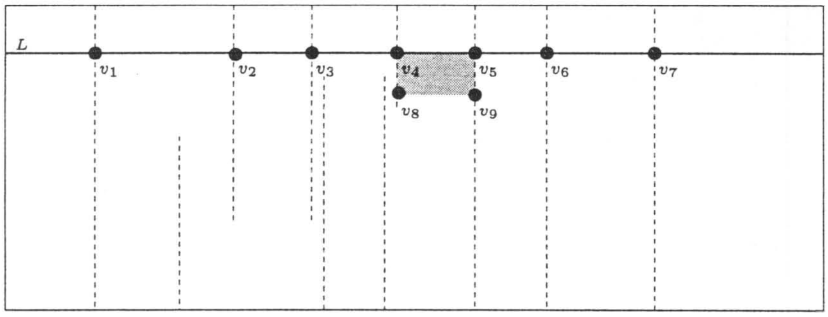

Figure 2.7: Computing the set of 1-medians

2.4.3 Computing the 1-median set

The 1-median set is defined as a collection of rectilinear polygons such that all the points lying inside these polygons are 1-median points of the given source points. Line segments and individual points are considered as degenerate cases of the polygons. Our 1-median algorithm described above can be used to determine the 1-median set of the source points in S. Consider figure 2.7, the horizontal line L intersects the active vertical segments at $v_1, v_2, .., v_7$. Suppose our 1-median algorithm reports that the points v_3, v_4 and v_5 have the minimum sum of x and y distances (i.e. these points are 1-median points), then the line segment (v_3, v_5) is an element of the median set and any point lying on this segment is a median point. This information can be gathered during the sweep process used to determine the intersection point with minimum total sum of distances. The rectangular region enclosed by the points (v_4, v_5, v_8, v_9) is also part of the median set if v_8 and v_9 are also median points. The complexity of the 1-median algorithm to find the 1-median set is $O(mn \log n + k)$, where k is the number of elements found in the 1-median set.

2.5 Conclusion

In this chapter we presented two algorithms that solve the 1-median problem for a set of source points in the presence of obstacles based on the relative orders of n(number of source points) and m(number of obstacles). Distances were calculated using the rectilinear L_1 metric. The first algorithm presented has a running time of $O(n(n+m))$, which improves the current best published time of $O(n(n+m)\log n)$ presented in [8]. If we know that the number of source points is significantly higher than the number of obstacles (which is commonly the case in many facility location applications), then we can further improve the algorithm to run in $O(mn \log n)$ time. The storage space requirement for our algorithm is linear, i.e. $O(m+n)$ as compared to the $O(mn)$ storage space used by the algorithm presented by Choi *et. al.* [8]. We believe that the methods described in this chapter can also be used to improve other similar facility location problems, such as the farthest neighbor problem as described in [4]. We also use these methods in the subsequent chapters to solve the shop-floor layout problem and the constrained clustering problem.

Chapter 3

The Shop Floor Layout Problem

3.1 Background and Problem Definition

Facility location plays an important role in the development of transportation networks where goods need to be transported between various sites. Typically, the locations of the demand facilities are known, and the objective is to determine the optimal location of one or more supply facilities that provide services to the demand facilities. This "optimally" placed supply facility minimizes the overall transportation cost of the network. In [15], the authors tackle such facility location problems, but assume that the facilities are points in the network and hence are not obstacles hindering travel. For many such applications, it is the case that the supply and demand facilities are considerably large in size. Consider a factory, where the raw materials are stored in a huge warehouse, and there are several independent production units that periodically require the raw materials to be delivered to them from the warehouse. Warehouses and production units usually occupy large space. The regions occupied by these facilities pose a barrier to travel and the distances have to be measured differently from the case where the facilities were points in the network.

The shop floor layout problem was introduced and studied by Wang *et. al.* in [35]. The layout is defined as a rectangular region in the plane. We are given several demand facilities which lie within the boundaries of a layout. $D = \{d_1, d_2 \ldots, d_n\}$ denotes the set of demand facilities. Also given is a supply facility S that can lie anywhere in the layout, except in the interior of a demand facility. The demand facilities are a barrier to travel. Distance measurements and travel paths are calculated as described in section 1.2. The objective is to find the best placement for the supply facility in the layout, and one optimal I/O point on each of the facilities (both supply and demand). The I/O points can be thought of as doors on the facilities through which the exchange of goods takes place.

Given the above set-up, we need to find a solution that minimizes the total transportation cost associated with the layout. Each rectangular demand facility d_i is a 4-tuple $(x_i, y_i, u_i, v_i), i = 1, 2, \ldots, n$, where u_i and v_i are the width and height of d_i respectively.

The top-left ($tl(d_i)$), top-right ($tr(d_i)$), bottom-left ($bl(d_i)$) and bottom-right ($br(d_i)$) vertices of d_i are $(x_i, y_i), (x_i+u_i, y_i), (x_i, y_i+v_i), (x_i+u_i, y_i+v_i)$. Similarly, let the four vertices of the supply facility S be $(x_0, y_0), (x_0+u_0, y_0), (x_0, y_0+v_0), (x_0+u_0, y_0+v_0)$. The goal is to find a point p_0 on supply facility S, and based on the location of p_0, we need to find a point p_i on each demand facility d_i such that $\sum_{i=1}^{n} d(p_0, d_i)$ is minimized for all possible $p_0 \in S$, where $d(p_0, d_i)$ is the length of the shortest rectilinear path from p_0 to demand facility d_i. Point p_i on the perimeter of d_i is the point that minimizes the distance from p_0 to that demand facility. The solution set $\{p_0, p_1, \ldots, p_n\}$ that satisfies the above function is the "optimal" solution and the I/O points are said to be optimally located.

Savas *et. al.* [31] study a similar problem, but assume that the I/O points or doors of the demand facilities are fixed and known as part of the input. The prescribed locations of the I/O points may not be optimal, and can result in extra transportation costs. Wang *et. al.* [35] addressed this issue in the shop-floor layout problem and assumed that the locations of the I/O points are unknown. They provide algorithms for several versions of this problem. In the first version, the supply facility is a fixed point in the layout. To obtain a solution to the problem one has to find the optimal I/O points on each of the demand facilities. The second version is an extension of the first, in which the supply facility is a point but its location is unknown. Thus the optimal location of the supply facility needs to be computed before the I/O points on the demand facilities can be determined. In the third version the optimal location of the supply point is restricted within a given convex rectilinear region.

In this chapter, we study the three versions of the shop-floor layout problem as described above. For each of the versions we present new algorithms that significantly improve the running times in [35].

3.2 Supply Facility as a Fixed Point

In the first version of the shop floor layout problem, the location of the supply facility is given as an input to the problem, and the goal is to find one I/O point p_i on the perimeter of each demand facility $d_i \in D$ such that $\sum_{i=1}^{n} d(p_0, p_i)$ is minimized. $p_0 = (x_0, y_0)$ is the given location of the supply point and $p_i = (x_i, y_i)$, $i = 1, \ldots, n$, is the location of the I/O point that lies on $B(d_i)$ (the perimeter of demand facility d_i, as described in section 1.2).

In [35], the authors present an algorithm that creates a network of nodes that are L_1 visible. Two points A and B are said to be L_1 visible if there exists an xy-monotone path between them (for definition of xy-monotone path refer to section 1.2). The nodes in this network represent the given supply point and the candidate I/O points for the demand facilities. The time required to generate this network is $O(n^2)$, since there are $O(n)$ nodes in the network and each pair has to be checked for L_1 visibility. The authors then apply Dijkstra's algorithm on the network to find a solution to the problem by generating the shortest paths between the nodes representing the source point and the candidate I/O points. This algorithm requires $O(n^4)$ total running time, where n in the number of demand facilities (rectangles) in the layout.

In the remainder of this section, we present a simpler and elegant algorithm that solves the shop floor layout problem with the supply facility as a fixed point in $O(n)$ time after a preprocessing of the input, a step that requires $O(n \log n)$ time. The algorithm starts by computing the horizontal and vertical line decompositions of the layout. A detailed explanation of how the decompositions are computed is provided in section 2.2.1. This generates a set of horizontal line segments H, and a set of vertical line segments V. We then generate candidate I/O points for each of the demand facilities in the following manner. Consider each line segment $h \in H$ and $v \in V$. h and v have end points on the borders of demand facilities or on the border of the layout. If such an end point falls on the border of a demand facility d_i, then the location of the end point is a candidate I/O point for facility d_i and is added to its list of candidates. The corner vertices of d_i are also added to its candidate list. Note that each intersection point of the horizontal line segments in H and the vertical line segments in V lying on a demand facility is a candidate I/O point for that demand facility. Figure 3.1 shows a shop floor layout with eight demand facilities and a fixed supply point p_0. It also depicts the HLD and VLD of the layout along with the candidate I/O points of the demand facilities (marked by 'x').

Lemma 3.2.1. *[35] The above technique generates all possible candidate I/O points which are optimally located.*

The proof for the above lemma can be found in the paper by Wang *et. al.* [35].

Lemma 3.2.2. *Given n demand facilities, the total number of candidate I/O points generated is always $O(n)$.*

Proof: The HLD and the VLD of the layout are computed by extending the four edges of each demand facility, and by drawing a horizontal and a vertical line through p_0. Both end points of each such extended line terminate either at the layout border or on a demand facility. Thus, each extended line creates at the most two candidate I/O points on some demand facilities. Each demand facility has four extended edges (two horizontal and two vertical), and can create at the most eight candidate I/O points on other demand facilities. Similarly, the supply point creates at the most four candidate I/O points. The corners of each demand facility are also considered as candidate points. Thus the maximum number of candidate I/O points is $8n + 4n + 4 = 12n + 4$.

Once the candidate I/O points are known, our algorithm computes the y-distance from the supply point p_0 to each line segment $h \in H$, and the x-distance from p_0 to each vertical line segment $v \in V$. Section 2.2.2 provides details about the computation of distances from a source point to lines in H and V. Consider a candidate I/O point p_i for demand facility d_i. As mentioned earlier, this candidate is an intersection point of some $h \in H$ and some $v \in V$. The total distance from p_0 to p_i is the sum of the y-distance and x-distance from p_0 to h and v respectively. In this manner, we compute the distance from p_0 to each candidate I/O point of d_i and choose the one that has shortest distance. This procedure is repeated for all demand facilities.

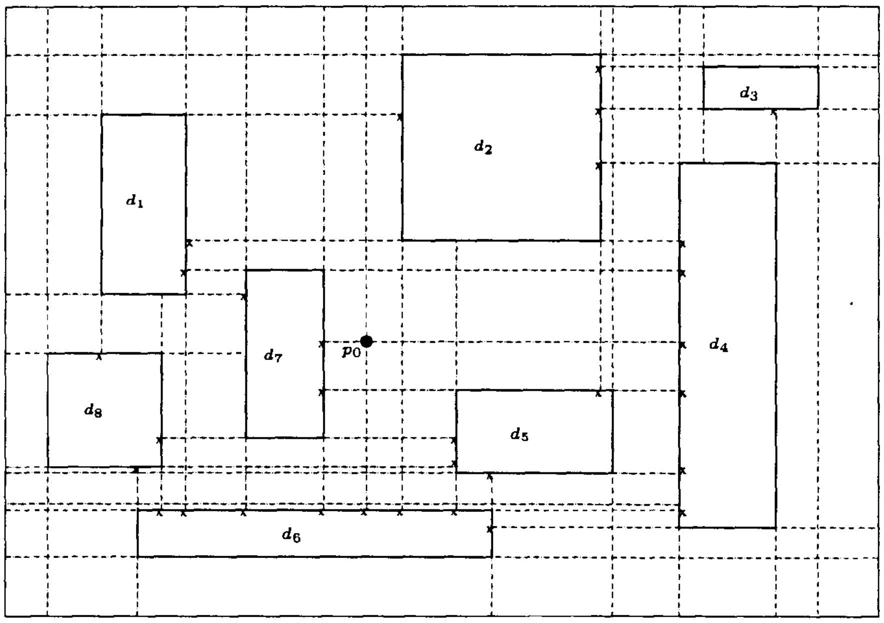

Figure 3.1: Shop Floor Layout Problem with fixed supply point and demand facilities.

Algorithm: SFL-FixedSupplyPoint
Input: Layout: Supply point p_0, set of demand facilities D.
Output: Optimal I/O points of demand facilities.

```
Step 1: Compute horizontal line decomposition (HLD) of the given layout.
        Compute vertical line decomposition (VLD) of the given layout.

Step 2: /* Determining the candidate I/O points for the demand facilities in D */.

        (i) For each line segment $l$ in $H \cup V$:
```

```
        {Let $e_1$ and $e_2$ be end points of $l$}.
        If $e_j$ (j=1,2) lies on the border of a demand facility $d_i$, add $e_j$
        to list $C_i$ of candidate I/O points of $d_i$.
    (ii) For each demand facility $d_i$ add its corner vertices to its list $C_i$ of
    candidate I/O points.

Step 3: For each $h \in H$, compute y-distance from $p_0$ to $h$ and for each $v \in V$,
    compute x-distance from $p_0$ to $v$.

Step 4: /* Find optimal I/O point for each demand facility $d_i$ */.

    (i) For each candidate in $C_i$ determine its distance from $p_0$.
    (ii) Select the candidate point, say $p_i$, with the minimum total distance.
    (iii) Set $p_i$ to be the optimal I/O point of $d_i$ with respect to $p_0$.
```

3.2.1 Analysis of Algorithm

In this section, we analyze our algorithm which finds the I/O points of the demand facilities given a fixed supply point p_0. The first step of this algorithm computes the HLD and the VLD of the given layout. We use the procedure described in section 2.2.1 to accomplish this. The running time associated with this step is $O(n \log n)$. In Step 2, in order to determine the candidate I/O points for the demand facilities, we examine the end points of each line segment in $H \cup V$. Step 3 computes the x-distances from the point p_0 to the elements of V, and the y-distances from the point p_0 to the elements of H. Section 2.2.2 explains how this can be performed in time linear to the number of elements in H and V, i.e $O(n)$ time. In Step 4, we examine the candidate I/O points of each demand facility and determine the optimal candidate. Since the number of candidates is $O(n)$ over all demand facilities (Lemma 3.2.2), this step takes linear time.

Thus, the running time of the shop floor layout problem with a fixed supply point is dominated by the time required to compute the HLD and the VLD of the given layout, and is $O(n \log n)$, where n is the number of demand facilities in the layout. This is a significant improvement over the $O(n^4)$ algorithm of Wang *et. al.* [35].

Theorem 3.2.1. *The shop floor layout problem where the supply facility is a fixed point can be solved in $O(n \log n)$ time.*

3.3 Location of Supply Point is Unknown

So far we assumed that the location of the supply point p_0 is fixed and is part of the input to the problem. This section examines a variation of the shop floor layout problem where the location of the supply point is unknown. Once the optimal location of the supply point in the layout is determined, the I/O points of the demand facilities can be computed as described in the previous section. We present an algorithm that solves this problem and significantly improves the running time of the algorithm described by Wang *et. al.* in [35].

In the previous chapter, we studied the 1-median problem where the objective was to find a 1-median point in the region such that the sum of distances from the point to a set of source points is minimized.

Lemma 3.3.1. *The solution to the above described shop floor layout problem where the supply point is to be placed optimally and the I/O points of the demand facilities are to be determined is equivalent to the solution of the 1-median problem where the I/O points are the source points and the supply point is the corresponding 1-median.*

The 1-median point is the location that has minimum total distance to all the source points in the region. The solution to the above shop floor layout problem is to locate an "optimal" point for the supply facility and then determine the I/O points on the demand facilities such that the sum of distances from the supply point to the I/O points is minimized. The supply point is therefore the 1-median point of the I/O points.

Lemma 3.3.2. *The y-distance from a horizontal line h_1 to another horizontal line h_2 can be determined by computing the y-distance from any point p on h_1 to the line h_2. The x-distance between two vertical lines can be computed similarly.*

This lemma is a simple extension of lemma 1.2.2. Let p and q be any two points lying on h_1, and r and s be any two points lying on h_2. By lemma 1.2.2,

$$d_y(s,p) = d_y(s,q) \tag{3.1}$$
$$d_y(r,p) = d_y(r,q) \tag{3.2}$$

Also,

$$d_y(p,r) = d_y(p,s) \tag{3.3}$$
$$d_y(q,r) = d_y(q,s) \tag{3.4}$$

It follows from (3.1) and (3.4) that

$$d_y(r,q) = d_y(p,s)$$

We now discuss our algorithm to solve the shop floor layout problem when the location of the supply facility is unknown. Given a layout consisting of n demand facilities, the algorithm starts by computing the horizontal line decomposition and the vertical line decomposition. This procedure creates two sets of line segments H and V. Lemma 3.3.1 implies that the intersection points of lines in H and V are the candidates for optimal placement of the supply point.

Next we find all the candidate I/O points of the demand facilities. These are the end-points of the lines in H and V, and the corners of the demand facilities. We then consider each line $h \in H$, and find the sum of its shortest y-distance to each demand facility $d \in D$. In order to know the shortest y-distance from h to a demand facility d, we need to know the y-distance from h to each of the candidate I/O points on d (only those caused by horizontal line segments in H). This is the same as knowing the y-distance from h to the horizontal lines causing candidate I/O points on d. In section 2.2.2 we examined how y-distances from a point are computed to the horizontal line segments in H. Using the same algorithm, we find the shortest y-distance from h to each demand facility d. Similarly, for each $v \in V$ we also find the sum of its shortest x-distance to each of the demand facility $d \in D$.

Among all the intersection points created by the line segments in H and V, we are interested in determining the intersection point p of two lines h and v that has minimum sum of x-distance at v and y-distance at h. This is the optimal location for the supply point p_0. We use the sweep-line method described in section 2.2.3 that maintains a balanced binary tree to determine the point p_0. Once p_0 is determined, we find the I/O points of the demand facilities using the algorithm described in the previous section for a fixed supply point.

Algorithm: SFL-SupplyPointUnknown
Input: Layout: Set of demand facilities D.
Output: Location of optimal supply point p_0, optimal I/O points of demand facilities.

Step 1: (i) Compute horizontal line decomposition (HLD) of the layout.
(ii) Compute vertical line decomposition (VLD) of the layout.

Step 2: Determine the candidate I/O points of the demand facilities.

Step 3: (i)For each $h \in H$ find the sum of shortest y-distance to each demand facility $d \in D$.

(ii) For each $v \in V$ find the sum of shortest x-distance to each demand facility $d \in D$.

Step 4: Find the optimal location of supply point p_0.

Step 5: Use algorithm **SFL-FixedSupplyPoint** to find I/O points on demand facilities given p_0 as the supply point.

3.3.1 Analysis

As in the previous algorithm, the HLD and VLD can be computed in $O(n \log n)$ time using the method described in 2.2.1. Also, the candidate I/O points can be determined in $O(n)$ time. In Step 3, we consider each horizontal(vertical) line segment and compute its shortest y-distance(x-distance) to all the demand facilities. For each line segment this procedure takes $O(n)$ time (refer to section 2.2 and its analysis for details). Hence the running time for Step 3 is $O(n^2)$. In Step 4, the optimal location of the supply point is determined in $O(n \log n)$ time using the method described in 2.2.3. Step 5 fixes the location of the supply point p_0 to the intersection point found in Step 4 and calls algorithm **SFL-FixedSupplyPoint**, which runs in $O(n)$ time.

Thus, our algorithm that solves the above version of the shop floor layout problem runs in $O(n^2)$ time, which is a substantial improvement over the $O(n^6)$ algorithm of Wang *et. al.* [35].

Theorem 3.3.1. *The shop floor layout problem, where the supply facility is a point whose location in the layout is not known, can be solved in $O(n^2)$ time.*

3.4 Location of Supply Point Restricted to a Given Convex Region

In the previous sections, the supply facility was a point lying anywhere in the feasible region of the layout. The location of the supply point was either known or its optimal location had to be determined. If the supply point is constrained to lie within a given convex region, we can easily handle this requirement in Step 5 of the previous algorithm during the plane sweep process to find the optimal intersection point. We only consider those intersection points that lie inside the constrained region. The given constrained region can be a simple rectangle as seen in figure 3.2 or in a more complicated convex rectilinear polygon such as the one seen in figure 3.3.

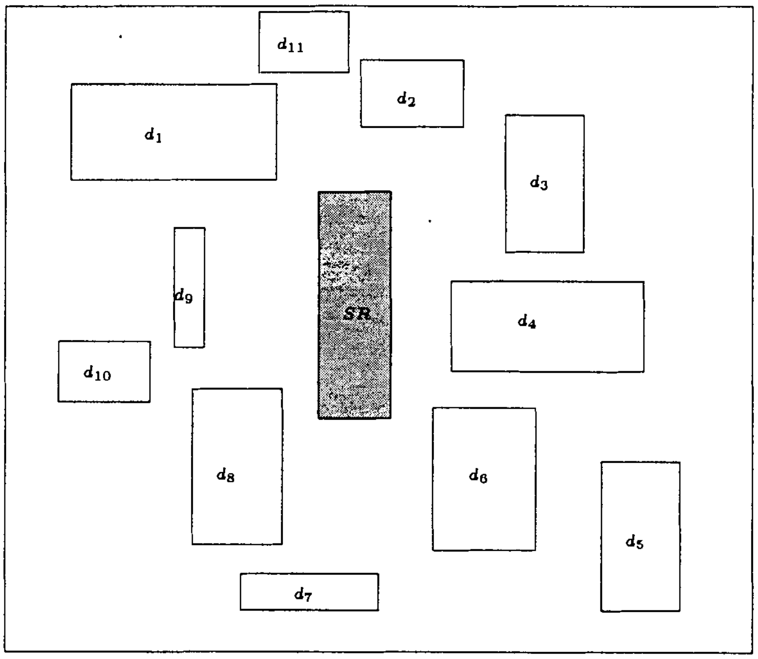

Figure 3.2: Shop Layout Problem - Locating supply point in a given rectangular region.

Theorem 3.4.1. *The shop floor layout problem, where the supply facility is a point whose location in the layout is not known and is constrained to lie in a given convex rectilinear region, can be solved in $O(mn)$ time, where the number of horizontal and*

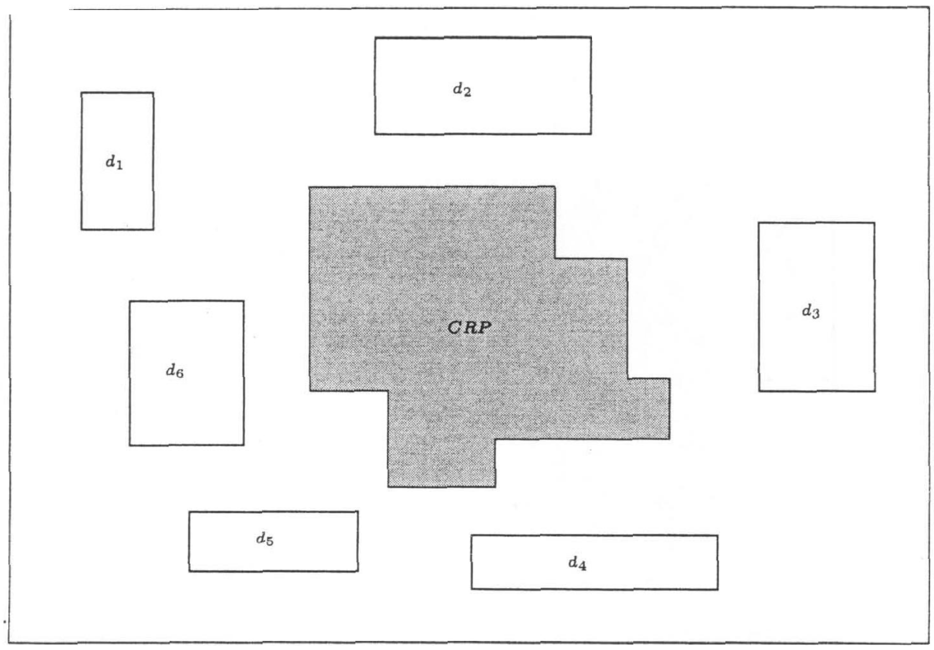

Figure 3.3: Convex rectilinear polygon containing the supply point

vertical ine segments intersecting the convex rectilinear region is $O(m)$ *and* $m \ll n$.

3.5 Conclusions

In this chapter we studied the shop floor layout problem first introduced by Wang *et. al.* [35]. The layout consisted of several rectangular demand facilities that were axis-parallel. All distances within the layout were calculated using the rectilinear L_1 metric. In the first version of the SFL problem, we were given a fixed supply point and the goal was to determine the optimal I/O points for the demand facilities. We presented an $O(n \log n)$ time algorithm to solve this problem, which is a significant improvement over the $O(n^4)$ algorithm in [35]. In the second version, we were required to determine the optimal location of the supply point, and find the corresponding I/O points of the demand facilities. We solved this problem in $O(n^2)$ time, compared to the $O(n^6)$ algorithm in [35]. Finally, by making a small change to the algorithms above, we solved the problem when the supply point is constrained to lie in a given convex region without blowing up the complexity.

Chapter 4

The Constrained Clustering Problem

4.1 Introduction

The clustering problem has been studied extensively in many different areas such as facility location, data mining, computational statistics, AI, pattern recognition, and most recently in the field of computational molecular biology. Clustering is the process of partitioning a given set of points into disjoint clusters in a way that similar points are placed in the same cluster. The similarity between points can be measured using any metric, e.g., Euclidean/Manhattan distance function. The goal is to find a "good" cluster, i.e. one that minimizes the objective function given the metric being used.

In facility location, a service provider (such as a postal service or a bank) may want to place a number of service centres (mail boxes or ATM machines) in a given geographical region of customers, with the objective that these centres be located conveniently for all customers [17]. A clustering algorithm can be applied to group the customers into sub-regions or clusters, and the service centres can be placed optimally in each cluster. Clustering also has applications in data mining [14], such as identifying a group of policy holders of an insurance company that have a high claim cost. Molecular biologists use clustering techniques to divide genes into groups based on their expression patterns. This can help in identifying the roles and functions of various genes.

Based on the application involved, many different methods such as BIRCH [37], DBSCAN, ISODATA [18], CLARA, CLARANS, COE-CLARANS [17], AUTOCLUST+ [13], etc. have been used to solve the clustering problem. One of the most common technique used is the k-means method [19], where the number of clusters to be formed is given by an input parameter k. Given n data points in d-dimensional space and a value k, the goal is to find k clusters such that a given objective function is minimized. The k-means clustering algorithm is an iterative process which begins with the selection of a set of k cluster centres. Each data point is assigned to the cluster centre that is closest to it, such that the mean squared distance from each data point to its centre is minimized. Once all the data points have been assigned to their closest cluster centres, a new set of cluster centres is computed by taking the mean of the data points in each cluster. This data point assignment to a cluster and cluster centre updating process is repeated until the total mean squared distance cannot be minimized further. Instead of minimizing the mean of the squared distances, we can minimize the sum of the distances from the data points to their respective cluster centres. In this case, the updated cluster centre is the median of the data points assigned to that cluster. Although this iterative algorithm yields a locally optimal solution and does not guarantee an optimal result, it is the algorithm of choice for the practitioners in fields such as data mining and statistical analysis because of its simplicity and flexibility. This algorithm can also be applied as a solution purifier to another algorithm to reduce distortion.

4.1.1 Formal Definitions

Clustering: Given a set S of n objects (or data points), a positive integer k, and a distance function $df : S \times S \rightarrow \Re$, the objective of the clustering problem is to partition the set S into k disjoint clusters $\{C_1, C_2, ..., C_k\}$, such that the sum of the distances of the points to their respective cluster centres is minimized, i.e., $DIST = \sum_{i=1}^{k} disp(C_i, rep_i)$ is minimized.
The representative (or cluster centre) rep_i of a cluster C_i is normally chosen to be the centroid of all the points in that cluster. $disp(C_i, rep_i) = \sum_{p \in C_i} df(p, rep_i)$ is the *displacement* of C_i, i.e., the sum of the distances of each point in that cluster to the cluster centre. If rep_i is calculated as the centroid of the points lying inside cluster C_i then the problem is known as the k-means clustering problem. If rep_i is calculated as the 1-median of the points lying inside C_i, then the problem is known as the k-median clustering problem.

Iterative Clustering: Each iteration in an iterative clustering algorithm can be divided into two steps, *cluster assignment* and *cluster updating*. Let $C_{i,t}$ denote the i^{th} cluster in iteration t, and let $rep_{i,t}$ denote the cluster centre of $C_{i,t}$ in iteration t. $DIST_t$ is the total displacement over all clusters in iteration t

1. **Cluster Assignment** : In iteration t, assign each data object $x \in S$ to a cluster $C_{i,t}$, such that $rep_{i,t-1}$ is the closest to x among all cluster centres $rep_{j,t-1}$ $\forall$ $j = 1, \ldots, k$.

2. **Cluster Updating** : Recompute $rep_{i,t}$, the cluster centre for cluster $C_{i,t}$ $\forall$ $i = 1, \ldots, k$. Also compute $DIST_t$.

The algorithm performs the above two steps at each iteration and stops when $DIST_{t-1} - DIST_t \leq \delta$, where δ is a predefined value. Note that initially the k cluster centres are chosen randomly from the set S of data objects (or the centres can be chosen using heuristics based on density of points, k-farthest points, etc).

It is easy to see that a naive implementation of the iterative clustering method results in a running time of $O(tnk)$, where t is the number of iterations required to reach the local minima. It is assumed that the distance function is computable in constant time. Several implementations of the k-means algorithm exist which improve the overall running time of the algorithm. The papers [2], [19] use kd-trees to provide efficient implementations. However, no such efficient implementation method is known for the iterative k-median clustering.

4.1.2 Presence of Obstacles

So far we have discussed the clustering algorithms and their implementations in the absence of obstacles. In case of many real-life applications, it is possible that there are physical entities (or obstacles) that pose as barriers to travel. These obstacles are part of the input to the clustering problem and need to be considered when calculating distances to assign the data points to the appropriate centres.

Some work has been done previously to address the presence of obstacles in the clustering problem. The COD-CLARANS algorithm by Tung *et. al.* [33] uses a partitioning based clustering method where the number of clusters to be formed is known apriori. It uses the CLARANS algorithm [27] which builds a visibility graph in order to compute the shortest distance between the data points in the presence of obstacles. The construction of this visibility graph is computationally expensive, and adds to the overall complexity of the algorithm. In [13], the authors use the Delaunay data structure to cluster a set of data points. The distances between points are measured in terms of the lengths of the edges between the points in the delaunay triangulation. Also, the number of clusters cannot be specified beforehand. Zaïane and Lee [36] use a variation of the DBSCAN algorithm called the DBCluC method, which is a density-based clustering algorithm.

In the remainder of this chapter, we discuss an efficient implementation of the iterative k-median algorithm that clusters the given set of data points in the presence of obstacles. The number of clusters to be formed is given by the input parameter k. In k-means clustering the new cluster centres are the centroid of the data points in the corresponding clusters. This method of updating the centres does not work in the presence of obstacles as it can result in some centres being placed in inaccessible regions, such as the inside of an obstacle. As illustrated in figure 4.1, the centroid of the data points around the golf-course happens to fall inside the golf-course, and this is unacceptable if our goal is to place a facility, say a mail-box. For this reason, we compute the 1-median of the data points in a given cluster to update its centre (as discussed in Chapter 2 of this thesis). This will guarantee that the updated centre lies outside the boundaries of the obstacles. Also, it should be noted that the Euclidean and Manhattan distance functions give an inaccurate measure of the distance between the data points and the cluster centres as the paths between these points could intersect obstacles in the region (as in the case of the data points and cluster centre around the housing block). In order to overcome this problem, we use rectilinear geodesic distances in the presence of obstacles. Also, we assume the obstacles are axis-parallel rectangles. As we will see later in this chapter, these two relaxations allow us to implement the iterative constrained k-median algorithm efficiently.

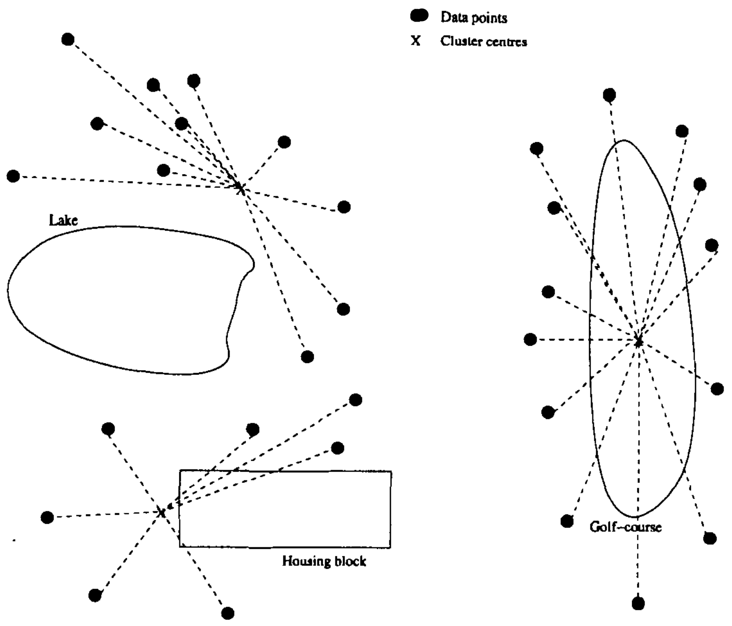

Figure 4.1: Effects of the presence of obstacles on the clustering problem

4.2 Our Algorithm

Let $S = \{s_1, s_2, \ldots, s_n\}$ denote a set of n data points lying in a region $\mathcal{R}$ of the 2-D plane and let $\mathcal{O} = \{o_1, o_2, \ldots, o_m\}$ denote the set of m disjoint axis-parallel rectangular obstacles also lying in $\mathcal{R}$. Note that the data points cannot lie in the interior of the obstacles. Also given is the value k, which denotes the number of clusters to be formed by the clustering algorithm. We present the iterative **k-median clustering algorithm**, which is similar to the iterative k-means algorithm. The two phases again are the cluster assignment and the cluster update (as discussed in 4.1.1). The goal is to create k clusters $\{C_1, C_2, \ldots, C_k\}$ such that each data point belongs to exactly one of these k clusters. We explain one iteration of this algorithm. For the first iteration we start with k initial cluster centres which are generated either randomly or using some heuristics. Let the set of initial centres be denoted by $\{rep_1, rep_2, \ldots, rep_k\}$.

The iterative k-median algorithm first computes the horizontal and vertical line decomposition (HLD and VLD) using only the obstacles in the region (i.e. the data points are not considered while constructing HLD and VLD). Details of how to compute the HLD and VLD can be found in section 2.2.1. As mentioned earlier, the clustering process comprises of two phases: (i) the cluster assignment phase where the data points are assigned to the closest cluster centre, and (ii) cluster update phase where the cluster centres are updated based on the assignments made in the previous phase. We now examine each of these phases in detail in the subsequent sections.

4.2.1 Cluster Assignment Phase

The objective is to assign each of the n data points in S to the cluster centre that is closest to it. Property 1.2.1 states that the shortest path between any cluster centre and any data point is either x-monotone (but not y-monotone) or y-monotone (but not x-monotone), or xy-monotone. A point p is non-y-monotone to a point q if the shortest path from p to q is x-monotone but not y-monotone. Similarly, a point p is non-x-monotone to a point q if the shortest path from p to q is y-monotone but not x-monotone.

For each data point $p \in S$ we find the closest cluster centre that is non-x-monotone, the closest cluster centre that is non-y-monotone and the closest centre that is xy-monotone. We then assign p to the closest of these three centres. We now divide this phase into three parts, i.e. (i) finding closest non-x-monotone cluster centres of the data points, (ii) finding closest non-y-monotone cluster centres for data points, and (iii) finding closest xy-monotone cluster centres for data points.

The iterative k-median clustering algorithm is formally described below:

```
Algorithm: ClusterAssignmentPhase
Input: Set of data points $S = \{s_1, s_2, .., s_n\}$, set of cluster centres $\{rep_1, rep_2, .., rep_k\}$,
set of obstacles $\mathcal{O} = \{o_1, o_2, .., o_m\}$.
Output: Set of clusters $\{C_1, C_2, .., C_k\}$.

Step 1: Compute the HLD and the VLD of the set of obstacles.

Step 2: For each data point $s \in S$ do
        (a) Determine the closest non-x-monotone cluster centre, $rep_x$.
        (b) Determine the closest non-y-monotone cluster centre, $rep_y$.
        (c) Determine the closest xy-monotone cluster centre, $rep_{xy}$.
        (d) Assign $s$ to the cluster represented by the cluster centre among $rep_x$,
        $rep_y$ and $rep_{xy}$ that is closest to $s$.
```

We now present a detailed description for each step of the above algorithm. Step 1 can be performed as described in previous chapters.

4.2.1.1 Step 2(a,b): Finding closest non-x-monotone centres

We explain the procedure to find the closest non-x-monotone cluster centres for the data points in S. The same method can be applied to find the closest non-y-monotone centres for the data points. As seen in fig.4.2, all the data points shown (depicted by shaded dots) are non-x-monotone with respect to the cluster centres shown (marked by an 'X').

During each iteration of the clustering algorithm, the set of cluster centres $rep_1, rep_2, \ldots, rep_k$ are known (computed in the previous iteration). For each cluster centre rep_i, $i = 1, \ldots, k$, we draw y(+x) and y(-x) paths starting at rep_i (see section 1.2 for definitions of such paths). All the points in the region defined by these two paths of a cluster centre are non-x-monotone to that centre. For each cluster centre we use the technique described in section 2.2 to find the distance to all the obstacles that fall in the region defined by these two paths. At each corner vertex of the obstacles, we store the cluster centre that is closest to that vertex and the distance to the corresponding cluster centre. Figure 4.2 shows the y(+x) and y(-x) paths of the three cluster centres rep_1, rep_2 and rep_3, represented by the dotted vertical lines.

Once all the cluster centres have been processed in this manner, we assign each data

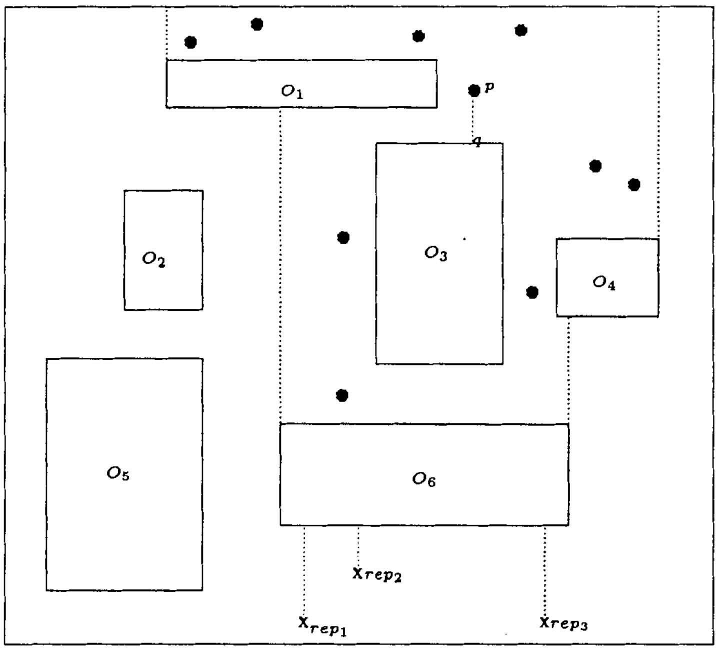

Figure 4.2: Finding non-x-monotone cluster centres for data point

point $p \in S$ by projecting p downwards to the first obstacle it hits. As seen in figure 4.2, point p has been projected onto the obstacle O_3. Let q be the projection of p on O_3. We have already computed the cluster centres that are closest to the corner vertices of O_3. By comparing the distances of the two cluster centres stored at the top-left and top-right corners of O_3 (the same cluster centre could be closest to both these corner vertices), we can determine the cluster centre rep_i that is closer to q (and hence to p) and the distance of q to rep_i. Note that if a data point p does not project onto any obstacle, then it implies that there is no cluster centre below p that is non-x-monotone to it.

Similarly, by drawing -y(-x) and -y(+x) paths from the cluster centre, we can determine the data points below the cluster centre that are non-x-monotone to it. We repeat the entire procedure for all cluster centres to determine the points that are non-x-monotone, and for each data point we determine the closest non-x-monotone cluster centre.

Thus, in order to determine the closest non-x-monotone (non-y-monotone) cluster centres of data points, we consider each cluster centre and determine the region containing the data points that are non-x-monotone (non-y-monotone) to it. Knowing the HLD(VLD), these regions can be determined in $O(m)$ time per cluster centre. Thus for all k cluster centres this takes $O(km)$ time. Projecting each data point onto an obstacle to determine the closest non-x-monotone (non-y-monotone) cluster centre takes constant time if the HLD and VLD of the data points and obstacles is known. Computing projections for all data points thus requires $O(n)$ time.

4.2.1.2 Step 2(c): Finding closest xy-monotone centres

We now explain the procedure to find the closest xy-monotone cluster centre for the data points. Consider the horizontal lines in the set H created by the horizontal line decomposition performed earlier. These lines divide the entire region into a number of slabs (see figure 4.3). Let the set of slabs be denoted by $SL = \{sl_1, sl_2 \ldots\}$. The total number of slabs created is $O(m)$, since there are m obstacles. Each slab contains data points from S, and a data point belongs to exactly one slab. Let the number of data points in slab sl_i be n_i. To find the closest xy-monotone centres of data points, we process slabs one at a time. A slab may or may not contain cluster centres. For each data point in the slab, we first find the closest xy-monotone cluster centre outside the slab (both above and below the slab). It is then compared with the closest cluster centre lying inside the slab (if such a cluster centre exists) to determine the actual closest xy-monotone cluster centre.

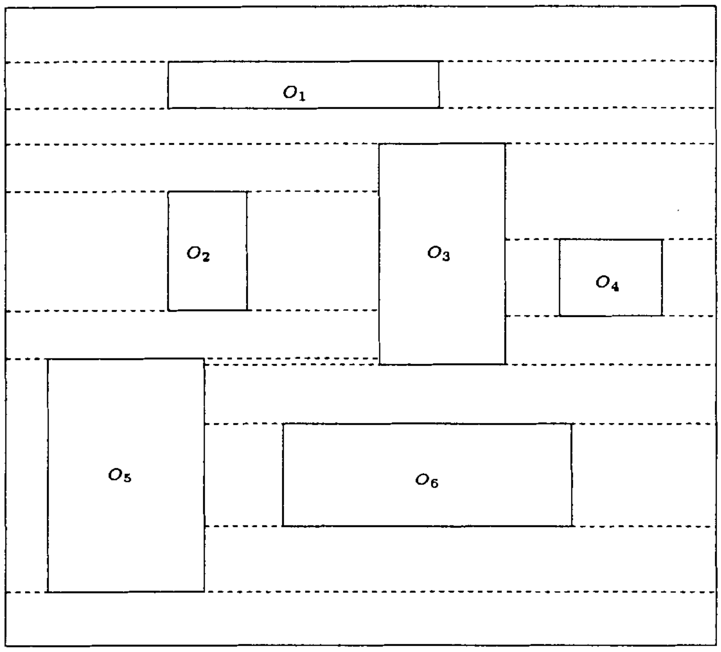

Figure 4.3: Generating slabs in the region containing data points and obstacles

Finding closest cluster centre lying above/below the slab

Consider a slab $sl \in SL$. We denote the top-left, top-right, bottom-left and bottom-right corners of sl by $tl(sl), tr(sl), bl(sl)$ and $br(sl)$ respectively. The left and right edges of sl are parts of the edges of obstacles or the boundary in the given region. All cluster centres that lie to the left and right, respectively, of these obstacles are not xy-monotone to the data points in sl and have been considered in the procedure for non-x-monotone and non-y-monotone cluster centres. For example, in figure 4.4, the cluster centres rep_1 and rep_2 are not xy-monotone to the data points in the highlighted slab and thus are not considered in this procedure. We find these non-xy-monotone cluster centres for slab sl in the following manner. From the top-left corner of slab sl we construct the -x(+y) path, and from the bottom-left corner of sl we construct the -x(-y) path. The entire region enclosed by these paths is not xy-monotone to the data points lying in sl. Similarly, we construct +x(+y) and +x(-y) paths from the top-right and bottom right corners of sl, and the region enclosed by these paths is not xy-monotone to the data points in sl. The definitions of these paths can be found in section 1.2. As we already know the set H (constructed by the HLD) containing horizontal lines passing through the top and bottom edges of obstacles, we can compute these paths in $O(m)$ time. Each cluster centre can be tested to determine if it lies inside the region defined by these paths by simply projecting the centre in the +y and -y directions. If the projected centre hits the paths in both directions, then it lies inside the non-xy-monotone region.

All cluster centres (not lying inside the slab sl) that are xy-monotone to at least some of the data points in sl are to be considered. We partition the lower edge of sl into smaller segments using the technique described below. Each such segment is closest to one cluster centre lying below sl and thus by projecting a data point inside sl onto the lower edge we can determine which cluster centre lying below the slab is closest to that data point. The same procedure is applied to the upper edge of sl and for a data point p we can thus determine the closest cluster centre below it and the closest cluster centre above it, and then choose the minimum among the two to be the closest cluster centre to p lying outside sl.

To divide the lower edge of sl (denoted by $le(sl)$) into smaller segments, we need to know the shortest distances from the cluster centres lying below sl to $le(sl)$. See figure 4.4. Some cluster centres are xy-monotone to all the data points in sl (such as rep_3), and some cluster centres are xy-monotone to only a few of the points in sl (such as rep_4). For each cluster centre we maintain its distance to $le(sl)$ and depict the distance function using the graph representation shown in figure 4.8. For a cluster centre that is xy-monotone to all the data points in sl (for example rep_3), the distance function is a vertical line going upwards representing the distance to $le(sl)$ and then a 45^o line for movement in the (+x)-direction and a 135^o line for movement in the (-x)-direction. The lower envelope of all the distance functions (highlighted in fig. 4.8) divides the edge $le(sl)$ into segments or intervals (I_0, I_1), (I_1, I_2),... . If we project a data point from sl onto the lower edge $le(sl)$ and it falls in the interval (I_1, I_2), then

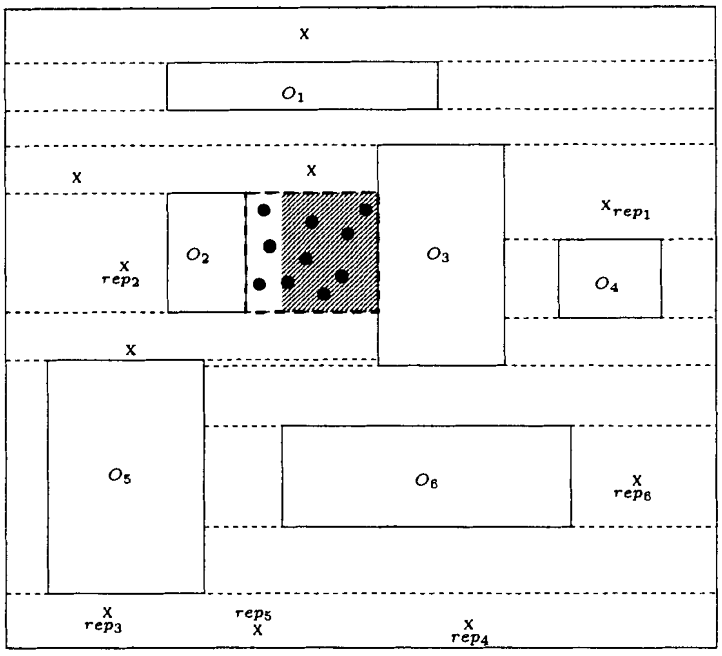

Figure 4.4: Finding closest xy-monotone cluster centres for data points

we know that it is closest to the cluster centre rep_5. For a cluster centre that is not xy-monotone to all the data points in sl (such as rep_4 in 4.4), we can determine the region containing those data points that do not need to be considered for this cluster centre and assign a value of infinity to the distance function corresponding to that region for the cluster centre. Thus the distance function representing the region of data points in sl that are not xy-monotone to rep_4 does not contribute to the lower envelope of the overall distance function. In figure 4.4, the data points that lie in the shaded region are not xy-monotone to rep_4 and thus do not contribute to the distance function (or have a distance of infinity to rep_4). But the data points lying in the unshaded region are xy-monotone to rep_4 and thus rep_4 is a candidate for closest cluster centre to these points.

Computing the distance functions

We now provide a detailed description of how the distance functions are calculated. We borrow the idea of calculating distance functions from Kusakari and Nishizeki citekusakari. Consider a cluster centre *rep*. As before, we can easily determine the non-xy-monotone and xy-monotone regions above *rep* by drawing y(+x) and y(-x) paths from *rep*. These two paths of *rep* follow a common route until they hit an obstacle. All data points lying inside the slabs that intersect this common route are xy-monotone to *rep*. The slabs that intersect the rest of the paths may have some data points that are not xy-monotone with respect to *rep*. If a slab *sl* intersects the y(-x) path of *rep* at a point A, then the distance function for slab *sl* (with respect to *rep*) is a 135^o degree line starting at A. Similarly, if a slab *sl* intersects the y(+x) path of *rep* at a point B, then the distance function for slab *sl* (with respect to *rep*) is a 45^o degree line starting at B. We need not consider slabs that lie entirely in the region defined by the two paths (since all points inside such a slab are not xy-monotone to *rep*). For all the other slabs above *rep* not encountered by the two paths, all data points lying inside are xy-monotone to *rep*. If such a slab lies to the right of *rep*, then the distance function is represented by a 45^o line starting at the bottom left corner of that slab. Similarly, if the slab lies to the left of *rep*, then the distance function is represented by a 135^o line starting at the bottom right corner of that slab. We use figures 4.5 and 4.6 to explain by example.

In figure 4.5, rep_4 is the cluster centre under consideration. $slab_1$ lies on the route common to both, the y(-x) and y(+x) paths of rep_4. Thus all the data points in $slab_1$ are xy-monotone to rep_4. The corresponding distance function is shown in 4.6. Now consider $slab_2$ in figure 4.6. The slab intersects the y(-x) path of rep_4 at A. Only those points in $slab_2$ that lie to the left of A are xy-monotone to rep_4. Thus, the distance function for $slab_2$ is a 135^o line starting at A. This is depicted in figure 4.6. Similarly, $slab_3$ in figure 4.5 intersects the y(+x) path of rep_4 at B. Only those points in $slab_3$ that lie to the right of B are xy-monotone to rep_4. Thus, the distance function for $slab_3$ is a 45^o line starting at B. This is also depicted in figure 4.6.

Using the procedure described above, we can compute the distance functions for each of the cluster centres to the slabs. The y(+x) and y(-x) paths can be constructed on $O(m)$ time per cluster centre, since the HLD and VLD are already known. The

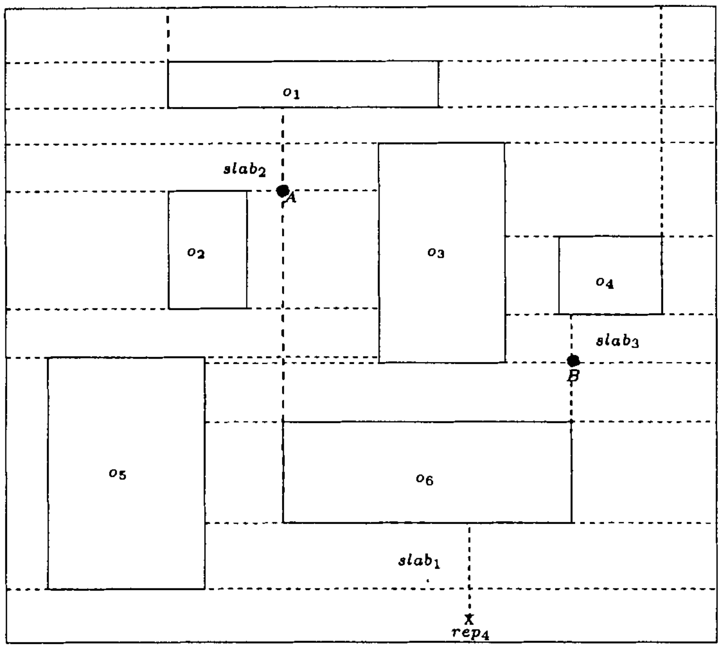

Figure 4.5: Computing distance functions for slabs to a cluster centre

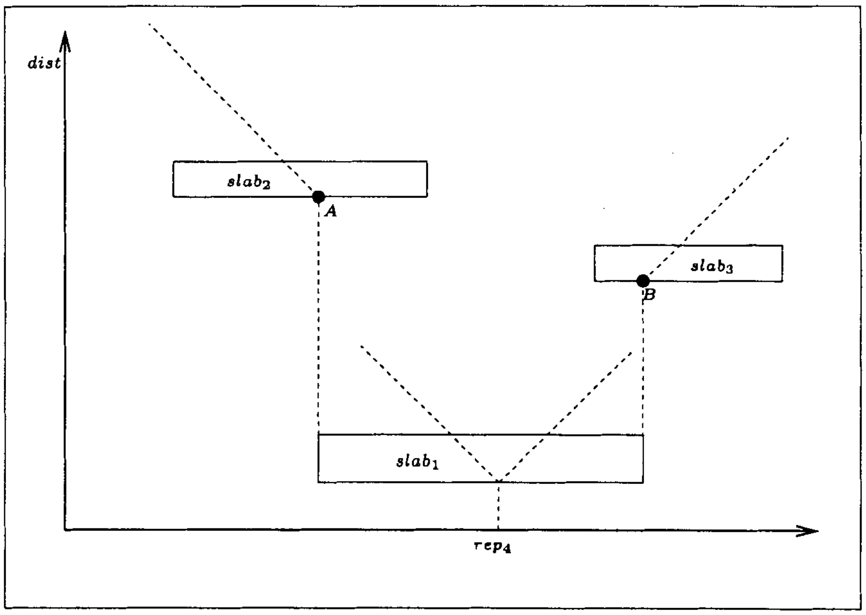

Figure 4.6: Graphical representation of the distance functions to slabs from rep_4

only drawback of this method is that it requires $O(km)$ storage space over all slabs, since we store information regarding the paths of each cluster centre with the slabs. In order to avoid this storage space complexity, we approach this problem in a systematic manner as follows. Suppose we start from the lowest slab in the region. As we move up, we encounter horizontal lines from H. Each such horizontal line h was created due to an obstacle. If h was created by the lower edge of an obstacle o, then the current slab is divided into two new slabs lying on either side of o. If h was created by the top edge of o, then two slabs on either side of o are merged to form a bigger slab above o. Consider figure 4.7. The lower edge of obstacle o_5 divides $slab_1$ into $slab_2$ and $slab_3$ lying to the left and right of o_5. Also, the upper edge of o_1 causes $slab_4$ and $slab_5$ to merge into a bigger slab $slab_6$ lying above o_1.

Thus, we can process each slab at a time and discard the path intersection and distance function information regarding the slab after we have finished processing it. We then move to the next slab and determine the information required to process that slab.

Finding the lower envelope of the distance functions

The lower envelope of the distance functions can be computed using a divide and conquer technique. The 45^o and 135^o rays are considered as individual line segments. Thus we are given $O(k)$ line segments (since there are k cluster centres and each can contribute one 45^o line segment and one 135^o line segment to the distance functions). The x-coordinates of the lower end point of a line segment is referred to as the x-value of that line segment. The line segments are first sorted based on their x-values. We then divide the set of line segments into two half sets based on x-value of the line segment in the middle of the set, and recurse on each of the two sets. The lower envelope of each subset can be represented by a polygonal x-monotone chain. The lower envelopes of two adjacent subsets are then merged together by finding the intersection points of their respective envelopes. Using the leftmost segment of the right subset, we can determine in logarithmic time (using binary search) the point where the segment intersects the envelope of the left subset. All points to the right of this

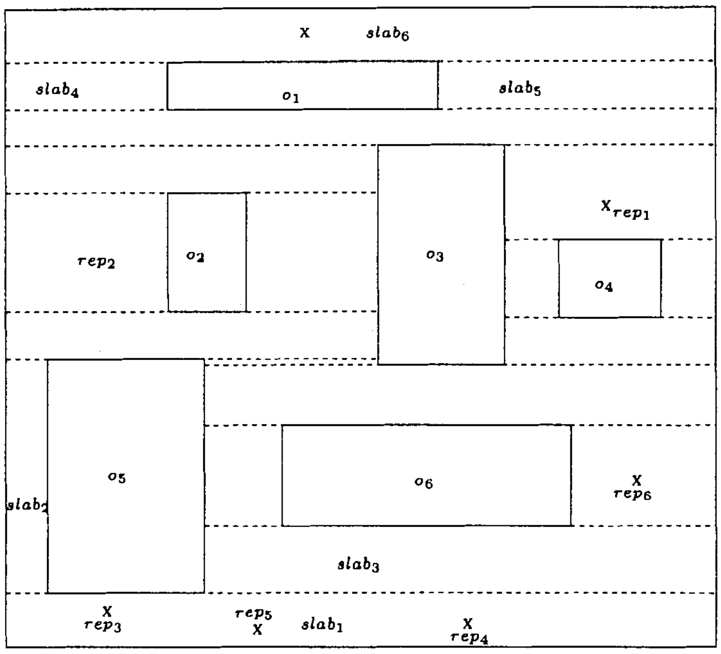

Figure 4.7: Calculating distance functions systematically

intersection point in the left subset can now be discarded, and the chains are merged. In this manner, we can merge the solutions of the subsets to obtain the lower envelope of the line segments. The lower envelope can thus be computed in $O(k \log k)$ time.

Finding closest cluster centre lying inside the slab

For all data points lying inside a slab, we have discussed how we can find the closest xy-monotone cluster centres lying above and below the slab. We also need to consider the cluster centres (if any) lying inside the slab as candidates for the closest xy-monotone centres to the data points. Note that the region defined by the slab is an unconstrained region, i.e. all cluster centres inside a given slab are xy-monotone to all the data points inside that slab. We use the properties of the Voronoi diagram to achieve our goal here. Given the cluster centres inside a slab, we can construct a Voronoi diagram such that the region defined by the slab is partitioned into cells where each cell contains exactly one cluster centre. All data points lying inside a given cell are closest to the cluster centre of that cell, than to the cluster centres of any

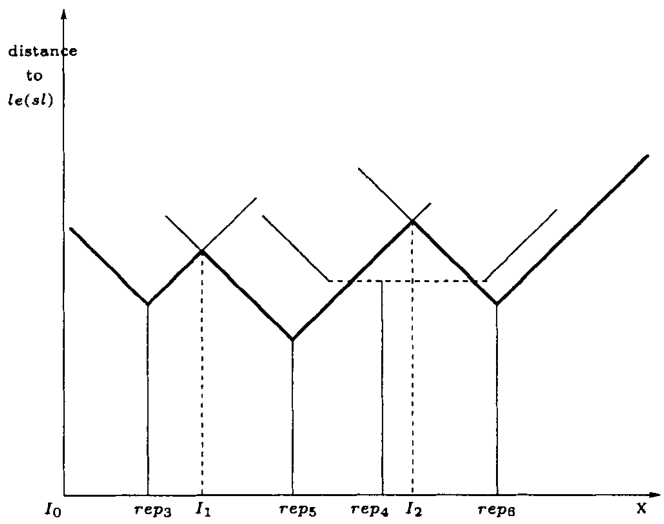

Figure 4.8: Graph representing distance functions of cluster centres to lower edge of slab sl

other cells. The Voronoi diagram for L_1 metric can be constructed using a divide and conquer approach in $O(k_i \log k_i)$ time [24], where k_i is the number of cluster centres in the given slab. Also, we can query the Voronoi data structure to determine which cell a given data point belongs to in $O(\log k_i)$ time. Thus, for a given slab this procedure requires a total of $O(k_i \log k_i + n_i \log k_i)$ time, where n_i is the number of data points lying inside the slab. Figure 4.9 shows the Voronoi diagram of a slab containing seven cluster centres.

We have explained how to find the closest xy-monotone cluster centres lying below data points in a given slab. The same technique can be used to find the closest xy-monotone cluster centres lying above the slab. We also determine the closest xy-monotone cluster centre lying within the slab. For each data point we choose the closer of the above three cluster centres to be the closest xy-monotone cluster centre. We then assign the data point to the cluster centre that is closest to it (from the closest non-x-monotone centre, the closest non-y-monotone centre and the closest xy-monotone centre).

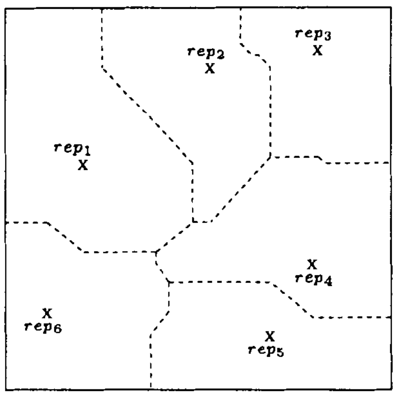

Figure 4.9: Voronoi representation of a slab with cluster centres

Let us analyze the time required to determine the closest xy-monotone cluster centre to the data points. For the slabs not containing cluster centres, we consider each slab and find the regions that are not xy-monotone to the slab (and hence to the data points lying inside the slab). Since we know the HLD (VLD), this process requires $O(m)$ time to determine the region, plus $O(k)$ time to test whether the cluster centres fall inside the region. Thus, for all slabs this procedure requires $O(m(m+k))$ time (as there are $O(m)$ slabs). Calculating the distance functions for a slab takes $O(km)$ time. The lower envelope of the distance functions for a slab is obtained in $O(k \log k)$ time. Thus, over all slabs, this step requires $O(m(km + k \log k))$ time. Finally, for each slab containing cluster centres, we create Voronoi diagrams of the cluster centres and determine which cell each data point lies in. Over all slabs, this step requires $O(k \log k + n \log k)$ time (as each data point and cluster centre lies exactly in one slab). Therefore, the entire cluster assignment phase takes $O(km^2)$ time.

4.2.2 Cluster Updating Phase

Once all the data points have been assigned to their respective closest cluster centres, we update the cluster centre rep_j of each cluster C_j, $j = 1, \ldots, k$, by computing the 1-median point of all the data points assigned to that cluster. We use the procedures described in chapter 2 to compute the 1-median of a set of points.

The cluster assignment and cluster updating phases are repeated until the solution cannot be improved further (as in the case of the algorithm defined in section 4.1.1).

4.3 Analysis

We analyze the running time of one iteration of the k-median clustering algorithm described above. From section 2.2.1, we know that the HLD and the VLD of the obstacles can be generated in $O(m \log m)$ time. In section 4.2.1.1 we have seen how the closest non-x-monotone cluster centres for the data points can be computed in $O(km + n)$ time. We can also compute the closest non-y-monotone cluster centres for the data points in $O(km + n)$ time.

Section 4.2.1.2 shows how the closest xy-monotone cluster centres (lying outside the slab) of the data points in a given slab can be computed in $O(km + k \log k)$ time. Thus for all slabs this procedure requires $O(m(km + k \log k))$ time. The closest cluster centres lying inside the slabs can be determined in $O(k \log k + n \log k)$ time using Voronoi diagrams.

We require a total of $O(n)$ time to determine the closest of the three cluster centres (non-x-monotone, non-y-monotone, xy-monotone) for all data points. We update each cluster centres by computing the 1-median of the points assigned to the clusters. For the clustering process it is common that the number of data points is much more than the number of obstacles or cluster centres, i.e., $k \ll n$ and $m \ll n$. Therefore we use the algorithm described in section 2.4 of chapter 2 which runs in $O(mn \log n)$ time.

The running time (per iteration) of our algorithm is $O(km^2 + mn \log n)$. The algorithm described above requires linear storage space, i.e., $O(m + n)$, to hold information about the data points, clusters and obstacles.

4.4 Conclusion

In this chapter, we have presented an efficient implementation of the k-median algorithm which uses the techniques described in chapter 2 to compute distances between the data points and the cluster centres. Unlike most clustering algorithms, our algorithm works under the presence of obstacles in the region and can quickly recompute the new cluster centres after every iteration using the 1-median algorithm presented in section 2.4. The iterative k-median clustering algorithm requires a total running time of $O(mn \log n + km^2)$ per iteration.

Chapter 5

Conclusions and Future Work

5.1 Summary

We conclude this thesis with a brief summary of our accomplishments. We study three facility location problems in the presence of obstacles: (i) the 1-median problem, (ii) the shop floor layout problem, and (iii) the iterative k-median clustering problem.

For all the algorithms presented in this thesis, we assume that the obstacles are axis-parallel rectangular objects that occupy space and pose as barriers to travel. Distances are computed using the L_1 geodesic metric.

Chapter 1 provides a brief introduction to facility location in the presence of obstacles and the preliminaries.

In chapter 2, we study the 1-median problem. An algorithm is presented to solve the 1-median problem, and this algorithm improves upon the running time of the algorithm presented by Choi *et. al.* [8]. The running time of our algorithm is $O(n(n+m))$, where n is the number of points and m is the number of obstacles. A modified algorithm is presented for the case where the number of points is considerably larger than the number of obstacles. This modified algorithm runs in $O(mn \log n)$ time. Our algorithm also has a reduced storage space requirement of $O(m+n)$, as compared to the $O(mn)$ requirement of the algorithm presented in [8].

Chapter 3 presents the shop floor layout problem as introduced by Wang *et. al.* in [35]. Chapter 1 establishes important results that are used to improve known results for several versions of the shop floor layout problem. For the case where the supply facility is a fixed point, we present an $O(n \log n)$ time algorithm which is a major improvement over the $O(n^4)$ algorithm in [35]. We present $O(n^2)$ algorithms if the location of the supply facility is unknown, or if the supply facility lies inside a pre-specified convex region. The algorithms in [35] for the same require $O(n^6)$ time.

Chapter 4 provides a brief introduction to the clustering problem, and examines the iterative k-median approach for clustering in the presence of obstacles. Very few algorithms are known under this setting. In this chapter we use the results established in chapter 2. Each iteration of our clustering algorithm runs in $O(mn \log n + km^2)$ time. The use of the 1-median algorithm to update the cluster centre ensures that the updated centre lies in the feasible region of the plane (i.e. not inside any of the obstacles).

5.2 Future Work

We believe that the techniques used in this thesis can be exploited to improve algorithms for other facility location problems in the presence of obstacles. For example, the farthest neighbors and center point problems discussed by Ben-Moshe *et. al.* in [4] have properties similar to the problems we have studied. Other improvements can include considering arbitrarily oriented obstacles, general shapes of obstacles, etc.

It would be interesting to reduce the running time of the 1-median algorithm mentioned in chapter 2. Considering each source point and computing its distance contributions to the other source points and obstacles leads to a quadratic time algorithm. If the distance contributions of the source points can be updated from the distances calculated for a previous source point, it may be possible to attain a subquadratic algorithm to solve the 1-median algorithm. Another interesting problem is the general p-median problem in the presence of obstacles when $p \geq 2$.

In the shop floor layout problem, we considered having only one supply facility for all the demand facilities. This problem can be extended to consider the location of more than one supply facility. Also, there can be more than one I/O point on each demand facility to receive goods from different supply facilities. When the location of the supply facility is not known, our algorithm runs in quadratic $O(n^2)$ time. We believe that it is possible to improve the running time of this algorithm using some distance updating mechanisms. We note that improving the running time of the 1-median algorithm in chapter 2 will be key to solving this shop floor layout problem more efficiently.

Bibliography

[1] A. Aggarwal, M. Klawe, S. Moran, P. Shor and R. Wilber, "Geometric Applications of a Matrix Searching Algorithm", *Algorithmica*, 2, 195-208, 1987.

[2] K. Alsabti, S. Ranka and V. Singh, "An Efficient K-Means Clustering Algorithm", *IPPS: 11th International Parallel Processing Symposium*, IEEE Computer Society Press, 1998.

[3] C. Bajaj, "The algebraic degree of geometric optimization problems", *Discrete and Computational Geometry*, 3, 177-191, 1988.

[4] B. Ben-Moshe, M.J. Katz and J.S.B. Mitchell, "Farthest neighbors and center points in the presence of rectangular obstacles", *Symposium on Computational Geometry*, 164-171, 2001.

[5] J. Borenstein, H.R. Everett and L. Feng, " 'Where am I?' Sensors and Methods for Mobile Robot Positioning", *Technical Report*, University of Michigan, 1996.

[6] P. Bradley, K.P. Bennett and A. Demiriz, "Constrained K-means Clustering", *Microsoft Research Technical Report* 2000-65, May 2000.

[7] V. Chepoi and F. Dragan, "Computing a Median Point of a Simple Rectilinear Polygon", *Information Processing Letters*, 49, 281-285, 1994.

[8] J. Choi, C. Shin and S. Kim, "Computing Weighted Rectilinear Median and Center Set in the Presence of Obstacles", *ISAAC*, Lecture Notes of Computer Science, Springer-Verlag, 1533, 29-38, 1998.

[9] K.L. Clarkson, S. Kapoor and P.M. Vaidya, "Rectilinear shortest paths through polygonal obstacles in $O(n(\log n)^2)$ time", *Proceedings of the 3rd Annual ACM Symposium on Computational Geometry*, 251-257, 1987.

[10] T.H. Cormen, C.E. Leiserson and R.L. Rivest, *Introduction to Algorithms*, The M.I.T. Press, Cambridge, 1997.

[11] Z. Drezner, H.W. Hamacher, *Facility Location - Applications and Theory*, Springer-Verlag, 2002.

[12] C. Eldershaw and M. Hegland, "Cluster Analysis using Triangulation", *Computational Techniques and Applications*, World Scientific, 201-208, 1997.

[13] V. Estivill-Castro and I. Lee, "AUTOCLUST+: Automatic Clustering of Point-Data Sets in the Presence of Obstacles", *Temporal, Spatial, and Spatio-Temporal Data Mining*, 133-146, 2000.

[14] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy, *Advances in Knowledge Discovery and Data Mining*, The M.I.T. Press, Cambridge, MA, 02142, 2000.

[15] R.L. Francis and J.A. White, *Facility Layout and Location - An Analytical Approach*, Prentice Hall, Englewood Cliffs, NJ, 1974.

[16] S. Guha and I. Suzuki, "Proximity Problems and the Voronoi Diagram an a Rectilinear Plane with Rectangular Obstacles", *FSTTCS*, 218-227, 1993.

[17] J.F. Hou, *Clustering with Obstacle Entities*, M.Sc. Thesis, Simon Fraser University, 1999.

[18] A.K. Jain and R.C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, N.J., 1988.

[19] T. Kanungo, D.M. Mount, N.S. Netanyahu, C. Piatko, R. Silverman and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24, 2002.

[20] Y. Kusakari and T. Nishizeki, "An algorithm for finding a region with the minimum total L_1-distance from prescribed terminals", *Proc. of ISAAC '97*, Lecture Notes in Computer Science 1350, 324-333, 1997.

[21] R.C. Larson and A.R. Odoni, *Urban Operations Research*, Prentice-Hall, N.J., 1981.

[22] R.C. Larson and G. Sadiq, "Facility Locations with the Manhattan metric in the presence of barriers to travel", *Operations Research*, 31, 652-669, 1983.

[23] R.C. Larson and V.O.K. Li, "Finding Minimum Rectilinear Distance paths in the Presence of Barriers", *Networks*, 11, 285-304, 1981.

[24] D.T. Lee, "Two-Dimensional Voronoi Diagrams in the L_p-Metric", *Journal of the ACM (JACM)*, 27, 604 - 618, Oct 1980.

[25] T. Lengauer, *Combinatorial algorithms for integrated circuit layout*, John Wiley & Sons, Inc., New York, NY, 1990.

[26] J.J. Leonard and H.F. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons", *IEEE Trans. Robotics and Automation*, 7(3), 376-382, June 1991.

[27] R.T. Ng and J. Han, "CLARANS: A Method for Clustering Objects for Spatial Data Mining", *IEEE Transactions on Knowledge and Data Engineering*, 14(5), 1003-1016, 2002.

[28] T. Ohtsuki, *Layout Design and Verification*, New York: Elsevier, 1986.

[29] F.P. Preparata and M. Shamos, *Computational Geometry - An Introduction*, Springer-Verlag, 1985.

[30] P.J. de Rezende, D.T. Lee and Y.F. Wu, "Rectilinear shortest paths in the presence of rectangular barriers", *Discrete and Computational Geometry*, 4, 41-53, 1989.

[31] S. Savas, R. Batta and R. Nagi, "Finite-Size Facility Placement in the Presence of Barriers to Rectilinear Travel", *Operations Research*, 50(6), 1018-1031, 2002.

[32] D.D. Sleator and R. E. Tarjan. "Self-adjusting binary search trees", *Journal of the ACM*, 32(3), 652-86, 1985.

[33] A.K.H. Tung, J. Han, L.V.S. Lakshmanan and R. T.Ng, "Constraint-Based Clustering in Large Databases", *International Conference on Database Theory*, 405-419, 2001.

[34] A.K.H. Tung, J. Hou and J. Han, "Spatial Clustering in the Presence of Obstacles", *17th International Conference on Data Engineering*, 359, 2001.

[35] S.-J. Wang, J. Bhadury and R. Nagi, "Locating a Supply Facility and Input/Output Points in a Layout Problem", *Location Science/Computers and Operations Research*, 29(6), 685-699, 2002.

[36] O.R. Zaïane and C.-H. Lee, "Clustering Spatial Data in the Presence of Obstacles: A Density-Based Approach", *International Database Engineering and Applications Symposium (IDEAS'02)* , July 2002.

[37] T. Zhang, R. Ramakrishnan and M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases", *SIGMOD Conference*, 103-114, 1996.