Approximation algorithms for multi-facility location

Prashanth Kodela

University of Nevada, Las Vegas

Follow this and additional works at: https://digitalscholarship.unlv.edu/rtds

Repository Citation
https://digitalscholarship.unlv.edu/rtds/1454
INFORMATION TO USERS

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

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

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

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

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

UMI®
APPROXIMATION ALGORITHMS FOR MULTI-FACILITY LOCATION

by

Prashanth Kodela

Bachelor of Science
Kakatiya University, India
2000

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree
Department of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
December 2002
The Thesis prepared by

PRASHANTH KODELA

Entitled

APPROXIMATE ALGORITHMS FOR
MULTI - FACILITY LOCATION

is approved in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative
ABSTRACT

Approximation Algorithms for Multi-Facility Location

by

Prashanth Kodela

Dr. Laxmi Gewali, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

This thesis deals with the development and implementation of efficient algorithms to obtain acceptable solutions for the location of several facilities to serve customer sites. The general version of facility location problem is known to be NP-hard.

For locating multiple facilities we use Voronoi diagram of initial facility locations to partition the customer sites into $k$ clusters. On each Voronoi region, solutions for single facility problem is obtained by using both Weizfield's algorithm and Center of Gravity. The customer space is again partitioned by using the newly computed locations. This iteration is continued to obtain a better solution for multi-facility location problem. We call the resulting algorithm: "Voronoi driven $k$-median algorithm".

We report experimental results on several test data that include randomly distributed customers and distinctly clustered customers. The observed results show that the proposed approximation algorithm produces good results.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>vi</td>
</tr>
<tr>
<td>CHAPTER 1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>CHAPTER 2 PROXIMITY STRUCTURES</td>
<td>4</td>
</tr>
<tr>
<td>Introduction</td>
<td>4</td>
</tr>
<tr>
<td>Voronoi Diagram</td>
<td>4</td>
</tr>
<tr>
<td>Relative Neighborhood Graphs</td>
<td>7</td>
</tr>
<tr>
<td>Nearest Neighborhood Graphs</td>
<td>9</td>
</tr>
<tr>
<td>Gabriel Graphs</td>
<td>9</td>
</tr>
<tr>
<td>CHAPTER 3 BOUNDING CIRCLE AND FACILITY LOCATION</td>
<td>13</td>
</tr>
<tr>
<td>Problem Definition</td>
<td>13</td>
</tr>
<tr>
<td>Algorithms for Smallest Bounding Circle</td>
<td>14</td>
</tr>
<tr>
<td>Variations of Smallest Bounding Circle</td>
<td>17</td>
</tr>
<tr>
<td>CHAPTER 4 FACILITY LOCATION FOR WEIGHTED POINTS</td>
<td>18</td>
</tr>
<tr>
<td>Problem Definition</td>
<td>18</td>
</tr>
<tr>
<td>Weiszfield’s Algorithm and Center of Gravity</td>
<td>19</td>
</tr>
<tr>
<td>Multi Facility Location</td>
<td>23</td>
</tr>
<tr>
<td>Voronoi Driven k-Median Approximation</td>
<td>23</td>
</tr>
<tr>
<td>CHAPTER 5 IMPLEMENTATION</td>
<td>28</td>
</tr>
<tr>
<td>Introduction</td>
<td>28</td>
</tr>
<tr>
<td>Data Structures</td>
<td>29</td>
</tr>
<tr>
<td>Code Development</td>
<td>35</td>
</tr>
<tr>
<td>Experimental Results</td>
<td>38</td>
</tr>
<tr>
<td>CHAPTER 6 CONCLUSION</td>
<td>51</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>55</td>
</tr>
<tr>
<td>VITA</td>
<td>58</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
LIST OF FIGURES

2.1 Voronoi Diagram of 10 points sites ....................................................................... 5
2.2 Delaunay Triangulation for 10 points ................................................................. 6
2.3 Illustrating Lune of Influence .............................................................................. 8
2.4 Different proximity structures for a set of 10 points ........................................... 11
2.5 Nearest Neighborhood Graph ............................................................................ 12
2.6 Illustrating Gabriel Graph .................................................................................. 12

3.1 Smallest Enclosing Circle for 10 customer sites ................................................. 14
3.2 Furthest Point Voronoi Diagram .......................................................................... 15
3.3 Furthest Point Voronoi Diagram with circles ..................................................... 16

4.1 Illustrating center of gravity and 1-median for symmetrically distributed points with symmetrical weights ......................................................................... 21
4.2 Illustrating center of gravity and 1-median for non-symmetrically distributed points with non-symmetrical weights ................................................................. 21
4.3 Illustrating center of gravity and 1-median for randomly distributed points with random weights ........................................................................................................ 22
4.4 Initial Facility Locations ..................................................................................... 24
4.5 Locations After One Iteration ............................................................................ 25

5.1 Voronoi Diagram .................................................................................................. 30
5.2 Illustrating the DCEL data structure .................................................................... 31
5.3 Table 1: Vertex Records ..................................................................................... 32
5.4 Table 3: Face Records ......................................................................................... 32
5.5 Table 2: Edge Records ....................................................................................... 33
5.6 Initial Facility Location ....................................................................................... 41
5.7 Facility Location after one round of execution .................................................. 42
5.8 Facility Location after final round of execution .................................................. 43
5.9 Table 4: Results for Customers Randomly Distributed in Rectangular Region 44
5.10 Snapshot for clustered customer sites ................................................................. 45
5.11 Table 5: Results for Distinctly Clustered Customers ......................................... 46
5.12 Weighted Distance versus No. of Customers for Randomized Data ............. 47
5.13 Weighted Distance versus No. of Customers for Clustered Data ................. 48
5.14 Convergence Steps versus No. of Customers for Clustered Data ................. 49
5.15 Convergence Steps versus No. of Customers for Randomized Data ............. 50

6.1 Solution Modification ......................................................................................... 53
ACKNOWLEDGMENTS

I would like to thank Dr. Laxmi Gewali for chairing my committee and advising this work. For this and for being generous with his time when I needed it I am deeply indebted to him. I would also like to specifically thank Dr. John Minor, Dr. Wolfgang Bein, and Dr. Rama Venkat for serving on the committee. I would like to thank the faculty of the Computer Science Department of the University of Nevada, Las Vegas for the formal education along with generous financial support. I am grateful to my parents for their inspiration. Special thanks go to all my friends here for their support.
CHAPTER 1

INTRODUCTION

The problem of locating facilities to serve customers is an important problem in operations research, computer science, and business applications. Variations of this problem have been studied in pattern recognition, geographic information system, location science, and statistics [1-4, 13, 12, 24, 30, 34]. In real world applications, facility location problems arise for planning and stationing service centers that include airports, waste disposal sites, hospitals, merchandise distribution centers, post offices, and fire stations. Algorithms for solving facility location problems have also found applications in the planning of reactors [5]. Several criteria are used to measure the quality of the location of a facility. Some of the factors used for measuring the quality of a facility location include total distance from the facility to the customers, distance between the furthest customer and facility, total travel time, and safety.

In the simplest form of the facility location problem, we are given \( n \) customers in a region and we are required to find a location to station a single facility to optimize a certain objective function. In most applications the objective is to minimize travel cost to visit or access the facility.

Several generalized versions of facility location problems have been considered. In one generalized version, it is required to find locations to station \( k \) facilities to serve the customers. These kind of generalized facility location problems are also known as multi-facility location problems. Several efficient algorithms have been reported to solve the single facility location problems by researchers in computational geometry, operations research, and discrete mathematics. However, efficient and practical algorithms are yet to be developed.
for solving multi-facility location problems.

Facility location problems have been modelled both as geometric optimization problems in the Euclidean space and as network optimization problems in weighted graphs. In the geometric model, customers can be represented by points in the Euclidean space and the facility is required to be located inside the smallest convex region enclosing the customers. In the graph model, locations for facilities and customers are at the nodes of the graph and traversal is allowed only along the edges of the graph. Thus there are only finite possibilities to station facilities in a finite graph. On the other hand, possible locations for stationing a facility in the geometric model is not bounded. In the geometric model distance is usually measured by using an Euclidean metric. In the graph model, traversal distance between a facility and a customer is usually measured in terms of the shortest distance between them in the underlying graph.

When the objective function is taken as the total weighted distance from customers to facilities, the facility location problem can be succinctly stated as the statistical problem of finding the median of customer site locations. In this view, the single facility location problem becomes the 1-median problem and the multi-facility location problem becomes the k-median problem.

For most applications, the single facility location problem can be abstracted elegantly by considering customers as points distributed in the two dimensional plane. Many researchers in computational geometry have tried to simplify the single-facility location problem as the problem of enclosing a set of customer points in a two dimensional plane by the smallest circle. In this approach, the center of the smallest enclosing circle is taken as the desired location for stationing the facility. While this simplified approach is quite appealing intuitively, it does not necessarily minimize the total travel cost for customers. However, this technique works well for reducing the furthest distance between the facility and the customers.

In most facility location problems, a single point is considered for stationing the facility. While this simplification is widely accepted in application areas, some researchers have considered facility location problems where the facility is constrained to lie on or within
certain predefined features or regions. Theoretically efficient algorithms have been reported
to optimally station a facility constrained to lie along a line segment [24]. Problems in
which facilities are constrained to lie within a convex region in the two dimensional plane
in the presence of customer points, are reported in [24].

In this thesis, we present a review of the geometric approach for modelling and solving
facility location problems. The main contribution of this thesis is the development and
implementation of an approximation algorithm for locating $k$ facilities in the presence of
$n$ customer points distributed in the plane. We report the performance of the proposed
approximation algorithm by executing it on several randomly distributed and clustered
test data. Generated results reveal that the k-median approximation is quite effective in
computing high-quality locations.

The thesis is organized as follows. In Chapter 2, we review the geometric proximity
structures that are useful for modelling points distributed in the plane. In particular, we
review all important data structures that have been used for capturing the proximity re­
lationship of customer points in the plane that can be used to implement the proposed
k-median approximation algorithm. The proximity structures reviewed include Voronoi di­
agram, Delaunay triangulation, relative neighborhood graph, Gabriel graph, and nearest
neighbor graph. In Chapter 3, we examine the circle encloser approach for facility location
proposed by computational geometers. We also examine circle encloser problem when the
center of the circle is constrained to lie on or within certain linear features and convex
regions. In Chapter 4, we present the development of our k-median approximation algo­

rithm based on successive refinement by computing the Voronoi diagram and 1-medians
of initial $k$ locations. The proposed algorithm works for point customers with assigned
weights representing the number of customers at that point. In Chapter 5, we describe the
implementation detail of the proposed k-median approximation algorithm. We also perform
experimental evaluation of the approximation algorithm by executing the algorithm on sev­
eral test data. Finally, in Chapter 6, we describe the performance of the proposed algorithm
by inspecting and comparing the generated results. We also discuss possible extensions and
scope for further investigation.
CHAPTER 2

PROXIMITY STRUCTURES

2.1 Introduction

Several structures have been proposed in computational geometry literature to organize the proximity relationship of points distributed in the two dimensional plane\[11.30.32]\. Some of the widely used geometric structures include Voronoi Diagram, Delaunay Triangulation, Minimum Spanning Tree, Relative Neighborhood Graph, etc. We next give a brief description of these geometric structures that can be used for developing facility location algorithms.

2.2 Voronoi Diagram

A widely used structure to represent the distribution of point sites in Euclidean space is the Voronoi diagram. The Voronoi diagram structure has been applied in various disciplines that include ecology, physics, geographic information systems, computer graphics, geometric computation and pattern recognition.

Computation of Voronoi diagram is a rather well studied problem in computational geometry, and several efficient algorithms for its construction have been reported. The Voronoi diagram of \( n \) point sites \( p_1, p_2, \ldots, p_n \) in two dimensions is the partitioning of the plane into \( n \) convex regions \( V(1), V(2), \ldots, V(n) \) such that all points in the region \( V(i) \) are closer to site \( p_i \) than any other site. The region \( V(i) \) is called the Voronoi cell induced by site \( p_i \). Figure 2.1 shows the Voronoi diagram of 10 points in the plane. It may be noted that the Voronoi cells could be both bounded or unbounded. A Voronoi cell \( V(p_i) \) corresponding to site \( p_i \) consists of all the points at least as close to \( p_i \) as to any other site.
It can be formally written in set notation as:

$$V(p_i) = \{x : |p_i - x| \leq |p_j - x| \forall j \neq i\}$$

Voronoi diagrams can also be defined by using half-planes. Let $H(p_i, p_j)$ be the half-plane induced by the line through $p_i$ and $p_j$ that contains $p_j$. Then $H(p_i, p_j)$ can be viewed as the set of points closer to $p_i$ than $p_j$.

The Voronoi region $V(i)$ can be written as the intersection of half planes

$$V(p_i) = \bigcap_{i \neq j} H(p_i, p_j)$$

The Voronoi diagram can be used to obtain the triangulation of the plane with sites as vertices. In fact the dual of a Voronoi diagram is a triangulation called the Delaunay triangulation. From the Voronoi diagram one can get Delaunay triangulation as follows. If two Voronoi regions share an edge then the corresponding sites are connected by an edge. The structure obtained by making such connections is precisely the Delaunay triangulation. Figure 2.2 illustrates the Delaunay diagram of point sites of Figure 2.1. Thus Delaunay triangulation can be viewed as the dual of Voronoi diagram.

Voronoi diagram satisfies interesting properties that can be used to compute minimum
spanning trees, triangulation, convex hull, and nearest neighbors. Some of the widely applicable properties of Voronoi diagrams can be stated as follows

Properties of Voronoi diagram (and Delaunay Triangulation)

a. Each Voronoi region $V(p_i)$ is convex.

b. If $v$ is a Voronoi vertex at the junction of $V(p_1), V(p_2)$ and $V(p_3)$ then $v$ is the center of the circle $C(v)$ determined by $p_1, p_2$ and $p_3$.

c. The interior of $C(v)$ contains no sites.

d. If there is some circle through $p_i$ and $p_j$ that contains no other sites, then $(p_i, p_j)$ is an edge of Delaunay Triangulation of the point sites $P$.

e. The site corresponding to an unbounded Voronoi region exactly corresponds to the vertices of the convex hull of point sites.

f. The nearest neighbor of site $p_i$ is one of the sites corresponding to cells incident on $V(i)$. This property is handy to compute the nearest neighbor of a site efficiently.

g. The minimum spanning tree of point sites $S$ is a subset of the Delaunay triangulation of $S$.

Applications of Voronoi diagram: Voronoi diagrams have been used in many fields such as computer graphics, pattern recognition, geographic information systems, robotics, etc.
Some widely known applications include the following.

a. The analysis of structures of complex shapes is done by finite element methods. In a finite element method it is necessary to partition the domain into simple shapes such as triangles and quadrilaterals. The dual of Voronoi diagram, the Delaunay triangulation, can be used to generate quality mesh in finite element methods.

b. The minimum spanning tree (MST) of a set of points is a minimum length tree that spans all the points. One of the applications of MST is in network topology where it minimizes the total wire length, which in turn minimizes both costs and time delays. It is known that MST is contained in Delaunay triangulation (DT) i.e.

\[ MST \subseteq DT \]

To compute MST of point sites in two dimensions, we can start with the Delaunay triangulation as the starting graph rather than the complete graph. Then any standard MST algorithm (Prim's or Kruskal's) is applied on DT. Since the number of edges of DT are linearly related to the number of vertices in DT, the MST algorithm applied on DT has faster execution time.

c. The path that visits every vertex in a given graph is called the traveling salesperson path (TSP). The TSP problem is known to be NP-hard. Delaunay triangulation can be used to find an approximate solution for TSP. In this approach, minimum spanning tree \( T_m \) of DT is computed first. Then the Euler tour of \( T_m \) can be taken as an approximate solution for TSP. In fact the length of the path obtained in this way is no more than twice the length of the optimal TSP tour.

2.3 Relative Neighborhood Graph

The Relative Neighborhood Graph (RNG) [34] of a set of point sites \( p_1, p_2, \ldots, p_n \) is a graph whose nodes correspond to the point sites, and two nodes \( p_i \) and \( p_j \) are connected by an edge if and only if they are at least as close to each other as to any other point. Mathematically, \( p_i \) and \( p_j \) are connected by an edge if the following condition is satisfied.
\[ d(p_i, p_j) \leq \max_{m \neq i,j} \{d(p_i, p_m), d(p_j, p_m)\} \]

where \( d(p_i, p_j) \) is the Euclidean distance between \( p_i \) and \( p_j \).

A more intuitive definition of RNG can be presented in terms of the lune of influence. Consider the circles centered at \( p_i \) and \( p_j \) each with radius \( d(p_i, p_j) \). The intersection of these two circles is called the lune of influence (denoted as Lune\((p_i, p_j)\)) induced by \( p_i \) and \( p_j \). The shaded area in the Figure 2.3 illustrates the lune of influence induced by \( p_i \) and \( p_j \) if Lune\((p_i, p_j)\) does not contain any other point site.

![Figure 2.3: Illustrating Lune of Influence](image)

Several interesting properties of RNG have been explored by Computational Geometers[32].

**Property 1:** Minimum Spanning Tree (MST) is a subgraph of RNG.

**Property 2:** RNG is a subgraph of Delaunay Triangulation (D). This leads to the following relation.

\[ MST \subseteq RNG \subseteq DT \]

Figure 2.4 shows the MST, RNG and DT of a set of point sites in two dimensions. Several interesting applications of RNG in pattern recognition have been reported by Toussaint[32]. In recent years RNG structures has been found to be very useful for building underlying
network in mobile computing[31,33].

2.4 Nearest Neighborhood Graphs

Another proximity structure in two dimensions proposed by computational geometers [32] is the Nearest Neighborhood Graph (NNG) of point sites. Nearest Neighborhood Graph (NNG) for a set of points \( P \) is defined as follows:

A point \( b \) is the nearest neighbor of point \( a \) iff \(|a - b| < \min_{c \neq a} |a - c| \), where \( c \in P \).

To find a nearest neighbor, all we have to do is find the Voronoi diagram in \( O(n \log n) \) time and find the cell in which the querying point is located. The nearest neighbor graph (NNG) is a graph in which there is an edge between the nodes if one point is nearest neighbor of the other. In essence \( NNG \subseteq DT \). Figure 2.5 is an example of the nearest neighbor graph.

The fact that \( NNG \) is a subset of Delaunay triangulation can be used to develop an efficient algorithm for computing the nearest neighborhood graph. First we find the Delaunay triangulation \( DT \) of point sites and then apply the nearest neighborhood check for vertex pair of edges. If the vertex pair of a Delaunay edge \( e = (a, b) \) also satisfies the nearest neighborhood property then \( e \) is also taken as an edge of \( NNG \). Otherwise, it is discarded. Thus \( NNG \) can be computed in \( O(n \log n) \) time.

2.5 Gabriel Graphs

The proximity structure of a set of points can also be represented by using a Gabriel Graph. Such graphs were introduced in relation to geographic variation analysis [32]. In recent years Gabriel Graphs have been investigated for application in Mobile Computing [31,33]. The intuitive definition of a Gabriel Graph is that there is an edge between two vertices \( a \) and \( b \) if the circle with \((a, b)\) as diameter has no points inside it. Figure 2.6 illustrates the Gabriel graph. In Figure 2.6, edge \((a, b)\) is part of the Gabriel Graph. Edge \((b, c)\) is not an edge of the Gabriel Graph because there is a point inside that circle with \((b, c)\) as diameter. The empty circle test property of Gabriel Graphs can be used to show that a Gabriel Graph is a subset of Delaunay triangulation. It is also known that vertices \( p \) and \( q \) are adjacent in the Gabriel Graph if and only if the Delaunay edge between \( p \)
and $q$ intersects its dual Voronoi edge. Since Delaunay triangulation can be computed in $O(n \log n)$ time, the stated properties of Gabriel Graph can be used in straightforward way to compute Gabriel Graph in $O(n \log n)$ time.
Figure 2.4: Different proximity structures for a set of 10 points
Figure 2.5: Nearest Neighborhood Graph

Figure 2.6: Illustrating Gabriel Graph

(a): Edge Existence Condition

(b): Gabriel Graph
CHAPTER 3
BOUNDING CIRCLE AND FACILITY LOCATION

3.1 Problem Definition

Consider a set of customer sites in a region such as a county or city. We are required to determine the location of facilities such as hospitals, distribution centers, or fire stations so that the customers can reach the facilities with minimum effort. Customers can have better access to facilities by reducing the total travel time to facilities. The general version of the facility location problem is the determination of locations for the facilities so that the customers can access the facilities with minimum effort. In one simplified version of the facility location problem the customer sites are represented by points in two dimensions and the objective is to minimize the distance to the furthest customer. This can be formally stated as follows:

Single Facility Location Problem (SFLP1)

Given: Point sites $c_1, c_2, \ldots, c_n$ distributed in the plane.

Question: Find the point $X$ in the plane so that the distance of the farthest customer to $X$ is minimized.

This version of the single facility location problem (SFLP1) has a nice geometrical representation. The optimum location of the facility exactly corresponds to the center of the smallest circle enclosing the customer sites. This illustrated in the Figure 3.1. This problem is also known as the Euclidean 1-Center Problem in the computational geometry community[8].
Figure 3.1: Smallest Enclosing Circle for 10 customer sites

3.2 Algorithms for Smallest Bounding Circle

Algorithms for computing the smallest enclosing circle for a set of \( n \) point sites have been considered by several authors\[24,27,32\]. A straightforward algorithm can be developed by noting the observation that three customer sites uniquely determine a circle. So the idea is to consider all combinations of three customer sites and construct corresponding circles. A circle constructed in this way is called feasible if it encloses all customer sites. The smallest circle among all feasible circles gives the required circle. Whether a circle is feasible or not can be determined in \( O(n) \) time. Since there are \( O(n^3) \) circles the straightforward approach takes \( O(n^4) \) time. Interesting variations of the problem of computing smallest enclosing problem is described in [24].

A significantly faster algorithm for computing smallest enclosing circle was proposed by Shamos, Hoey and Preparata\[32\]. This algorithm runs in \( O(n \log n) \) time and is based on the computation of Furthest Point Voronoi Diagram (FPVD).

To describe this algorithm we can start with the motion of the furthest point voronoi diagram induced by a set of \( n \) point sites.

Consider \( n \) point sites \( s_1, s_2, \ldots, s_n \) in the plane. Site \( s_i \) is the furthest neighbor of a point \( p \) in the plane if \( s_i \) is the site at largest distance from \( p \). The set of points \( p \) in the plane for which \( s_i \) is the furthest neighbor is the furthest Voronoi region induced by \( s_i \). Let \( FV(i) \) denote the furthest point Voronoi region induced by site \( s_i \). Figure 3.2 shows the furthest
It may be noted that the furthest region of some sites are empty. It can be easily seen that from a vertex \( v_j \) of furthest point Voronoi diagram there are exactly three sites as its furthest neighbors. For example, from vertex \( v_x \) furthest sites are \( s_2, s_3, s_9 \). Furthermore, the three furthest neighbor sites of a vertex \( v_j \) define a circle that exactly encloses all \( n \) sites. We can construct enclosing circles corresponding to each vertex of the furthest point Voronoi diagram. The smallest among these circles gives the smallest enclosing circle. This is illustrated in Figure 3.3. The furthest point Voronoi diagram of \( n \) point sites can be determined in \( O(n \log n) \) time [32]. Hence this algorithm can be adopted to compute the smallest enclosing circle within the same enclosing circle within the same time complexity.

![Figure 3.2: Furthest Point Voronoi Diagram](image)

It may be observed that if the convex hull of \( n \) sides has \( h \) vertices then there will be exactly \( h \) regions in the furthest Voronoi diagram. The regions corresponding to sites inside the convex hull are empty.
The \(O(n \log n)\) algorithm based on furthest point Voronoi diagram is suitable for computing the smallest enclosing circle. An applet that computes smallest enclosing circle is found at www.cs.brown.edu/people/tor/java/mec.

Although the time complexity \(O(n \log n)\) is quite fast, it is not optimal. An optimal \(O(n)\) algorithm for computing smallest enclosing circle was reported by Megiddo[27]. This algorithm is based on prune and search paradigm. The algorithm iterates \(O(n)\) time and at each iteration eliminates a constant fraction of the input sites. Details are quite complicated. Megiddo's prune and search algorithm is very important theoretically (in fact, it is the only optimal algorithm) but is not easy to implement.

Figure 3.3: Furthest Point Voronoi Diagram with circles
3.3 Variations of Smallest Bounding Circle Problem

Although point facility is considered as an acceptable model for facility location, some researchers have considered facility locations modeled as geometric shapes with finite extension. Megiddo [27] was one of the early researchers to investigate the problem of facility location where the facility is constrained to lie in a line segment. Recently, Hortado, Sacristan, and Toussaint [24] have reported several efficient algorithms for facility location where facility is not necessarily a point. In particular, they have developed a linear time algorithm for the problem of constructing the minimum enclosing circle of a set of \( n \) points with the center constrained to satisfy \( m \) linear constraints. They also reported a linear time algorithm for the circle encloser problem when the center is constrained to lie in a \( m \)-vertex convex polygon.
CHAPTER 4

FACILITY LOCATION FOR WEIGHTED POINTS

4.1 Problem Definition

Consider a set of customer sites $c_1, c_2, \ldots, c_n$ in the two dimensional plane. Let the coordinate of customer $c_i$ be $(x_i, y_i)$. For each customer $c_i$ a weight $w_i$ is assigned which reflects the number of customers at $(x_i, y_i)$. The weight $w_i$ can also be used to represent other factors such as purchasing power, etc.

We want to locate a serving facility at a point $(x, y)$ such that the total travel cost for customers to travel to the facility is minimized. Various metrics have been explored to measure distance between customers and facilities in operation research literature [3]. Manhattan, $L_2$, and $L_\infty$ are examples of distance metrics considered in location theory research. In Manhattan metric, distance between two points $(x_i, y_i)$ and $(x_j, y_j)$ is given by the expression $|x_i - x_j| + |y_i - y_j|$. In $L_2$-metric (which is also known as the standard Euclidean metric), the distance is given by the expression $\sqrt{(x_k - x_i)^2 + (y_k - y_i)^2}$. Finally, in $L_\infty$-metric, distance is simply given by the square of Euclidean metric. It may be noted that Euclidean metric is the most widely used metric in facility location applications.

One of the key factors suitable for measuring the quality of a facility location is the cost of travel to the facility. One simple way to measure the travel cost is the total weighted distance of customers to the facility. Let $d(q, c_i)$ represent the Euclidean distance between a point $q$ and customer site $c_i$. If the facility is located at point $q$ then the total weighted distance to $q$, denoted by $WD(q)$ is given by

$$WD(q) = \sum_{i=1}^{n} d(q, c_i)w_i$$

Points that optimize the total weighted distance to customer sites are known as median.
points in location theory literature. The following is a formal definition of 1-median.

**1-Median:** A point \( q \) that minimizes the weighted distance \( WD(q) \) is called the 1-median of the set \( S = \{ c_1, c_2, \ldots, c_n \} \) [23,25]

### 4.2 Weiszfield’s Algorithm and Center of Gravity

Computing 1-median of a set of weighted points in two dimensions is a well known problem in location theory [23]. An elegant algorithm for computing 1-median of weighted points in two dimensions was initially proposed by Weiszfield[35]. Weiszfield’s algorithm uses steepest descent method to search the solution. The algorithm proceeds to determine the solution by making a convex combination of weighted points.

Weiszfield’s algorithm starts by picking a suitable point \((x_0', y_0')\) as the starting approximate solution for 1-median. It uses a convex combination of coordinates of customer sites. in terms of coefficients generated by the recently computed solution, to generate a new improved solution. Let \((x_k', y_k')\) denote the solution obtained at the \(k^{th}\) iteration. For each point \(p_i(x_i, y_i)\) the algorithm computes normalized weights \(g_i = \frac{w_i}{\sqrt{((x_k-x_i)^2+(y_k-y_i)^2)}}\). It next computes the coefficients \(l_i = \frac{g_i}{\sum_{i=1}^{n} g_i}\). The improved location \((x_{k+1}', y_{k+1}')\) is then obtained by taking a convex combination of weighted points given by:

\[
x_{k+1}' = l_1 x_1 + l_2 x_2 + \ldots + l_n x_n
\]
\[
y_{k+1}' = l_1 y_1 + l_2 y_2 + \ldots + l_n y_n
\]

For each approximate solution the algorithm computes the weighted distance to customer sites. If the improvement to solution is more than a certain threshold value then the algorithm iterates to compute the next improvement. Otherwise, the algorithm stops and reports the recently computed location as the required 1-median. A formal sketch of the algorithm as given below.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
**Weiszfeld’s Algorithm**

**Input:** Weighted point set \( S = \{ c_1, c_2, ..., c_n \} \) in two dimensional plane. Point \( c_i \) has weight \( w_i \) and coordinate \((x_i, y_i)\).

**Output:** Point \((x_i, y_i)\) representing the 1-median of set \( S \).

**Step 1:** Start with \((x_0, y_0)\) as the initial point. One can take center of gravity of \( S \) as \((x_0, y_0)\). Set \( k \) to 0.

**Step 2:** While \((k < \text{MAX} \text{ and improvement in solution} \leq \text{target value})\) {

a. For each point \( c_i \) compute \( g_i = w_i / (\sqrt{(x_k - x_i)^2 + (y_k - y_i)^2}) \)

b. \( g = g_1 + g_2 + ... + g_n \)

c. For each point \( c_i \) compute \( l_i = \frac{g_i}{g} \)

d. \( k = k + 1 \)

e. \( x_k = l_1 \cdot x_1 + l_2 \cdot x_2 + ... + l_n \cdot x_n \)

f. \( y_k = l_1 \cdot y_1 + l_2 \cdot y_2 + ... + l_n \cdot y_n \)

g. determine improvement in solution

}

The number of steps required to reach convergence for Weiszfeld’s algorithm depends on the desired accuracy in location result and the accuracy of representation of numbers. The time complexity of Weiszfeld’s algorithm can be written as \( O(nL) \), where \( L \) represents the factor for bit complexity of the representation of numbers and the desired accuracy for location computation. Weiszfeld’s algorithm always finds a solution within the numerical accuracy if enough iterations are made. The algorithm finds the solution for any starting point. A good starting point is the center of gravity of weighted points. It may be noted that the center of gravity \( p^* \) of weighted point set \( S \) is given by the expression:

\[
p^* = \frac{\sum_{i=1}^{n} \frac{x_i}{w_i} \cdot p_i \cdot w_i}{\sum_{i=1}^{n} w_i}
\]

At this point it is appropriate to compare 1-median and center of gravity\((cg)\). If points are more or less symmetrically distributed and weights are also more or less symmetrically distributed then the center of gravity and 1-median tend to be closer to each other. This is illustrated in Figure 4.1. On the other hand, if points and weights are distributed in a...
skewed way then 1-median and center of gravity are far apart. This is illustrated in Figure 4.2.

For randomly distributed points with randomly distributed weights the center of gravity and 1-median are not very far apart (Figure 4.3). There is a very interesting relationship between 1-median and center of gravity under $L^2_2$ metric. This stated in the following observation.

**Observation 1:** Under $L^2_2$ metric, i.e. when distance between $c_i$ and $c_j$ is taken as $(x_i - x_j)^2 + (x_i - x_j)^2$ then the center of gravity is the 1-median.
The notion of 1-median for Euclidean graphs have been introduced[23]. The nodes of the graph are assigned weights and the edges of the graph represent the path connecting the nodes. An edge $e$ connecting nodes $n_1$ and $n_2$ is assigned weight equal to the Euclidean distance between $n_1$ and $n_2$. It is known that 1-median of an Euclidean graph always lies on the nodes of the graph[23,25]. This property leads to a direct algorithm for computing 1-median of an Euclidean graph: compute the weighted distance from each node, and the node corresponding to the smallest weighted distance is reported as the required 1-median of the graph.

The notion of 1-median of an Euclidean graph can be generalized in a straightforward way to the notion of p-median. The p-median of an Euclidean graph $G(V,E)$ is the set of points that minimizes the total weighted distance to all the nodes. It is also known that the p-median also always lie on the nodes of the graph. However, finding the p-median of
a Euclidean graph is known to be NP-Hard[25].

A variation of the 1-median problem is the computation of a minimum spanning tree of an Euclidean graph constrained to have its median at a given node. This problem is called the Median Constrained Minimum Spanning Tree( MCMST ) problem. It was recently shown that MCMST is NP-Hard.

### 4.3 Multi Facility Location

In the multi-facility location problem, we are given a set of weighted points in two dimensions, and we need to find the locations to station $k$-facilities so that the total weighted distance from facilities to the respective customers is minimized. The problem can be formally stated as follows.

**Multi Facility Problem (MFP)**

*Given:* A set of weighted points $S = c_1, c_2, ..., c_n$, an integer $k$.

*Question:* Find $k$ locations $q_1, q_2, ..., q_k$ to station $k$ facilities that minimizes $\sum_{i=1}^{k} WD(q_i)$. where $WD(q_i)$ is the weighted distance from $q_i$ to the customers served by facility at $q_i$.

The set of $k$ locations that gives a solution for MFP is also called the $k$-median of weighted point set $S$. As mentioned earlier, MFP is known to be NP-Hard when the customer points are constrained to be vertices of an Euclidean graph[25].

### 4.4 Voronoi Driven k-Median Approximation

We now proceed to develop a heuristic for finding an approximate solution for multi-facility location problem by capturing properties of Voronoi diagrams of potential facility locations and the 1-medians of customer sites. To explore the underlying ingredients of the heuristic we start with a small running example. Consider the problem of locating four facilities in the presence of eighteen weighted customer sites in the plane as shown in Figure 4.4.

The Voronoi diagram induced by the facilities is drawn by dashed lines. Let $V(q_i^0)$ denote the Voronoi polygon corresponding to facility $q_i^0$. By the definition of the Voronoi diagram all the customer points in $V(q_i^0)$ are closer to $q_i^0$ than to other facilities. Hence for
this facilities configuration \((q_i^0, q_i^1, q_i^2)\), the total weighted distances is minimized if the customers in the region \(V(q_i^0)\) are served by facility located at \(q_i^0\). If we examine the customers in a Voronoi region \(V(q_i^0)\) and the corresponding location \(q_i^0\), we find that the location \(q_i^0\) is optimum if it corresponds to the 1-median of customers lying on the region \(V(q_i^0)\). However, \(q_i^0\) is not necessarily the 1-median of customers in \(V(q_i^0)\).

Consider computing the 1-median of customers in \(V(q_i^0)\). Let the 1-median of customers in \(V(q_i^0)\) be \(q_i^1\). This suggests that shifting the facility location from \(q_i^0\) to \(q_i^1\) could possibly reduce the total weighted distances. Of course when \(q_i^0\) is changed to \(q_i^1\) the corresponding Voronoi diagram of facility locations is different than the old Voronoi diagram. Figure 4.5 illustrates the Voronoi diagram of the new locations.

This strategy of generating possibly improved locations \(q_i^{i+1}\)'s from the 1-median of
Figure 4.5: Locations After One Iteration

$V(q_i^j)$'s is iterated to progressively search for better locations. The search for better locations can stop if incremental improvement in the quality of the generated solutions is less than a certain predefined value $\delta$. The quality of solution in each incremental step can be measured by the difference of the old and the new weighted distance. A formal sketch of this algorithm is listed below.

**Algorithm Voronoi Driven k-Median Approximation**

**Input:**
(a) Points $c_1, c_2, \ldots, c_n$ in 2-d representing customers. The weights and co-ordinates of point $c_i$ are $w_i$ and $(x_i, y_i)$, respectively.

(b) Integer $k$ denoting the number of facilities.

(c) $\delta$, desired accuracy. Limit, maximum iterations.
Output: Co-ordinates of $k$ locations where the facilities can be located.

Step1: (a) Pick suitable distinct points $q_1, q_2, ..., q_k$ as the initial values of $k$ locations.
      (b) Improvement = Large Number,
      (c) $\text{CumWt} = \text{Large Number}$
      (d) $j = 1$:

Step2: While ((improvement $> \delta$) and ($j < \text{Limit}$))
{

Step3: Find the Voronoi Diagram of $q_1, q_2, ..., q_k$.
        Let $V(q_i)$ be the Voronoi Region corresponding to $q_i$.

Step4: (a) $\text{NewCumWt} = 0$;
        (b) For $i = 1$ to $k$ do
            {
                (c) set $q_i$ to 1-median of sites in $V(q_i)$.
                (d) $W_t =$ weighted distances of sites in $V(q_i)$.
                (e) $\text{NewCumWt} = \text{NewCumWt} + W_t$;
            }
        (f) Improvement = $\text{CumWt} - \text{NewCumWt}$; $j = j + 1$;
        (g) $\text{CumWt} = \text{NewCumWt}$:
}

Step5: Output $q_1, q_2, ..., q_k$ as the required location.

Complexity Analysis: Like Weiszfeld's algorithm, the exact time complexity of the Voronoi driven $k$-median approximation algorithm depends on the numerical accuracy of the representation of numbers and the desired accuracy in the computation of locations. For each iteration of the outer loop the algorithm computes the Voronoi diagram of $k$ sites and 1-median of customers in each Voronoi region. The total time for Step 4 is $O(knL)$, where $L$ is the factor representing the numerical accuracy of representation. The time for
Step 4 dominates the time for Step 3 which is \(O(k \log k)\). Hence the total time can be written as \(O(knL^2)\).
CHAPTER 5

IMPLEMENTATION AND RESULTS

5.1 Introduction

In this chapter we briefly describe the implementation of the proposed k-mean approximation algorithm for multi-facility location problem. We make use of the Java class for Delaunay triangulation developed by O'Rourke and his group[32]. The Java source code for Delaunay triangulation can be downloaded from the site http://cs.smith.edu/orourke/books/ftp.html.

We develop our own class for Voronoi diagram by extending the class for Delaunay triangulation into a "suitable" data structure that allows the traversal of the structure by edges, faces and vertices efficiently. We also represent the Voronoi diagram in a similar "suitable" data structure. Since Voronoi diagrams and Delaunay triangulations are duals of each other, we also incorporate this dual relationship in the implemented structure. The main building blocks used for implementing the k-mean approximation algorithm are the Delaunay triangulation and Voronoi diagram of facility location points represented in a "special" data structure.

In the proposed k-mean approximation algorithm, the location of candidate facility sites (and hence their Voronoi diagram and Delaunay triangulation) change on each iteration of the search. This iteration continues until the incremental improvement is less than a threshold value. Hence to obtain an efficient implementation of the proposed algorithm, the faces of the Delaunay triangulation and the faces of the Voronoi diagram should be traversable with minimal overhead. Both Delaunay triangulation and Voronoi diagram are planar graphs and hence we need to look for an efficient data structure for representing such
graphs. We give a detailed explanation of the doubly connected edge list data structure used in the implementation of the Voronoi diagram and Delaunay triangulation.

In order to evaluate the proposed algorithm, we tested the algorithm on different sets of data. In particular, randomly generated customer sites of various sizes namely 50, 100, 250, 500 and 1000 are considered. The algorithm is also executed on distinct clustered customer sites. The number of facilities to create is taken as the number of clusters. The motivation behind using clustered customers is that the best solution for facility location is more or less known. The experimental results for these kinds of data are tabulated for measuring the performance of the proposed algorithm.

5.2 Data Structures

A Voronoi diagram is the partitioning of the plane into a planar graph consisting of edges, faces, and vertices. There are several data structures available for representing planar graphs:\[11,32]:

(i) Winged-edge data structure
(ii) Quad-edge data structure
(iii) Doubly connected edge list (DCEL) data structure

In our implementation, we use the doubly connected edge list data structure. Three components are needed to represent a Voronoi diagram by DCEL. These are (i) Vertex record, (ii) Edge record, and (iii) Face record. The Voronoi diagram of Figure 5.1 is represented in a DCEL structure as shown in Figure 5.2.

The Vertex Record: The Vertex record contains the coordinates of a vertex and also stores the information of an arbitrary edge incident on the vertex. Table 1: Figure 5.3 illustrates the records of all the vertices of the Voronoi diagram represented in the DCEL data structure of Figure 5.2.
(ii) **Edge Record**: The Edge record is an important part of the DCEL data structure. We observe that each edge bounds two faces, so an edge can be represented by a pair of half-edges directed in opposite directions as shown in Figure 5.2. These half-edges are twin-edges of each other. The record of each half-edge $e_i$ contains the following information.

(a) **Edge**: This is the index of the record of the half-edge $e_i$.

(b) **Starting Vertex**: Index of the vertex from which the half-edge $e_i$ starts.

(c) **Twin Edge**: Index of the record of the twin of $e_i$.

(d) **Previous Edge**: Consider the ordered list of half-edges bounding the face incident to $e_i$.

   Previous Edge is the index of the half-edge that occurs before $e_i$ in the ordered list.

(e) **Next Edge**: This gives the index of the half-edge that occurs after $e_i$ in the ordered list.

(f) **Incident Face**: Index of the face on which half-edge $e_i$ is incident upon.

Since each edge is a directed edge on the boundary of the incident face, we can distinguish the origin and the destination of an edge. Here the start of an edge is the end vertex of the previous edge and its end vertex becomes the start of the next edge. The edge records for
Figure 5.2 are shown in Table 2: Figure 5.4.

**Face Record:** The face record stores the index of a half-edge incident on it. Additional information related to face such as area or perimeter, etc. might be useful. In our implementation, Outer Bounding Edge gives the index of a half edge (any one) bounding the face. And Other Info gives the additional information associated with face as necessary for applications of interest.

The versatility of the DCEL data structure lies in the fact that it simplifies many operations and supports efficient implementation. Some examples are:

(i) **Computing the size of a face:** The size of a face can be computed by using the starting outer boundary edge in the face record. Then this half-edge is used to traverse around the face to get the size of the face. A straightforward algorithm based on this idea is:

```c
int getSizeOfFace( int boundingEdge )
{
...
```
Figure 5.3: Table 1: Vertex Records

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Coordinates</th>
<th>Incident Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>(50,20)</td>
<td>e31</td>
</tr>
<tr>
<td>v2</td>
<td>(50.59)</td>
<td>e24</td>
</tr>
<tr>
<td>v3</td>
<td>(50.178)</td>
<td>e25</td>
</tr>
<tr>
<td>v4</td>
<td>(50.199)</td>
<td>e26</td>
</tr>
<tr>
<td>v5</td>
<td>(235.199)</td>
<td>e27</td>
</tr>
<tr>
<td>v6</td>
<td>(235.177)</td>
<td>e28</td>
</tr>
<tr>
<td>v7</td>
<td>(235.20)</td>
<td>e29</td>
</tr>
<tr>
<td>v8</td>
<td>(217.20)</td>
<td>e30</td>
</tr>
<tr>
<td>v9</td>
<td>(73.78)</td>
<td>e5</td>
</tr>
<tr>
<td>v10</td>
<td>(94.149)</td>
<td>e20</td>
</tr>
<tr>
<td>v11</td>
<td>(173.140)</td>
<td>e14</td>
</tr>
<tr>
<td>v12</td>
<td>(174.66)</td>
<td>e9</td>
</tr>
</tbody>
</table>

Figure 5.4: Table 3: Face Records

<table>
<thead>
<tr>
<th>Face</th>
<th>Outer Bound Edge</th>
<th>Other Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>e8</td>
<td>w1</td>
</tr>
<tr>
<td>f2</td>
<td>e9</td>
<td>w2</td>
</tr>
<tr>
<td>f3</td>
<td>e18</td>
<td>w3</td>
</tr>
<tr>
<td>f4</td>
<td>e22</td>
<td>w4</td>
</tr>
<tr>
<td>f5</td>
<td>e0</td>
<td>w5</td>
</tr>
<tr>
<td>f6</td>
<td>e26</td>
<td>w6</td>
</tr>
</tbody>
</table>

The time complexity of the above algorithm is $O(m)$, where $m$ is the size of the face i.e., the number of edges bounding the face. For face f1, the bounding edge is $e_8$ and when traversed using this edge, the size of the face is 5.
<table>
<thead>
<tr>
<th>Edge</th>
<th>Starting Vertex</th>
<th>Twin Edge</th>
<th>Previous Edge</th>
<th>Next Edge</th>
<th>Incident Face</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_0$</td>
<td>$v_{10}$</td>
<td>$e_{23}$</td>
<td>$e_1$</td>
<td>$e_3$</td>
<td>$f_3$</td>
</tr>
<tr>
<td>$e_1$</td>
<td>$v_{11}$</td>
<td>$e_{19}$</td>
<td>$e_2$</td>
<td>$e_0$</td>
<td>$f_3$</td>
</tr>
<tr>
<td>$e_2$</td>
<td>$v_{12}$</td>
<td>$e_{13}$</td>
<td>$e_3$</td>
<td>$e_1$</td>
<td>$f_3$</td>
</tr>
<tr>
<td>$e_3$</td>
<td>$v_9$</td>
<td>$e_4$</td>
<td>$e_0$</td>
<td>$e_2$</td>
<td>$f_3$</td>
</tr>
<tr>
<td>$e_4$</td>
<td>$v_{12}$</td>
<td>$e_3$</td>
<td>$e_8$</td>
<td>$e_5$</td>
<td>$f_1$</td>
</tr>
<tr>
<td>$e_5$</td>
<td>$v_9$</td>
<td>$e_{22}$</td>
<td>$e_4$</td>
<td>$e_6$</td>
<td>$f_1$</td>
</tr>
<tr>
<td>$e_6$</td>
<td>$v_2$</td>
<td>$e_{31}$</td>
<td>$e_5$</td>
<td>$e_7$</td>
<td>$f_1$</td>
</tr>
<tr>
<td>$e_7$</td>
<td>$v_1$</td>
<td>$e_{30}$</td>
<td>$e_6$</td>
<td>$e_8$</td>
<td>$f_1$</td>
</tr>
<tr>
<td>$e_8$</td>
<td>$v_8$</td>
<td>$e_9$</td>
<td>$e_7$</td>
<td>$e_4$</td>
<td>$f_1$</td>
</tr>
<tr>
<td>$e_9$</td>
<td>$v_{12}$</td>
<td>$e_8$</td>
<td>$e_{13}$</td>
<td>$e_{10}$</td>
<td>$f_2$</td>
</tr>
<tr>
<td>$e_{10}$</td>
<td>$v_8$</td>
<td>$e_{29}$</td>
<td>$e_9$</td>
<td>$e_{11}$</td>
<td>$f_2$</td>
</tr>
<tr>
<td>$e_{11}$</td>
<td>$v_7$</td>
<td>$e_{28}$</td>
<td>$e_{10}$</td>
<td>$e_{12}$</td>
<td>$f_2$</td>
</tr>
<tr>
<td>$e_{12}$</td>
<td>$v_6$</td>
<td>$e_{14}$</td>
<td>$e_{11}$</td>
<td>$e_{13}$</td>
<td>$f_2$</td>
</tr>
<tr>
<td>$e_{13}$</td>
<td>$v_{11}$</td>
<td>$e_2$</td>
<td>$e_{12}$</td>
<td>$e_9$</td>
<td>$f_2$</td>
</tr>
<tr>
<td>$e_{14}$</td>
<td>$v_{11}$</td>
<td>$e_{12}$</td>
<td>$e_{19}$</td>
<td>$e_{15}$</td>
<td>$f_3$</td>
</tr>
<tr>
<td>$e_{15}$</td>
<td>$v_6$</td>
<td>$e_{27}$</td>
<td>$e_{14}$</td>
<td>$e_{16}$</td>
<td>$f_3$</td>
</tr>
<tr>
<td>$e_{16}$</td>
<td>$v_5$</td>
<td>$e_{26}$</td>
<td>$e_{15}$</td>
<td>$e_{17}$</td>
<td>$f_3$</td>
</tr>
<tr>
<td>$e_{17}$</td>
<td>$v_4$</td>
<td>$e_{25}$</td>
<td>$e_{16}$</td>
<td>$e_{18}$</td>
<td>$f_3$</td>
</tr>
<tr>
<td>$e_{18}$</td>
<td>$v_3$</td>
<td>$e_{20}$</td>
<td>$e_{17}$</td>
<td>$e_{19}$</td>
<td>$f_3$</td>
</tr>
<tr>
<td>$e_{19}$</td>
<td>$v_{10}$</td>
<td>$e_1$</td>
<td>$e_{18}$</td>
<td>$e_{14}$</td>
<td>$f_3$</td>
</tr>
<tr>
<td>$e_{20}$</td>
<td>$v_{10}$</td>
<td>$e_{18}$</td>
<td>$e_{23}$</td>
<td>$e_{21}$</td>
<td>$f_4$</td>
</tr>
<tr>
<td>$e_{21}$</td>
<td>$v_3$</td>
<td>$e_{24}$</td>
<td>$e_{20}$</td>
<td>$e_{22}$</td>
<td>$f_4$</td>
</tr>
<tr>
<td>$e_{22}$</td>
<td>$v_2$</td>
<td>$e_5$</td>
<td>$e_{21}$</td>
<td>$e_{23}$</td>
<td>$f_4$</td>
</tr>
<tr>
<td>$e_{23}$</td>
<td>$v_9$</td>
<td>$e_0$</td>
<td>$e_{22}$</td>
<td>$e_{20}$</td>
<td>$f_4$</td>
</tr>
<tr>
<td>$e_{24}$</td>
<td>$v_2$</td>
<td>$e_{21}$</td>
<td>$e_{31}$</td>
<td>$e_{25}$</td>
<td>$f_6$</td>
</tr>
<tr>
<td>$e_{25}$</td>
<td>$v_3$</td>
<td>$e_{17}$</td>
<td>$e_{24}$</td>
<td>$e_{26}$</td>
<td>$f_6$</td>
</tr>
<tr>
<td>$e_{26}$</td>
<td>$v_4$</td>
<td>$e_{16}$</td>
<td>$e_{25}$</td>
<td>$e_{27}$</td>
<td>$f_6$</td>
</tr>
<tr>
<td>$e_{27}$</td>
<td>$v_5$</td>
<td>$e_{15}$</td>
<td>$e_{26}$</td>
<td>$e_{28}$</td>
<td>$f_6$</td>
</tr>
<tr>
<td>$e_{28}$</td>
<td>$v_6$</td>
<td>$e_{11}$</td>
<td>$e_{27}$</td>
<td>$e_{29}$</td>
<td>$f_6$</td>
</tr>
<tr>
<td>$e_{29}$</td>
<td>$v_7$</td>
<td>$e_{10}$</td>
<td>$e_{28}$</td>
<td>$e_{30}$</td>
<td>$f_6$</td>
</tr>
<tr>
<td>$e_{30}$</td>
<td>$v_8$</td>
<td>$e_7$</td>
<td>$e_{29}$</td>
<td>$e_{31}$</td>
<td>$f_6$</td>
</tr>
<tr>
<td>$e_{31}$</td>
<td>$v_1$</td>
<td>$e_6$</td>
<td>$e_{30}$</td>
<td>$e_{24}$</td>
<td>$f_6$</td>
</tr>
</tbody>
</table>

Figure 5.5: Table 2: Edge Records
(ii) **The Degree of a Vertex**: The degree of a vertex can be determined by finding the incident edge on the vertex from the Vertex records. This is then used to find the degree of the vertex. An algorithm for this given below:

```c
int getVertexDegree( int vertex )
{
    int count = 1, incidentEdge, newEdge;
    incidentEdge = getIncidentEdge( vertex );
    newEdge = incidentEdge;
    while( incidentEdge != getNextEdge( newEdge ) )
    {
        newEdge = getTwinEdge( newEdge );
        count++;  
        newEdge = getNextEdge( newEdge );
    }
return count;
}
```

The time complexity of the algorithm is \( O(m) \), where \( m \) is the number of edges incident upon the vertex.

(iii) **Nearest Neighbors**: We can find the nearest neighbor for a site very quickly. Here observe that every face has a site in it. So the nearest neighbor is the site with the minimum distance from the site in question. The algorithm for this is as follows:

```c
int getNearestNeighbor(int face)
{
Point p1,p2;
int bEdge, minIndex, neighbor;
double dist, minDistance = large value;
p1 = getFaceVertex(face);
bEdge = getBoundingEdge(face);
```
while( bEdge != getNextEdge( bEdge ) )
{
    neighbor = getFace( getTwinEdge(bEdge) );
    p2 = getFaceVertex( neighbor );
    dist = getDist( face , neighbor);
    if( dist < minDistance )
    {
        minIndex = neighbour;
        minDistance = dist;
    }
    bEdge = getNextEdge( bEdge );
}
return minIndex;

The time complexity of the above algorithm is $O(m)$, where $m$ is the size of the face.

5.3 Code Development

The main classes developed to implement the approximation algorithm are: (i) DCEL. (ii) LocCanvas. (iii) DELAUNAY, (iv) WebLocTestMultiMedian, and (v) LocCanvas. We use the Delaunay triangulation class called DelaunayTri developed by O'Rourke and his group to obtain the initial triangulation of point sites. The Java Delaunay triangulation code developed by O'Rourke and his group presents the output as a list of vertices and triangles. These triangles are not available in the form that allows traversal of faces and edges. We take the output of DelaunayTri class and represent it in doubly connected edge list form that allows the user to navigate the triangulation easily by faces, edges, and vertices. The class DELAUNAY accepts data type DelaunayTri and converts it into doubly connected edge list form.

The class DCEL has an extensive set of members that include the computation of
Voronoi diagrams. In this class both Voronoi diagram and Delaunay triangulation are represented in doubly connected edge list form. Since Voronoi diagram is the dual of the Delaunay triangulation, the class DCEL maintains this relationship so that the user of the class can navigate back and forth between both structures. In addition, the class contains member functions to display both Voronoi diagrams and Delaunay triangulations. Other member functions of DCEL include (i) computation of face area, (ii) listing of customer sites by Voronoi region, (iii) computation of face size etc.

The LocCanvas class is designed as a GUI interface to execute and display the computed facility locations. The interface allows the user to enter data manually by mouse click or from a file. The input data can be entered to the program by cut and paste of text file.

Finally, the class WebLocTestMultiMedian works as a driver program to have LocCanvas build a GUI and prompt the user for computation of multi facility locations. Below are some of the important members of the main classes of the implementation.

**Listing of Class Members**

**WebLoc_MultiMedian.java**

```java
WebLoc_MultiMedian()
void actionPerformed(ActionEvent e)
void main(String arg[])
```

**WebLocTest_MultiMedian.java**

```java
WebLocTest_MultiMedian()
void actionPerformed(ActionEvent e)
```

**LocCanvas.java**

```java
LocCanvas(TextArea tArea, Button bt, Checkbox cBox1, TextField tField)
void compDelaunay()
int checkEmptyVoronoi(int facIndex, int fNum[], int xC[], int yC[], int nC)
void computeWtDis(int findex, int fnum[], int xC[], int yC[], int wC[], PrintWriter pw, boolean ch)
```
void drawCustomers(Graphics g)
cPointi oneMedianByCG(int x[], int y[], int w[], int faceIndex, int nC)
cPointi oneMedianByWfld(int x[], int y[], int w[], cPointi currSol)
void clear()
void refreshCanvas()
void refreshTextArea()
int getIndexOfClosestPoint(Point p1)
void replaceIthPoint(Point p1, int index)
void mousePressed(MouseEvent e)
void mouseMoved(MouseEvent e)
void mouseDragged(MouseEvent e)
void chooseColor(Graphics g, int i)

DCEL.java

DCEL( int x[], int y[], int numOfVertices, int startIndex[], int endIndex[], int numEdges,
int faceSize[], int numOfFaces, int w1[])
void printDcelDelaunay()
void printString(String s, int i)
void drawDelaunay(Graphics g)
void drawFace(Graphics g, int faceIndex)
void drawCenter(Graphics g, int faceIndex)
void drawSites(Graphics g)
void drawMidPoints(Graphics g)
void computeVoronoi()
void computeVorFace(int vIndex)
void printVoronoi()
void drawVorPoly(Graphics g, int vIndex)
void drawVor(Graphics g)
void fillVorPoly(Graphics g, int fl)
void fillDelaunay(Graphics g, int fl)
void chooseColor(Graphics g, int i)
double polyArea(int x[], int y[])
double faceArea(int faceIndex)
void fillMaxFace(Graphics g)
void getComponent(int x[], int y[], int n, int compNum, int x1[], int y1[], int n1)
void test(Vector V1[])
void getCustomersByVoronoiRegion(int xC[], int yC[], int fNum[], int nC, int numOfFaces)

Delaunay.java

DELAUNAY(int xx[], int yy[], int N)
void setFaceSize(int size1[])

5.4 Experimental Results

In order to evaluate the performance of the proposed k-median approximation algorithm, we executed the algorithm on various input data of different sizes. The k-median approximation algorithm was executed on several randomly generated customers of sizes 50-1000 and randomly generated weights of sizes 1-8. We recorded the number of steps needed to reach the required convergence and calculated the weighted distance for the resulting solutions.

NOTE: The algorithm assumes that there will be at least one customer point where the facility has to be located.
5.4.1 Customer sites distributed randomly:

We use the standard random number generation facility provided by the Java programming language. We first generate a random integer \( p_x \) between 1 and \( X_{\text{max}} \). Specifically, this is done as \( p_x = 1 + (\text{int})(X_{\text{max}} \times \text{Math.random}) \). Similarly, \( p_y = 1 + (\text{int})(Y_{\text{max}} \times \text{Math.random}) \).

Then the pair \( (p_x, p_y) \) gives the coordinate of a randomly generated point in the rectangle \((1, 1) \times (X_{\text{max}}, Y_{\text{max}})\). We executed the k-median approximation algorithm for randomly generated customer sites of size 50, 100, 250, 500 and 1000. For each customer size we ran the algorithm to locate 7 facilities. In the first experiment we started with 7 "suitably" chosen points as the initial location of the facilities. To measure the quality of improvement in generated solution, the total weighted distance of customers in the corresponding facility is computed. The Sum of weighted distances are computed for both initial facility location and for the facility location generated by the algorithm. Furthermore, for each execution the number of steps required to reach the final solution is recorded. These results are tabulated in Table 4 in Figure 5.9, and a snap-shot of program execution is shown in Figure 5.6, 5.7 and 5.8.

5.4.2 Clustered Customer Sites

To better evaluate the performance of the proposed algorithm, it would be better to execute it on customer sites for which the optimal location of facilities is more or less known. If customer sites are in \( k \) distinct clusters and we need to locate \( k \) facilities then the obvious solution would be the center of clusters. We therefore constructed several clustered customer sites and executed the algorithms by picking arbitrary initial customer locations. When the algorithm was executed on clustered customers of varying sizes the generated solution always progressed towards the center of the cluster. The values of weighted distance and the number of steps needed to reach the convergence is tabulated in Figure 5.11. A snap shot of the output is shown in Figure 5.10.
The different 7 initial facility locations used in the examples are as follows

Type A
\[ x = ( 213.278,295.214,163.249,257 ) \]
\[ y = ( 50.37.88,99,51,81,133 ) \]
\[ w = ( 1, 2, 1, 3, 2, 3 ) \]

Type B
\[ x = (121.254,359,69,161,315,304) \]
\[ y = (125.99,223,248,357,349,147) \]
\[ w = (1, 2, 1, 3, 2, 3) \]

Type C
\[ x = (77.368,402,63,130,295) \]
\[ y = (65,66,378,380,295,211,127) \]
\[ w = (1, 2, 1, 3, 2, 3) \]

Type D
\[ x = (88,148,95,388,411,322,367) \]
\[ y = (77,49,138,275,318,307,361) \]
\[ w = (1, 2, 1, 3, 2, 3) \]

Type E
\[ x = (113,283,63,323,133,238,188) \]
\[ y = (72,74,222,227,325,324,161) \]
\[ w = (1, 2, 1, 3, 2, 3) \]
where \( x, y \) are the coordinates and \( w \) is the weight of the facility site.

The interpretation and discussions of the results exhibited in the Tables, Graphs, and Snap-Shots are described in Chapter 6.

![Initial Facility Location](image)

Figure 5.6: Initial Facility Location
Figure 5.7: Facility Location after one round of execution
Figure 5.8: Facility Location after final round of execution
<table>
<thead>
<tr>
<th>S.No.</th>
<th>Site Type</th>
<th>No. of Points</th>
<th>Wt. Dist Before</th>
<th>Wt. Dist After CG</th>
<th>Convergence Steps</th>
<th>Wt. Dist After Wz Conv. Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>1000</td>
<td>598688</td>
<td>232672</td>
<td>25</td>
<td>238865</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>1000</td>
<td>265477</td>
<td>231454</td>
<td>17</td>
<td>236941</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>1000</td>
<td>257977</td>
<td>231519</td>
<td>26</td>
<td>237602</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>1000</td>
<td>402895</td>
<td>231951</td>
<td>27</td>
<td>236041</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
<td>1000</td>
<td>282266</td>
<td>233700</td>
<td>17</td>
<td>233111</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>500</td>
<td>292568</td>
<td>110345</td>
<td>27</td>
<td>112451</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>500</td>
<td>132564</td>
<td>110123</td>
<td>12</td>
<td>113320</td>
</tr>
<tr>
<td>8</td>
<td>C</td>
<td>500</td>
<td>126950</td>
<td>113459</td>
<td>17</td>
<td>116775</td>
</tr>
<tr>
<td>9</td>
<td>D</td>
<td>500</td>
<td>196619</td>
<td>110318</td>
<td>20</td>
<td>115316</td>
</tr>
<tr>
<td>10</td>
<td>E</td>
<td>500</td>
<td>141834</td>
<td>115624</td>
<td>14</td>
<td>116614</td>
</tr>
<tr>
<td>11</td>
<td>A</td>
<td>250</td>
<td>145028</td>
<td>56029</td>
<td>13</td>
<td>58374</td>
</tr>
<tr>
<td>12</td>
<td>B</td>
<td>250</td>
<td>63682</td>
<td>54867</td>
<td>16</td>
<td>56954</td>
</tr>
<tr>
<td>13</td>
<td>C</td>
<td>250</td>
<td>63725</td>
<td>54971</td>
<td>11</td>
<td>58045</td>
</tr>
<tr>
<td>14</td>
<td>D</td>
<td>250</td>
<td>96738</td>
<td>56521</td>
<td>21</td>
<td>56282</td>
</tr>
<tr>
<td>15</td>
<td>E</td>
<td>250</td>
<td>70812</td>
<td>55092</td>
<td>11</td>
<td>58194</td>
</tr>
<tr>
<td>16</td>
<td>A</td>
<td>100</td>
<td>55342</td>
<td>22685</td>
<td>12</td>
<td>23012</td>
</tr>
<tr>
<td>17</td>
<td>B</td>
<td>100</td>
<td>25210</td>
<td>21527</td>
<td>8</td>
<td>24422</td>
</tr>
<tr>
<td>18</td>
<td>C</td>
<td>100</td>
<td>28914</td>
<td>23527</td>
<td>9</td>
<td>25222</td>
</tr>
<tr>
<td>19</td>
<td>D</td>
<td>100</td>
<td>37161</td>
<td>23829</td>
<td>6</td>
<td>25234</td>
</tr>
<tr>
<td>20</td>
<td>E</td>
<td>100</td>
<td>27416</td>
<td>22539</td>
<td>8</td>
<td>24544</td>
</tr>
<tr>
<td>21</td>
<td>A</td>
<td>50</td>
<td>29527</td>
<td>11653</td>
<td>7</td>
<td>11913</td>
</tr>
<tr>
<td>22</td>
<td>B</td>
<td>50</td>
<td>14055</td>
<td>10527</td>
<td>6</td>
<td>11497</td>
</tr>
<tr>
<td>23</td>
<td>C</td>
<td>50</td>
<td>14186</td>
<td>10214</td>
<td>5</td>
<td>12392</td>
</tr>
<tr>
<td>24</td>
<td>D</td>
<td>50</td>
<td>19038</td>
<td>10855</td>
<td>7</td>
<td>12034</td>
</tr>
<tr>
<td>25</td>
<td>E</td>
<td>50</td>
<td>14762</td>
<td>10613</td>
<td>6</td>
<td>13644</td>
</tr>
</tbody>
</table>

Figure 5.9: Table 4: Results for Customers Randomly Distributed in Rectangular Region
Figure 5.10: Snapshot for clustered customer sites
<table>
<thead>
<tr>
<th>S.No.</th>
<th>Site Type</th>
<th>No. of Points</th>
<th>Wt. Dist Before</th>
<th>Wt. Dist After CG</th>
<th>Convergence Steps</th>
<th>Wt. Dist. After Wz Conv. Steps</th>
<th>Conv. Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>1000</td>
<td>605932</td>
<td>195305</td>
<td>13</td>
<td>196261</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>1000</td>
<td>326281</td>
<td>174235</td>
<td>14</td>
<td>167629</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>1000</td>
<td>323821</td>
<td>195189</td>
<td>12</td>
<td>203099</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>1000</td>
<td>491206</td>
<td>190788</td>
<td>23</td>
<td>171735</td>
<td>28</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
<td>1000</td>
<td>251639</td>
<td>181112</td>
<td>11</td>
<td>184951</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>500</td>
<td>384509</td>
<td>128624</td>
<td>5</td>
<td>106353</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>500</td>
<td>180891</td>
<td>67528</td>
<td>10</td>
<td>80344</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>C</td>
<td>500</td>
<td>194112</td>
<td>72368</td>
<td>7</td>
<td>103980</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>D</td>
<td>500</td>
<td>211548</td>
<td>89880</td>
<td>11</td>
<td>89090</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>E</td>
<td>500</td>
<td>138378</td>
<td>53117</td>
<td>6</td>
<td>54020</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>A</td>
<td>250</td>
<td>194094</td>
<td>61101</td>
<td>5</td>
<td>39142</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td>B</td>
<td>250</td>
<td>98770</td>
<td>18125</td>
<td>10</td>
<td>30348</td>
<td>8</td>
</tr>
<tr>
<td>13</td>
<td>C</td>
<td>250</td>
<td>79757</td>
<td>17993</td>
<td>6</td>
<td>26596</td>
<td>6</td>
</tr>
<tr>
<td>14</td>
<td>D</td>
<td>250</td>
<td>110806</td>
<td>17452</td>
<td>7</td>
<td>51978</td>
<td>6</td>
</tr>
<tr>
<td>15</td>
<td>E</td>
<td>250</td>
<td>84443</td>
<td>17000</td>
<td>8</td>
<td>18251</td>
<td>9</td>
</tr>
<tr>
<td>16</td>
<td>A</td>
<td>100</td>
<td>93099</td>
<td>41757</td>
<td>5</td>
<td>41322</td>
<td>5</td>
</tr>
<tr>
<td>17</td>
<td>B</td>
<td>100</td>
<td>22296</td>
<td>7875</td>
<td>4</td>
<td>7971</td>
<td>5</td>
</tr>
<tr>
<td>18</td>
<td>C</td>
<td>100</td>
<td>37131</td>
<td>6132</td>
<td>6</td>
<td>6158</td>
<td>7</td>
</tr>
<tr>
<td>19</td>
<td>D</td>
<td>100</td>
<td>46307</td>
<td>7731</td>
<td>5</td>
<td>7826</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>E</td>
<td>100</td>
<td>20730</td>
<td>6756</td>
<td>6</td>
<td>6868</td>
<td>6</td>
</tr>
<tr>
<td>21</td>
<td>A</td>
<td>50</td>
<td>43600</td>
<td>4578</td>
<td>5</td>
<td>4668</td>
<td>5</td>
</tr>
<tr>
<td>22</td>
<td>B</td>
<td>50</td>
<td>11427</td>
<td>4548</td>
<td>4</td>
<td>4676</td>
<td>5</td>
</tr>
<tr>
<td>23</td>
<td>C</td>
<td>50</td>
<td>28316</td>
<td>3683</td>
<td>4</td>
<td>3603</td>
<td>13</td>
</tr>
<tr>
<td>24</td>
<td>D</td>
<td>50</td>
<td>7227</td>
<td>4507</td>
<td>4</td>
<td>4581</td>
<td>5</td>
</tr>
<tr>
<td>25</td>
<td>E</td>
<td>50</td>
<td>14666</td>
<td>3508</td>
<td>6</td>
<td>3697</td>
<td>7</td>
</tr>
</tbody>
</table>

Figure 5.11: Table 5: Results for Distinctly Clustered Customers
Figure 5.12: Weighted Distance versus No. of Customers for Randomized Data
Figure 5.13: Weighted Distance versus No. of Customers for Clustered Data
Figure 5.14: Convergence Steps versus No. of Customers for Clustered Data
Figure 5.15: Convergence Steps versus No. of Customers for Randomized Data

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
CHAPTER 6

CONCLUSION

We have presented a critical review of the facility location algorithms which have been developed by using tools from computational geometry.

We proposed a Voronoi based $k$-median approximation algorithm to compute locations to station $k$ facilities in the presence of $n$ weighted customer points. The proposed approximation algorithm is a searching algorithm that is driven by the iterative computation of Voronoi diagram and 1-medians of the initially selected $k$ locations. At each iteration the previous locations are improved. The iteration process terminates if the incremental improvement in the solution is less than a certain preselected threshold value. The algorithm is fairly simple to describe and understand and can be implemented without much difficulty.

We implemented the proposed algorithm in the JAVA programming language using a doubly connected edge list (DCEL) data structure to represent the Voronoi diagram of candidate facility locations. We executed the algorithm on several test data that include (i) randomly generated customer points in a rectangular area, and (ii) distinctly clustered customer points.

In order to test the performance of the algorithm, we needed to generate customer data for which the optimum solution is more or less known. For this precise reason, we generated $k$ clustered customer points. For such clustered points set, the optimum location to station $k$ facilities must be precisely at the center of each cluster. We executed the algorithm on such clustered customer points by picking arbitrary starting locations. The generated solutions, shown in Table 5 (Figure 5.11), confirms that the Voronoi driven $k$-median approximation algorithm produces near optimal solutions for clustered customer
For randomly generated customer data, the solutions obtained by the algorithm are as shown in Table 4, Figure 5.9. Inspection of the weighted distances in this table shows that the quality of generated locations are fairly good. Of course, for randomly generated point customers, we do not know the exact optimal solution and hence it is difficult to indicate how far the generated solution differs from the optimal solution. Furthermore, for all the executions, the algorithm converges to yield an approximate solution in less than 72 steps for data consisting of 1000 point customers.

For all test data (random, and clustered), the computation of single facility location in a Voronoi region is done by using (i) Weizfield’s method, and (ii) Center of Gravity method. The Weizfield’s algorithm is guaranteed to generate the optimal solution for a single facility location, within the pre-selected numerical accuracy. Thus it is expected that the Voronoi driven $k$-median approximation that uses Weizfield’s method will produce better quality results. Surprisingly, our experiment on different kinds and sizes of data shows that the Voronoi driven $k$-median algorithm based on Center of Gravity yields slightly better results. An inspection of the number of data in Figure 5.9 and 5.11 shows that the version based on the Center of Gravity converges in fewer number of iterations compared to the version based on Weizfield’s method.

The Voronoi based $k$-median approximation algorithm can be extended to address several variations of the multi-facility location problem. The initial locations that are used to start the Voronoi based $k$-median approximation algorithm are picked rather arbitrarily. Certainly, better quality starting locations would help the algorithm to reach convergence in fewer number of iterations. In this regard, it would be interesting to partition the customer points into $k$ disjoint components, each having comparable total weights. The center of gravity of customer points in such region could be taken as a good starting location. We believe that the plane sweep technique of computational geometry could be used to partition customer points into $k$ disjoint rectangular regions of comparable total weights. Detailed exploration of this approach to generate quality starting points would be interesting.

For the purpose of simplification, most multi-facility location problems allow any point
in the plane as a possible candidate for stationing the facility. In real world applications, not all points in the plane can be taken as feasible locations. For example, a county could contain regions such as lakes, parks, and private areas. Certainly, facilities are not allowed to be stationed in such forbidden regions. Stated in another way, only certain regions in the plane are designated as legal facility location regions.

Figure 6.1: Solution Modification

For such cases it would be necessary to extend the proposed Voronoi driven $k$-median approximation algorithm to address the possibility of forbidden regions. One approach to
tackle the presence of forbidden regions could be to first find the solution by assuming the whole domain as non-forbidden. If the generated solution points lie outside the forbidden regions then this solution is taken as an acceptable solution. If some of the generated points lie inside forbidden regions (we refer to such points as forbidden candidates) then we could locate points on the boundary of forbidden regions that are nearest to the forbidden candidates. Such boundary points could be taken as the approximate solution points. Figure 6.1 illustrates the concepts of this construction. Detailed investigation of this technique would be a possible extension of the proposed approximation algorithm.

As mentioned in Chapter 1, facility location problems in which solutions are constrained to lie along a given line have been considered. It would be interesting to investigate the applicability of the Voronoi based \( k \)-approximation algorithm to include such constraints. We believe that the intersection points of the Voronoi regions and line constraints could possibly help in locating improved solutions, and this issue is worth further investigation.

Most of the multi-facility location problems have been considered in two dimensions. The proposed Voronoi based \( k \)-approximation algorithm is also designed for customer points distributed in two dimensions. It seems very difficult to extend this approach to three dimensions. It may be noted that both Weizfield's method and Center of Gravity method for solving single facility location problems can be extended in a straightforward way to higher dimension. It would be extremely useful to develop a practical and simple approximation algorithm for multi-facility location problems that can be easily generalized to three dimensions.

Finally, customers are represented by weighted points in our model. It would be more general to extend the algorithm where customers are represented by shapes of finite extension such as convex polygons.
BIBLIOGRAPHY


VITA

Graduate College
University of Nevada, Las Vegas

Prashanth Kodela

Local Address:
    1170, Maryland Circle, Apt 1
    Las Vegas, NV 89119

Degrees:
    Bachelor of Science, Computer Science, 2000
    Kakatiya University, India

Thesis Title: Approximation Algorithms for Multi-Facility Location

Thesis Examination Committee:
    Chairperson, Dr. Laxmi Gewali, Ph.D.
    Committee Member, Dr. John T. Minor, Ph.D.
    Committee Member, Dr. Wolfgang Bein, Ph.D.
    Graduate Faculty Representative, Dr. Rama Venkat, Ph.D.