Algorithms for some Geometric Facility Location and Path Planning Problems

Doctoral dissertation submitted by

Sasanka Roy

for award of the Ph.D. degree of the

Indian Statistical Institute, Kolkata

Advisor :

Dr. Sandip Das

Advanced Computing and Microelectronics Unit

Indian Statistical Institute

Kolkata 700 108

June 2007

DEDICATION

to . . . who love me

Acknowledgment

My words are inadequate to convey my gratitude towards my supervisor Dr. Sandip Das. He had introduced me to the field of Computer Science, particularly the field of Computational Geometry and it has been a wonderful experience, ever since. I express my deep gratitude for his invaluable advice and guidance that made me more matured, both academically and otherwise.

Professor Subhas C. Nandy, who took me on the process of learning and made himself available even through his very heavy work and teaching schedule. Thank you does not seem sufficient but it is said with appreciation and respect. His company and assurance at the time of need would be remembered lifelong.

I owe a huge debt of gratitude to Professor Bhargab B. Bhattacharya for his valuable comments, suggestions and encouragement. His comments and criticisms have led a vast improvement in the quality of this thesis.

My special thanks also goes to Professor Bhabani P. Sinha who greatly enriched my knowledge with his exceptional insights in Computer Science. I am also thankful for his support and advice.

I am deeply indebted to the anonymous reviewers of this thesis who provided critical, insightful and timely feedback that helped me improve certain results and enhance the quality of my presentation.

I must also mention the encouragement, advice, and support I received from Professor Jayasree Dattagupta, Professor Nabanita Das, Dr. Susmita Sur-Kolay, Dr. Krishnendu Mukhopadhaya, Dr. Srabani Mukhopadhyaya and Mr. Partha P. Goswami. My heart-felt thanks to all of them. I am also thankful to Smt. Abira Das for her nonstop encouragement throughout my research.

I cherished my stay in the institute largely due to my friends. I thank Pritha, Subhasis-da, Arijit-da, Buddha-da, Sasthi-da, Gautam, Debasis, Sandeep, Bony, Pramod, Sandi,

Debabrata, Subhra, Ritwik, Sumana, Abhijit and many others for their encouragement and delightful company. I could not forget the cooperation of Boni and Pramod during this period. I also could not ever forget acquaintance of Partha Sarathi Mandal (beloved Partha Da) at the advent of my research career in ISI and also the love that I got form his family during this period. I express my sincere thanks to Pujari-da and other office staffs of ISI for assistance in official matters.

I would like to thank the Indian Statistical Institute, Kolkata for providing me a good research environment, the opportunity to get acquainted with distinguished researchers and financial support during the last couple of years of this research work.

I feel a deep sense of gratitude for my parents, my wife and specially my son who formed part of my vision and encouraged me to pursue this research work. I also feel a deep sense of gratitude for my uncle Sri Pulin Chandra Roy who always encourages me in my study since my childhood. I am also grateful to Subhra and her family for being supportive during this tenure.

(Sasanka Roy)
June 5, 2007
Advanced Computing & Microelectronics Unit
Indian Statistical Institute
Kolkata 700 108

Contents

Chapter 1

Introduction

The facility location problem is a resource allocation problem that mainly deals with adequate placement of various types of facilities to serve a distributed set of demands satisfying the nature of interactions between the demands and facilities and optimizing the cost of placing/maintaining the facilities and the quality of services.

The facility location problem is well-studied in the Operations Research literature and recently has received a lot of attention in the Computer Science community. For a company, the facility location problem provides more strategic decisions than just giving importance to locate the lowest cost space for storing its products. While identifying the location of the company's distribution centers (facilities) for maintaining the necessary service levels to the customers, it must consider several things, for example, the freight costs, the cost of a new/leased structures, several other logistics costs, and also the inherent risk and viability involved in the choice of those locations. Several variations of facility location problem can be formulated depending on the nature of the objective function and the constraints on the facilities. As examples, we may cite the problems on cost reduction, demand capture, equitable service supply, fast response time etc. For locating emergency facilities, such as hospitals, fire-fighting stations etc., covering a region using minimum radius circles is a natural mapping of the corresponding facility

location problem where the objective is to minimize the radius of the circles indicating the worst-case response time.

In the classical facility location problem, it is generally assumed that the communication path between a facility and a customer should not be obstructed by any obstacle. But, this is not always a realistic assumption with respect to the practical instances. So, depending upon the application, we model the problem assuming both the customers and the facilities as points in a polygonal region P and we measure the distance between a customer c_i and a point-facility x_j by their geodesic distance, i.e., the shortest path between c_i and x_j in P avoiding the obstacles. Finding the geodesic shortest path is an essential tool for solving several variations of the facility location problem. The complication of identifying the geodesic shortest paths increases when we consider different constraints that should be obeyed by the resulting path, or the region under study goes in higher dimension.

This thesis is a study on designing efficient algorithms for some application specific geometric facility location and constrained path planning problems. In the next two sections, we briefly overview the existing literature on the facility location problem and the geodesic path planning problem in two and higher dimensions. The scope of the thesis appears in the next section.

1.1 Facility Location

A typical facility location problem deals with locating facilities as a subset of a given set of objects on a given environment to cover the clients located on the same environment, say a bounded/unbounded plane, a terrain or some network, with an aim to optimize certain objective function. Formally, the problem is defined as follows: given a weighted set D of demand locations with weight distribution w, a set F of feasible facility locations with nonnegative cost distribution f, and a distance function d that measures cost between a pair of locations; the objective is to find a set $F' \subseteq F$, so as to optimize

certain objective function $\mathcal{F}$.

The environment that we generally consider is either continuous/discrete geometric space with Manhattan (L_1) or Euclidean (L_2) distance metric, or some graph network domain with usual graph theoretic distance metric. Locations for the facility or demand are typically of type points, lines, paths, cycles etc. or vertices of a graph, depending upon the type of environment. The objective function to be optimized is formulated considering the nature of the problem. The classical facility location problems are known as *k-center* and *k-median* problems, where k is a positive integer. The *1-center* problem was originally posed in the year 1857 by Sylvester [147], and the *1-median* problem was first introduced in Weber's book [155] in the year 1909, and is referred to as *Weber's problem* in the literature. The *1-median* problem in geometric domain is to identify the ideal location of a single facility in the plane that minimizes average time needed to reach any arbitrary demand location from the facility avoiding the obstacles present in that plane. If instead of one facility, the problem is to locate k (≥ 1) facilities, then it is known as *k-median* problem. In the *k-center* problem, the objective is to find a set F' of k supply points (facilities) so that the maximum distance between a demand point and its nearest supply point (in F') is minimized. Besides the *k-median* and *k-center* problems, several other application-specific facility location problems are considered in the literature depending on the nature of the objective functions. For example, in obnoxious facility location problem [24, 29, 102], the objective is to identify the locations F' of k undesirable facilities that maximizes the distance between a demand point in D and its nearest site in F'. The objective in the fault tolerant facility location problem [146] is to place a given number (say k) of facilities such that the total cost for accessing r facilities from any client is minimized. Multiple facilities provide a safeguard against failure. Glozman et al. [76] studied the problem of covering a set of points by a given number of shapes of some specific kind.

Hershberger [90, 91] presented algorithms for partitioning a given set D of n points into two subsets in order to minimize the sum and the maximum length of their diameters respectively. A useful extension of the facility location problem is the capacitated

facility location problem, where we need to consider an additional constraint on the size of each cluster, where a cluster is the subset of demand points served by the same facility. If n and k denote the number of demand points and the number of clusters (facilities) respectively, then the size of each cluster may be at most a constant c, where $c \geq \lceil n/k \rceil$. Most of these facility location problems are NP-Hard if k (the number of facilities) is considered as a part of the input [114]. For a fixed value of k, many of these problems can be solved in polynomial time, but some of them still remain intractable[3].

The facility location problem is a long-standing research problem. New variations are coming up from many practical applications. So, the exhaustive review of this literature is really a difficult task. A survey on *location theory* considering various environments and application-specific constraints are described in the book of Drezner [58]. A very good overview on *location theory* in the perspective of mathematical programming is available in the book by Mirchandani and Francis [120]. Some important recent results are available in the literature [3, 78]. In the next subsection, we present a brief review on efficient algorithms for facility location problems involving computational geometric technique.

1.1.1 Geometric facility location

The oldest problems on facility location, i.e., 1-*center* and 1-*median* problems, are geometric in nature. Till date, several new variations of geometric facility location problem are coming up in the literature depending on the necessity of newly evolved applications. Many of these problems are solved using the standard geometric tools, for example, Voronoi diagram, convex hull, visibility among the others [27]. In fact, the facility location is an important area of research in the field of computational geometry. In this thesis, the attention is centered on the variations of the problem that are particularly important in the transportation management and wireless communication. These variations of the problem consider the standard norm of considering facilities as points on the Euclidean plane, obstacles in the form of convex polygons and distance measures

in L_2 metric. In our case, the constraints are on the location of the facilities, which is a line, or a line segment, or the boundary (or a specific edge) of the obstacle polygon, etc.

1.1.2 Geometric k-median and k-center problems

The geometric k-median problems and the k-center problems of various types have received considerable attention in the computational geometry community in the recent years in the context of facility location. The 1-median problem in L_∞ metric can be solved in $O(n)$ time, where $n = |D|$ [22]. Several variations of this problem are NP-Hard in $\mathbb{R}^d$ for $d \geq 2$ and $k \geq 2$ [118]. Bajaj [20] has quoted that, under L_2 metric the *basic* Weber problem cannot be solved using radicals. But, several geometric techniques exist for computing the near-optimal solutions for the geometric k-median problem in both discrete and continuous domain [58, 89, 156]. A few references are also available on the k-median problem where the demand points form a continuous region [65]. The specific applications are in mobile communication where the facilities are the base stations and the demand points are the mobile users.

Several recent results for computing the approximate solutions of the *k-median* problem in the plane have been proposed in [18, 39, 40, 113]. Among these, the algorithms given in [39, 40] produce a solution which is a constant multiple of the optimum solution. Some popular local heuristics search techniques for hard combinatorial optimization problems also work nicely for this problem [14, 112, 104].

In the k-center problem, the goal is to find k center points (facilities) in order to minimize the maximum distance between a demand point and its nearest facility. In the *basic* k-center problem, the set of demand points D is discrete in $\mathbb{R}^d$, the set of facility locations F is the entire $\mathbb{R}^d$ plane, and the distance function is the Euclidean L_2 or L_∞ metric. This is known to be NP-complete if $d \geq 2$ [114]. For small values of k, parametric search technique is a useful tool for solving this problem efficiently. In its decision version, a radius r is given, and the problem is to determine whether D can be covered by the

union of k balls of radius r. In the discrete k-center problem, F is required to be a subset of demand points D. Hwang et al. [96] proposed an $n^{O(\sqrt{k})}$ time algorithm for the k-center problem in $\mathbb{R}^2$. A contemporary paper by the same set of authors [95] proposed another algorithm for the discrete version of this problem with the same running time. Therefore it makes sense to search for efficient approximation algorithms and heuristics for the general version [93, 128] and for the *basic* k-center problem in two or higher dimensions considering k as small and fixed constant [28, 38, 64, 76, 91, 98, 101, 140].

The first algorithmic result on the Euclidean *1-center* problem appeared in [63], and an $O(n^2)$ time algorithm was proposed in that paper. Later, Shamos and Hoey [139], Preparata [129] and Shamos [138] improved the time complexity of the problem to $O(n \log n)$. But, Bhattacharya et al. [33] pointed out that the diameter of the point set computed by the algorithms in [138] and [139] are incorrect, but in spite of that, [138] correctly reports the minimum enclosing circle.

Lee [110] proposed the furthest point Voronoi diagram, and using that data structure, the 1-center problem can be solved in $O(n \log n)$ time. Finally Megiddo [116] found an optimal $O(n)$ time algorithm for solving this problem using prune-and-search technique. The dynamic version (i.e., where insertion and deletion of points are allowed), can be found in [2, 21]. Many other variations of the 1-center problem are available in [56, 57, 59, 117].

While much has been done on such unconstrained versions of the classical 1-center problem, little has been done in the constrained case. Megiddo in [116] studied the situation where the center of the smallest enclosing circle lies on a given straight line. In [94], Hurtado, Sacristan and Toussaint provided an $O(n+m)$ time algorithm for finding minimum enclosing circle with its center constrained to satisfy m linear inequalities. This takes help of linear programming in $\mathbb{R}^2$.

Bose et al. [35] considered the generalized version of the problem where the center of the smallest enclosing circle of P is constrained to lie inside a given simple polygon of size m. Their proposed algorithm runs in $O((n+m)\log(n+m)+k)$ time, where

k is the number of intersections of the boundary of the polygon with the furthest point Voronoi diagram of P. In the worst case, k may be $O(n^2)$. This result was later improved to $O((n+m)\log m + m\log n)$ in [36]. In a further generalization of this problem, where r (≥ 1) simple polygons with a total of m vertices are given, locating the center of the smallest enclosing circle of P with its center inside one of those polygons was discussed by Bose and Wang [36]. The time complexity of this version is $O((m+n)\log n + (n\sqrt{r}+m)\log m + m\sqrt{r} + r^{\frac{3}{2}}\log r)$.

In the *2-center problem* for the point set P, the objective is to cover P by two closed disks whose radii are as small as possible. Sharir [140] presented a near-linear algorithm for this problem which runs in $O(n\log^9 n)$ time. Currently the best known algorithm for this problem was proposed by Chan [38]. It suggests two algorithms for this problem. The first one is a deterministic algorithm, and it runs in $O(n\log^2 n(\log\log n)^2)$ time; the second one is a randomized algorithm that runs in $O(n\log^2 n)$ time with high probability. A variation of this problem is the *discrete two-center problem*, where the objective is to find two closed disks whose union covers the point set P and whose centers are a pair of points in P. This problem was solved in $O(n^{4/3}\log^5 n)$ time by Agarwal et al. [4]. Recently, Kim et al. [103] proposed much efficient algorithms for both of the standard and discrete versions of the 2-center problem where the demand points are vertices of a convex polygon. Their algorithms run in $O(n\log^3 n\log\log n)$ and $O(n\log^2 n)$ time respectively. An improved result on the upper bound of the continuous/discrete weighted k-center problem on a tree has appeared very recently [23]. It proposes a linear time algorithm for the weighted 2-center problem.

1.1.3 Guard placement problem

Guard placement is another important area of research in geometric location theory. In general a point p on a given polygon guards the polygon P if every point $u \in P$ is visible from point p. There are several definitions of visibility in the polygon guarding problem. The most common one is that *a point p is visible from q if the line segment*

$[p, q]$ *does not pass through any obstacle*. Sometimes a single guard may not be able to see the entire polygon. In *Art Gallery theorem*, Chvatal [48] established the fact that $\lfloor n/3 \rfloor$ guards are always sufficient and occasionally necessary to guard the interior of a simple polygon with n vertices. Since then a tremendous amount of research effort was rendered on *Art Gallery problem* [125, 135, 142]. Linear time algorithms for computing the visibility polygon under standard notion of visibility was studied in [60]. A detailed survey on the problem of placing guards to watch the interior and exterior region of several types of polygons is available in [135]. It also considers the same problem under many different notions of visibility.

We will consider the notion of *L-visibility*, which is defined by Gewali et al. [72] as follows: consider a guard (robot) that can sense any movement or sound in its *territory* and can reach the source of the problem in short time. The *territory* of a robot at location p is a region R such that for each point $u \in R$ there exists a path from p to u avoiding the obstacles whose distance is bounded by a constant L, where the constant L denotes the power of the robot. Here, each point in the region R is said to be *L-visible* from p. Similar concepts of *L-visibility* were also used in [16, 124].

Given a point p on the boundary of a polygon P, and a constant L, the *external L-visibility problem* deals with the area computation outside the boundary of the polygon which is *L-visible* to the guard located at p. Here the obstacle is the polygon P itself.

The optimization version of the *external L-visibility problem* is to identify the location p of the guard on boundary of P with an objective to maximize the area of the region outside the polygon P which is *L-visible* to p. Gewali et al. [72], proposed a linear time algorithm for the external L-visibility problem where P is a convex polygon, and L is less than or equal to half the perimeter of P. As mentioned by the authors of [72], the open question was to design a polynomial time algorithm for the external L-visibility problem where L is greater than half of the perimeter of P.

1.2 Path Planning

Shortest path problem is one of the fundamental problems in all major application areas of computer science like facility location, operation research, robot motion planning to name a few. There are varieties of different shortest path problems depending on the applications in networks, transportation, guarding, visibility related problems, to name a few. Some of which have been solved and many of which have not even been considered. Computing a shortest path between two nodes s and t in a graph is a natural mapping of many real life problems particularly in the transportation and network domain where connection pattern among stations (nodes) in the network is already established.

Computing an optimal path in geometric domain satisfying different constraints is a fundamental problem in computational geometry, where the mapping of these types of problems into the standard graph theoretic problem is nontrivial. In many cases, these are solved by applying different intelligent techniques rather than mapping them into graph theory. Here, the objective is to design efficient algorithms to avoid explicit construction of the entire underlying graph whose size may become very large depending on the specific applications. Shortest paths between two points on the plane or on the surfaces of a polyhedra is one of the fundamental problems that have wide applications in several facility location problems, for example, traffic control, search and rescue, pipeline placement for fluid distribution, city planning etc. Depending upon the applications, it may be required to find the minimum cost path on the polyhedral surfaces satisfying some constraints imposed by the problem specification.

In graph theory, two major variations of the shortest path problem exist − (i) single pair shortest path, and (ii) all pair shortest path. Given a edge-weighted graph $G = (V, E)$, the objective in the first variation is to find the minimum cost path of each node in V from a specific node s, called source. This yields the shortest path tree rooted at s. The cost of a path is the sum of the weights of all the edges along that path. In the unweighted case (where unit weight is attached to each member in E), this problem can

be solved in $O(|E|)$ time using *Breadth First search* [51]. If the weight attached to each edge is a real positive number, then this problem can be solved by the classical *Dijkstra's algorithm* [54], and it needs $O(|E| \log |V|)$ time. The time complexity of this problem can be improved to $O(|E| + |V| \log |V|)$ using the *Fibonacci heaps* of Fredman and Tarjan [67]. Further improvements on the time complexity is possible if the edge weights are nonnegative integers [68, 69, 87, 150, 151, 152]. For undirected graphs with integer edge weights, improved algorithms were proposed by Thorup [150, 151] which bypasses the sorting step, and runs in $O(|E|)$ time. Dijkstra's algorithm does not work if the edge weights are allowed to be both positive and negative. The traditional algorithm, due to Bellman and Ford [51], runs in $O(|E||V|)$ time. Better results are available in [70, 71, 77] where the edge weights are both positive and negative integers.

In the second variation of the shortest path problem, the objective is to report the shortest path between each pair of vertices $(u, v) \in V$. The standard Floyd-Warshall's algorithm for this problem runs in $O(|V|^3)$ time. If the graph is sparse enough (i.e. $|E| << O(|V|^2)$), then one may use Johnson's algorithm, which runs in $O(|E||V|)$ time [99]. There exist many interesting variations of the shortest path problem in graphs. For a detailed survey, see [160].

Due to the high running time for computing the all-pair shortest paths in a graph, tremendous efforts were given on computing the approximate shortest paths between each pair of vertices in G. A path is said to be a multiplicative k-approximate shortest path if $\pi(u, v) \leq k \times \pi_{opt}(u, v)$, where $\pi(u, v)$ and $\pi_{opt}(u, v)$ are respectively the length of the approximate and the exact shortest paths between the two vertices u and v. If the length of each approximate path $\pi(u, v)$ is such that $\pi_{opt}(u, v) \leq \pi(u, v) \leq \pi_{opt}(u, v) + k$ then $\pi(u, v)$ is said to be the additive k-approximation of $\pi_{opt}(u, v)$. In [55], Dor et al. showed that for any finite k, the multiplicative k-approximations of the all pair shortest paths can not be computed in time less than that of multiplying two Boolean matrices (of size $O(|V| \times |V|)$). Zwick [158], proposed an algorithm for computing $(1+\epsilon)$-approximate paths. For further details on this topic, see [15, 50, 160]. Aingworth et al. [8] showed that the additive approximation results for this problem can be obtained

without using matrix multiplication. Several improved algorithms were proposed in [55, 62]. Many open problems still remain regarding the possible tradeoffs between the preprocessing time, space requirement, query answering time and the multiplicative approximation constant of the distance oracle.

1.2.1 Geometric shortest path

In the geometric version of the shortest path problem, we have polyhedral obstacles in $\mathbb{R}^d$ plane ($d \geq 1$). The objective is to find a path between a pair of points p and q avoiding the obstacles, which is of minimum length with respect to some specified metric. The general version of the shortest path problem in $\mathbb{R}^d$ was shown to be *NP-hard* by Canny and Reif [37] for $d \geq 3$. An extensive research has been done on solving geometric shortest path problems on two and three dimensions. In $\mathbb{R}^2$, the most basic version of the geometric shortest path problem is to find shortest path between two points s and t inside a simple polygon without holes. This can be solved in linear time [41, 111] assuming that the triangulation of a simple polygon can be done in linear time [43]. Given a simple polygon P and a point s inside it, Guibas et al. [85] constructed a *shortest path tree* data structure in $O(n)$ time and space such that for any arbitrary query point $q \in P$, the length of the shortest path from s to q can be reported in $O(\log n)$ time. This idea was further improved by Guibas and Hershberger in [84]; here the shortest path can also be reported in $O(\log n + k)$ time, where k is the number of the vertices on the output path. The dynamic version of this problem allows the polygon P to change by adding and/or deleting edges and vertices. If deletion or addition is made in such a way that it does not create a hole inside the polygon then a linear space data structure was proposed by Goodrich et al. [82], which can report (i) the length of the shortest path between any two query points $s, t \in P$ in $O(\log^2 n)$ time, (ii) the shortest path itself in $O(\log^2 n + k)$ time, and (iii) the updating of P can also be handled in $O(\log^2 n)$ time. This is an improvement of a contemporary result on the same problem given by Chiang et al. [47]. Several parallel algorithms on this problem were developed in [61, 80, 81, 92]. A detailed review on this topic is available in [135].

While discussing on the shortest path inside a polygon, we need to introduce the concept of visibility graph. Two points p and q inside a polygon (with/without holes) are said to be visible if the line segment joining them does not intersect the boundary of the polygon (as well as the boundary of the holes). The visibility graph (also called the vertex visibility graph) of a polygon with/without holes is an undirected graph whose nodes are the vertices of the polygon; an edge between a pair of nodes indicates that the corresponding vertices are mutually visible. The shortest path between a pair of points p and q inside a simple polygon with holes has direct relation with the visibility graph. The problem can be solved in $O(|V|\log|V| + |E|)$ time using the algorithm proposed by Ghosh and Mount [75]. For details, see the paper by Fredman and Tarjan [67] on shortest paths in visibility graph. The visibility graph may also find a variety of applications in robot vision and path planning.

Guibas et al. [85] and Toussaint et al. [153] first established a relationship between the weak visibility of a polygon and the Euclidean shortest paths. A polygon is said to be weakly visible from an edge e of the polygon P if for each point z inside P, there exists a point $w \in e$ such that w and z are mutually visible [73]. The shortest path tree is extensively used as a tool for computing different types of visibility polygons inside a given simple polygon [30, 31, 32, 53, 97].

In $\mathbb{R}^3$, the shortest path between a pair of points may not always pass along the surface of the polyhedral obstacles. As already mentioned, this problem is NP-complete. But, there exists many practical problems which need shortest path passing along the surface of the polyhedron. Such type of shortest paths are refereed to as geodesic shortest paths.

1.2.2 Shortest path through polyhedral surface

The shortest path problem between two points s and t on the surface of an unweighted polyhedron is studied extensively in the literature. Sharir and Schorr [141] presented an $O(n^3 \log n)$ time algorithm for finding the geodesic shortest path between two points on the surface of a convex polyhedron with n vertices. Mitchell et al. [121] studied

the generalized version of this problem where the restriction of convexity is removed. The time complexity of the proposed algorithm is $O(n^2 \log n)$. After a long time Chen and Han [46] improved the time complexity to $O(n^2)$, and then Kapoor [100] reduced it to $O(n \log^2 n)$. Recently, Schreiber and Sharir [136] proposed an interesting and implementable optimal algorithm for computing the shortest path map from a fixed source s on the surface of a convex polytope P in three dimensions; the running time of this algorithm is $O(n \log n)$ and it requires $O(n \log n)$ space. Two approximation algorithms for this problem were proposed by Varadarajan and Agarwal [154]; they can produce paths of length $7(1+\epsilon) \times opt$ and $15(1+\epsilon) \times opt$ respectively; opt is the length of the optimal path between s and t, and ϵ is an user specified degree of precision. The running times are respectively $O(n^{5/3} \log(5n/3))$ and $O(n^{8/5} \log(8n/5))$. For convex polyhedron, an approximation algorithm was proposed by Agarwal et al. [1], which produces $(1+\epsilon) \times opt$ solution, and runs in $O(n/\sqrt{\epsilon})$ time.

In the weighted version of the problem, each face f is assigned a weight $w(f) \in [0, \infty)$. The weight of a face indicates the cost of traversing unit length through that face. Given two points s and t on the surface of the polyhedron, the objective is to find a path Π from s to t of minimum cost. Formally, the cost of a path is defined as follows:

> Let Π be the concatenation of a set of line segments $\{\sigma_1, \sigma_2, \ldots \sigma_k\}$, such that each σ_i may be any one of the following two types: (A) it lies completely on a single face f, or (B) it lies on an edge shared by a pair of faces f' and f''. The cost of the path Π, denoted by $cost(\Pi)$, is equal to $\sum_{i=1}^{k} w_i \mid \sigma_i \mid$, where $\mid \sigma \mid$ is used to denote the length of a line segment σ, and $w_i = w(f)$ or $min(w(f'), w(f''))$ depending on whether σ_i is of type A or type B respectively.

The first work on approximating the minimum cost path of the weighted polyhedral surface appeared in a seminal paper by Mitchell and Papadimitrou [122]. It uses *continuous Dijkstra* method [121] and exploits the fact that the minimum cost path follows *Snell's law of refraction*. The algorithm locates a path whose weighted length is guaranteed to be within a factor of $(1+\epsilon)$ of the length of an optimal path, where $\epsilon(>0)$ is an user-defined constant. The time complexity of the algorithm is $O(n^8 \times \log(\frac{nNW}{\epsilon}))$,

where N is the largest integer coordinate among the vertices and W is the maximum weight among the faces of the polyhedron. An implementable method for solving the minimum cost path problem was given by Mata and Mitchell [115]; this formulates the problem as a graph search problem and assures a solution of length $(1+\epsilon) \times opt$. The running time of the algorithm is $O(\frac{n^3 N^2 W}{\epsilon w})$, where W is defined earlier and w is the smallest weight among all faces of the polyhedron.

Lanthier in his Ph.D. thesis [107] proposed several algorithms on this problem. The fastest algorithm runs in $O(n \log n)$ time and produces a solution with worst case length $\frac{2}{\sin(\theta)} \times opt$, where opt denotes the length of the optimum path. Lanthier et al. [109] also proposed an approximation algorithm for the minimum cost path problem, which adds equally spaced Steiner points on the edges of the polyhedron, and approximates the cost of the optimum path to $opt' = opt + LW$, where W is as defined earlier, and L is the longest edge of the polyhedron. The running time of the algorithm is $O(n^5)$. In the same paper, another algorithm was presented using graph spanners; it runs in $O(n^3 \log n)$ time to report an approximation of the optimum path whose cost is no more than $\beta(opt + LW)$, where $\beta > 1$. The best known implementable algorithm for this problem is due to Aleksandrov et al. [10], and it claims to produce a path of cost $(1+\epsilon) \times opt$ among a pair of query points in a weighted polyhedron. Several further improvements were also proposed by Aleksandrov et al. in a sequence of papers [11, 12, 13]. These are based on the method of introducing Steiner points on the edges of the triangulated polyhedron. The latest version [13] runs in $O(C(P)\frac{n}{\sqrt{\epsilon}} \log \frac{n}{\epsilon} \log \frac{1}{\epsilon})$, where $C(P)$ captures the geometric parameters and the weights of the faces of the given weighted polyhedron. The geometric parameters include the longest edge of the triangulated polyhedron and the smallest angle among all the triangles. In that paper the authors raised a very important question about the existence of a method which does not depend on the geometric parameters of the polyhedron. Recently, Reif and Sun [133] observed that the dependency of $C(P)$ on the weights of the faces can be removed. It needs to be mentioned that the time complexity of computing shortest paths on the unweighted polyhedral surfaces does not depend on the geometric parameters [1, 45, 46, 154].

In the special case, where an edge sequence $\mathcal{E} = \{e_1, e_2, \ldots, e_k\}$ of the triangulated polyhedron is specified along with the source (s) and target (t), and the objective is to find an weighted shortest path from s to t through the interior points of $e_1, e_2, \ldots, e_k$ (in that order), then also the running time of the algorithm for computing an optimal (or approximate) weighted shortest path depends on the geometric parameters. The main reason, as observed by Mitchell and Papadimitriou in [122], is that the local optimality criteria along an edge is given by Snell's law of refraction. Thus, if the number of edges in the edge sequence is k, then the required objective function is the sum of square roots of k quadratic expressions with k unknowns along with several other constraints. While solving, the elimination process of the variables leads to a very high degree algebraic equation whose coefficients depend on the geometric parameters.

1.2.3 Shortest path on the surface of a terrain

The problem of studying different variations of the facility location problem on terrain is an important area of research in GIS [17]. Berg and Kreveld [26] studied several variations of the path finding problem on the surface of a polyhedral terrain. A terrain $\mathcal{T}$ is a two dimensional surface in the three dimensional space with a special property: the vertical line at any point on the XY-plane intersects the surface of the terrain at most once. In other words, it is the graph of a function $f : A \subset R^2 \rightarrow R$, where A is a region on the XY-plane and $f(p)$ indicates the height of the surface of $\mathcal{T}$ at the point $p \in A$. Thus, a terrain can be viewed as a polyhedral surface specified by a set of faces, edges and vertices, where the vertices are the end points of the edges, each edge is the intersection of two faces, each face is a plane in 3D, and the projections of all the faces on the XY-plane are mutually non-intersecting at their interior. Given a polyhedral terrain $\mathcal{T}$ with n vertices, the proposed algorithm efficiently answers the query for the existence of a path between a pair of points s and t on the surface of $\mathcal{T}$, such that for each point $p = (x(p), y(p), z(p))$ on the path $z(p) \leq \xi$ for some given altitude (height) ξ. It also determines the minimum total ascent/descent among the path(s) between s and t, where the *total ascent* of a non-monotone path is defined in [26].

Recently an interesting variation of the shortest path problem in the context of a terrain was proposed by Mitchell and Sharir [123], where the objective is to compute the L_1-shortest path between a given pair of points, such that the path is restricted to lie on or above the given polyhedral terrain $\mathcal{T}$. The proposed algorithm runs in $O(n^3 \log n)$ time, where n is the number of faces of $\mathcal{T}$. The same paper also studies another variation of the shortest path problem on a terrain like structure, where a set of n vertical walls parallel to the x-axis are given. Each wall is positioned on the xy-plane. The i-th wall, is positioned at $y = a_i$, and its top boundary (denoted by e_i) is a line of the form $z = b_i x + c_i$, where a_i, b_i and c_i are given constants, $a_1 < a_2 < \ldots < a_n$. The objective is to report the L_2-shortest path between a given pair of query points s and t, where $s < a_1$ and $t > a_n$. The problem is referred to as the *L_2-shortest path over walls*. Note that, the shortest path is always monotone with respect to the y-axis, and it bends on the edges e_i, $i = 1, 2, \ldots, n$. It is also proved that, the shortest path from s to t is the concatenation of two sub-paths, one of them is monotone ascending and the second one is monotone descending with respect to z-coordinate. The standard method for solving this problem involves a preprocessing phase which splits each edge e_i into segments, and then defines the shortest path map [135]. The optimal L_2-shortest path from s to t can be obtained by following an appropriate path in that map. It is proved that the size of the shortest path map is $O(n^2)$ in the worst case, but finding a polynomial time algorithm for constructing the map is left as an open problem [123].

There are several other variations of path finding problem on the surfaces of polyhedral terrain. Rowe and Ross [134] addressed the problem of finding minimum-energy routes for a mobile agent across some hilly terrain. The specific problem they have considered is to find the optimal path of a mobile agent under anisotropic friction and gravity effect. Lanthier et al. [108] proposed an $(1+\epsilon)$-approximation algorithm for computing the shortest path in anisotropic scenario. Recently, Gray and Evans [83] introduced the concept of uncertain terrain, and proved that finding shortest path in an uncertain terrain is NP-complete.

1.3 Scope of the Thesis

This thesis is a study on some important variations of the facility location problem. We first concentrate on a constrained version of minimum enclosing circle problem which has potential applications to the placement of base stations for wireless communication avoiding the forbidden zones. If a feasible route (a line or a line segment) is given online, the same problem of placing a single base station on it, is also studied. Next, we consider the problem of placing guard under L-visibility (see Subsection 1.1.3). These three problems are elaborated in Subsections 1.3.1, 1.3.2 and 1.3.3.

We then switch to two important variations of the shortest path problem on the surface of a polyhedron in $\mathbb{R}^3$ and on the surface of a terrain. These are useful tools for the facility location problems, and are described in Subsections 1.3.4 and 1.3.5.

1.3.1 Constrained minimum enclosing circle problem

Here a convex polygonal region P is given. The objective is to cover P by a circle of minimum radius whose center is placed on the boundary of P. To be more precise, our task is to identify the point on the boundary of P where the center of the desired circle needs to be placed. Our proposed algorithm produces the optimum solution in $O(n)$ time, where n is the number of vertices of P.

Instead of covering P with one such constrained circle (with its center on the boundary of P), covering P with k circles ($k \geq 2$) with their centers on the boundary of P is a trivial generalization of the problem. We will consider only a special case with $k = 2$, and an edge e of P is specified for placing the center of two covering circles. The objective is to minimize the maximum radius among these two circles. Our proposed algorithm for this problem also runs in $O(n)$ time when the edge e of P is given a priori.

1.3.2 Constrained minimum enclosing circle problem: a query version

We will also address a query answering version of the constrained minimum enclosing circle problem for the polygon P. Here a polygon P is given and the query problem is to report the smallest enclosing circle of P whose center lies on a line ℓ, supplied at the query time. We will preprocess P in $O(n \log n)$ time and $O(n)$ space, and given any arbitrary query line segment ℓ, the center and radius of the smallest enclosing circle can be reported in $O(\log^2 n)$ time. We will use this result for solving the following problem posed by Bose and Wang [36]:

> Given a set P of n points, and a set of r simple polygons with a total of m vertices, compute the smallest enclosing circle of P whose center lies inside one of these r polygons.

An algorithm for this problem was also proposed in [36]; it runs in $O((m+n)\log n + (n\sqrt{r}+m)\log m + m\sqrt{r} + r^{\frac{3}{2}}\log r)$ time. We show that, it can be solved in a much simpler way using our method, and the time complexity improves to $O(n \log n + m\log^2 n)$.

1.3.3 Guard placement under L-visibility

Given a set of polygonal obstacles, two points a and b are said to be *L-visible* if the length of the shortest path from a to b avoiding the obstacles is no more than L. For a given convex polygon P, Gewali et al. [72], addressed the problem of placing a single guard on the boundary of P such that it can see maximum area outside the polygon P under *L-visibility*. Their proposed linear time algorithm works when L is less than or equal to half the perimeter of P. They also posed an open question of designing an efficient algorithm when L is greater than half the perimeter of P. In this thesis, we will address that open problem and present a linear time algorithm.

1.3.4 Approximate shortest path in weighted polyhedra

In this problem, a triangulated polyhedron in $\mathbb{R}^3$ is given. Its each face is attached with a positive weight. The objective is to compute the minimum cost path between a pair of points s and t on the surface of the polyhedron, where the path is constrained to lie on the surface of the polyhedron. The cost of a path is defined in subsection 1.2.2.

For any given $\epsilon > 0$, the best known algorithm that produces $(1+\epsilon)$-approximation result for this problem, appeared in [13]. It is based on the method of introducing Steiner points on the edges of the triangulated polyhedron, and it runs in $O(C(P)\frac{n}{\sqrt{\epsilon}} \log \frac{n}{\epsilon} \log \frac{1}{\epsilon})$ time, where $C(P)$ captures the weights of the faces, and two geometric parameters of the given weighted polyhedron. These are (i) the length of the longest edge L of the triangulated polyhedron and (ii) the smallest angle θ among all the triangles. The existence of a method which does not depend on the geometric parameters of the polyhedron is mentioned as an important issue [13].

We have tried to remove type (ii) geometric parameter of $C(P)$ from the time complexity of the algorithm, but it plays role in the approximation factor. The time complexity of our algorithm is $O(n(\log^2 \frac{L}{\epsilon}) + n(\log \frac{L}{\epsilon}) \log n)$. In general, the approximation factor of our algorithm is $(1 + \frac{1}{sin\theta}) + \epsilon nW$, where W is the maximum weight among the faces of the triangulated polyhedron P.

1.3.5 Monotone descent path problem on a polyhedral terrain

Next we will consider the shortest monotone descent path problem in a polyhedral terrain. Here a polyhedral terrain $\mathcal{T}$ and a point s is given. The point s is considered to be the source of water in the terrain like hilly area. The objective is to identify the region on the surface of the terrain, where the water can flow from s. Thus, our problem is to find the region (R) on the terrain such that for every point $p \in R$ there exist monotone descent path (as defined below) from s to p.

A path will be said to be a monotone descent if for every pair of points $p = (x(p), y(p), z(p))$ and $q = (x(q), y(q), z(q))$ on the path with p is closer to s than q, then $z(p) \geq z(q)$.

In this thesis, we will develop an algorithm to identify the descent flow region of s (called $DFR(s)$) in $\mathcal{T}$. It needs $O(n\log n)$ time and stores $DFR(s)$ in a data structure of size $O(n)$, where n is the number of faces in the triangulated terrain. Given an arbitrary point t, it can report whether a monotone descent path exists from s to t along the surface of $\mathcal{T}$ by searching the $DFR(s)$ in $O(\log n)$ time.

It needs to be mentioned that, if a query point $t \in \mathcal{T}$ lies in the descent flow region of s, then reporting the shortest monotone descent path from s to t seems to be difficult for an arbitrary terrain $\mathcal{T}$ [26]. We study some restricted classes of terrain for which the shortest monotone path problem can be solved in polynomial time.

In particular, we have shown that for a convex terrain, the shortest monotone descent path from s to a query point t can be reported in polynomial time. We preprocess the given terrain $\mathcal{T}$ to create a data structure in $O(n^2 \log n)$ time and $O(n^2)$ space. This data structure stores the region in $DFR(s)$ which can be reached from s through a sequence of convex faces. Now, if a query point t lies in such a region, our algorithm can output the shortest monotone descent path from s to t in $O(k + \log n)$ time, where k is the number of faces through which the path passes.

We could identify another class of terrain where the shortest monotone descent path among a pair of points s and t can be found in polynomial time, but the convexity among each pair of faces in $DFR(s)$ is not required.

Here, along with the source (s) and destination (t), a sequence of faces $\mathcal{F} = \{f_0, f_1, \ldots, f_m\}$ is given, where $s \in f_0$, $t \in f_m$. The objective is to find the shortest descent flow path through $\mathcal{F}$, provided it exists. This problem also seems to be difficult in its general form. But we could design an efficient solution in this setup where the edges separating the consecutive faces in $\mathcal{F}$ are parallel to each other. The time complexity of the proposed algorithm is $O(m)$.

Chapter 2

Constrained Minimum Enclosing Circle Problem

In this chapter, we present a constrained version of the classical 1-center problem that has applications particularly in wireless communication and disaster management systems. In wireless communication system, base stations are positioned appropriately in the desired region. Each base station is assigned a range r to transmit messages to other radio stations or mobile radio sets inside a circular range of radius r centered at its placement location. In general, the power required by a base station is proportional to the square of its range and so the placement and efficient range assignment of the base stations is an important area of research.

Sometimes fixing the location of the base station becomes difficult if the region is a huge water body, or a dense forest or some other prohibited zone. However, we need to provide mobile communication service inside that region. In order to minimize the power requirement (or effectively the range) of the base stations, we have to place the base stations in some appropriate locations on the boundary of that region. For the sake of simplicity, we consider that the given region is convex. Here the objective is to locate the position of base station(s) with some additional constraints such that every point

inside that polygon is covered by these base station(s). In other words, every point inside that polygon is within the range of at least one base station and the maximum among the ranges of these base stations is minimized. We will consider the following two problems in the context of placing base stations on the boundary of a polygonal region.

Problem $P1$: Locate a point α on the boundary of the polygon P such that the maximum among the distances from α to all the points inside the polygon P is minimized.

Problem $P2$: Identify two points γ and δ on a given edge e of the polygon P and a real number r such that every point x inside the polygon P is covered by at least one of the circles centered at γ or δ of radius r and the value of r is minimum for such choices of γ and δ.

In Section 2.1, we address the problem $P1$ and propose a linear time algorithm for this problem. In Section 2.2, a linear time algorithm for the problem $P2$ is proposed.

2.1 Problem $P1$

Euclidean 1-center problem is a well known problem which has a long history. Here the problem is to find the smallest circle that encloses a given set of n points. In the standard 1-center problem, there is no restriction on placement of the center of that circle. Some interesting results were provided by Megiddo [116] and Hurtado et al. [94] on constrained version of the problem. Megiddo [116] studied the case where the center of the smallest enclosing circle must lie on a given straight line. Hurtado et al. [94] used linear programming to provide an $O(n + m)$ time algorithm for finding minimum enclosing circle with its center satisfying m linear constraints.

We will address problem $P1$ with a different type of constraint. Instead of placing the center inside a given convex region, we consider the center on the boundary of the given

convex polygon, and the objective is to cover the entire region inside the polygon. A similar problem was first addressed by Bose and Toussaint [35], where the center of the minimum enclosing circle lies on the boundary of a convex polygon of size n and the objective is to cover a set of m points which may not lie on or inside the polygon. An $O((n+m)\log(n+m))$ time algorithm for that problem was also presented in that paper. Here we derive some interesting geometric characterizations and propose an $O(n)$ time algorithm for problem $P1$ that avoids the use of linear programming techniques.

Let the vertices of the convex polygon P be $\{v_0, v_1, \ldots, v_{n-1}\}$ in anticlockwise order. We will use e_i to denote the edge (v_i, v_{i+1}) of P. If $\mathcal{C}$ denotes the minimum radius circle enclosing P whose center α is on the boundary of P, then $\mathcal{C}$ must satisfy the following simple but interesting observations.

Observation 2.1 *The circle $\mathcal{C}$ must pass through at least one vertex of the polygon P.*

Observation 2.2 *Let e be the edge of the polygon P that contains the point α. If the circle $\mathcal{C}$ passes through exactly one vertex v of polygon P, then the line $\overline{v\alpha}$ is perpendicular to the edge e at point α.*

Let us consider the furthest point Voronoi diagram $\mathcal{V}(P)$ of the vertices $\{v_0, v_1, \ldots, v_{n-1}\}$ of the polygon P. It partitions the plane into regions, $\mathcal{R}(v_0), \mathcal{R}(v_1), \mathcal{R}(v_2), \ldots, \mathcal{R}(v_{n-1})$, such that for any point $p \in \mathcal{R}(v_j)$, $d(p, v_j) \geq d(p, v_i)$ for all $v_i \in P$, where $d(.,.)$ denotes the Euclidean distance between a pair of points. From Observation 2.1, we can conclude that if v_i is on boundary of $\mathcal{C}$ then v_i is farthest vertex from α and hence α must be in $\mathcal{R}(v_i)$. The circle $\mathcal{C}$ may pass through more than one vertex of the polygon P, and in that case we have the following observation.

Observation 2.3 *If the circle $\mathcal{C}$ passes through two vertices of polygon P, then α must be at the intersection point of an edge of $\mathcal{V}(P)$ with an edge of the polygon P. Moreover, if $\mathcal{C}$ is passing through more than two vertices of polygon P, then α coincides with a vertex of $\mathcal{V}(P)$.*

From the above observations, we can conclude the following lemma.

Lemma 2.1 *If the center α of the circle $\mathcal{C}$ lies on an edge e of the polygon P, then α must coincide with either the perpendicular projection of some vertex of P on the edge e, or with the intersection point of an edge of $\mathcal{V}(P)$ and the edge e of P.*

Proof : Lemma follows from Observations 2.2 and 2.3. □

We consider each edge e of P, and locate the vertices of P whose projection on e lies inside the edge segment e. We use $\mathcal{B}$ to denote the set of these points on e which are obtained by these projections. Note that, if a point $\beta \in \mathcal{B}$ is the projection of a vertex v on an edge of P and $\beta \in \mathcal{R}(v)$, then the smallest enclosing circle of P with center at β passes through v. So, we consider only those members in $\mathcal{B}$ that lie in the farthest point Voronoi region of the corresponding vertex of P. Each element $\beta \in \mathcal{B}$ is attached with a vertex v such that $\beta \in \mathcal{R}(v)$.

In order to compute the smallest enclosing circle with center on the boundary of P and which passes through two vertices of P, we need to compute the set of points $\mathcal{A}$ that are generated due to the intersection of the edges of $\mathcal{V}(P)$ with the edges of the polygon P. Each element $u \in \mathcal{A}$ is attached with a pair of vertices (v, v') of P such that u is the point of intersection of the boundary of P and the Voronoi edge that corresponds to the bisector of the line segment $[v, v']$, and v' appears after v along the boundary of P in anticlockwise order. The members in set $\mathcal{A}$ partition the boundary of polygon into a set of polygonal chains. Each of these chains must lie inside a single Voronoi region, and it is formed by a sequence of polygonal edges bounded by a pair of consecutive members of set $\mathcal{A}$. Let us denote the set of these polygonal chains as $\mathcal{D}(P)$.

The computation of the sets $\mathcal{A}$ and $\mathcal{B}$ using the farthest point Voronoi diagram needs $O(n \log n)$ time. Although the farthest point Voronoi diagram for the vertices of a convex polygon can be computed in linear time [5], computation of all the intersection points of the Voronoi edges with polygonal boundary of polygon P needs $O(n \log n)$ time (as stated by Bose and Toussaint [35]).

We present a simple $O(n)$ time algorithm for finding the set $\mathcal{A}$ and $\mathcal{B}$. This avoids the computation of the farthest point Voronoi diagram. But, we need to study the properties of the farthest point Voronoi diagram of the vertices of a convex polygon, which will help in the formulation of our algorithm.

Lemma 2.2 *[34] Each cell of $\mathcal{V}(P)$ is an unbounded convex region.*

Lemma 2.3 *Let e be an edge of the polygon P. The perpendicular bisector of e must define one of the edges of the boundary of a Voronoi cell of $\mathcal{V}(P)$. Furthermore, this boundary is a half-line.*

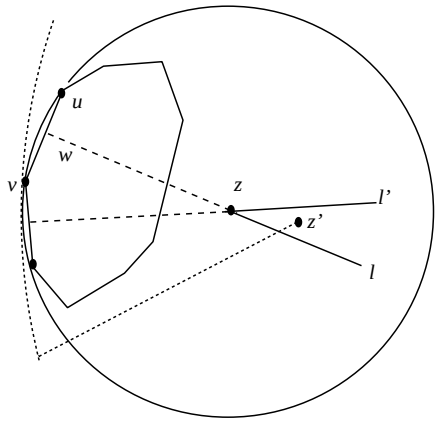

Figure 2.1: Illustrating the proof of Lemma 2.3

Proof : Let l be the perpendicular bisector of an edge $e = (u, v)$ of P. The line l intersects e at a point w. If we move along the line l from w on the direction towards the interior of the polygon then we can locate a point z on line l such that the circle centered at z with radius equal to $d(z, u)$ encloses all other vertices of P. Hence, the portion of l from z towards the other side of w is a half line, and it defines Voronoi edge separating $\mathcal{R}(u)$ and $\mathcal{R}(v)$ (see Figure 2.1). □

Lemma 2.4 *If the perpendicular bisectors of a pair of adjacent edges $e_i = (v_{i-1}, v_i)$ and $e_{i+1} = (v_i, v_{i+1})$ of a convex polygon P intersect outside P, then $P \cap \mathcal{R}(v_i) = \emptyset$.*

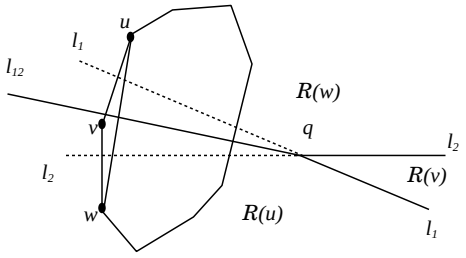

Figure 2.2: Illustrating the proof of Lemma 2.4

Proof : Let l_1 and l_2 be the perpendicular bisectors of e_i and e_{i+1} respectively and they intersect at a point q outside P. The Voronoi cells $\mathcal{R}(v_{i-1})$ and $\mathcal{R}(v_i)$ are in the two different half planes defined by the line l_1. Similarly, the Voronoi cells $\mathcal{R}(v_i)$ and $\mathcal{R}(v_{i+1})$ are in the two different half planes defined by the line l_2. The Voronoi cell $\mathcal{R}(v_i)$ is in the common region of the aforesaid two half planes as shown in Figure 2.2, which is outside the polygon P. □

From the above lemma, we can conclude that the intersection of a Voronoi cell with boundary of P is a simple contiguous chain and hence the cardinality of the set $\mathcal{A}$ is less than or equal to the number of vertices in P. Let $u_0, u_1, \ldots, u_{k-1}$ be the points in set $\mathcal{A}$ and they are in anticlockwise order on the boundary of the polygon P. As mentioned earlier, the members in $\mathcal{A}$ define the set of polygonal chains $\mathcal{D}(P)$. Let $\mathcal{D}(P) = \{\psi_0, \psi_1, \ldots, \psi_{k-1}\}$, where the chain ψ_i is bounded by the points u_i and u_{i+1}. Let $\psi_i \in \mathcal{R}(v'_i)$, where $v'_i \in \{v_0, v_1, \ldots, v_{n-1}\}$. The farthest neighbor of all the vertices in ψ_i (if exists) is v'_i. We will use $f(v)$ to denote the farthest neighbor of vertex v, $v \in P$. If $v \in \psi_i$, then $f(v) = v'_i$. We would also introduce a new function $index()$, where $index(v'_i) = j$ whenever $v'_i = v_j$. The following lemma demonstrates the arrangement of Voronoi cells along the boundary of the polygon.

Lemma 2.5 *If* $index(v'_r) = min\{index(v'_0), index(v'_1), \ldots, index(v'_{k-1})\}$, *then* $index(v'_r) < index(v'_{r+1}) < \ldots < index(v'_{k-1}) < index(v'_0) < \ldots < index(v'_{r-1})$.

Proof : Two adjacent polygonal chains ψ_r and ψ_{r+1} meet at point u_{r+1}, which is on the perpendicular bisector (say l) of the line segment joining the vertices $v'_r, v'_{r+1} \in P$.

The vertex v'_r (resp. v'_{r+1}) and the polygonal chain ψ_r (resp. ψ_{r+1}) lie in different sides of the line l as shown in Figure 2.3. So, a circle C centered at u_{r+1} with radius $d(u_{r+1}, v'_r)$ passes through v'_r and v'_{r+1}, and contains the polygon P. As the vertex v'_{r+1} is on anticlockwise direction of v'_r, and the index of v'_r is the least among all the indices of v'_i $(0 \leq i < k)$, we have $index(v'_r) < index(v'_{r+1})$.

Next, we prove the remaining part of the result. Note that, any circle with center on the boundary of P and containing P does not intersect the circular arc $\widehat{v'_r v'_{r+1}}$ of C (see Figure 2.3). If $index(v'_{r+1}) - index(v'_r) > 1$, then for any integer $\beta \in [index(v'_r), index(v'_{r+1})]$, there does not exist a point on boundary of polygon P from which v_β is farthest among all vertices of polygon P. Therefore, $index(v'_j) > index(v'_{r+1})$ for all $j = 0, 1, \ldots, r-1, r+2, \ldots, k-1$. Hence, the lemma follows. □

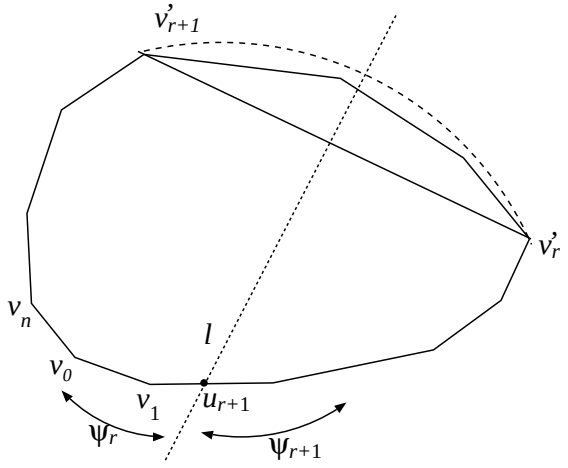

Figure 2.3: Illustrating the proof of Lemma 2.5

2.1.1 Algorithm

We first compute two arrays $\mathcal{A}$ and $\mathcal{B}$. One of the elements from $\mathcal{A}$ or $\mathcal{B}$ will be the center of the maximum enclosing circle of P.

Algorithm **Computation-of-Array-$\mathcal{A}$**

Input: Polygon P with n vertices.

Output: The array $\mathcal{A}$ generated in anticlockwise order.

Procedure: We traverse the vertices of P in anticlockwise order. If for two consecutive vertices v_i and v_{i+1}, $f(v_i) \neq f(v_{i+1})$, then we compute the u_is' (the members in $\mathcal{A}$) that lie on edge $e_i = (v_i, v_{i+1})$ as follows:

Let $v_\gamma = f(v_i)$ and $v_\delta = f(v_{i+1})$.
(* For every pair of vertices v_j and $v_{j'}$ with $\gamma \leq j < j' \leq \delta$, their perpendicular bisector intersects e_i *)
Set $j = \gamma$ and $j' = j + 1$.

Repeat the following steps *until* $j' = \delta$

Step 1: Draw the perpendicular bisector of v_j and $v_{j'}$, and compute its intersection point u .

Step 2: *If* u appears to the right (towards anticlockwise direction) of the last element of array $\mathcal{A}$, *then* add u in the array $\mathcal{A}$ with the pair of vertices $(v_j, v_{j'})$.
Set $j = j'$ and $j' = j' + 1$.

Step 3: *If* u appears to the left (towards clockwise direction) of the last element of array $\mathcal{A}$, *then* (* the region $\mathcal{R}(v_j)$ does not intersect the boundary of P *) delete the last element of $u' \in \mathcal{A}$.
If the pair of vertices attached to u' is $(v_\theta, v_{\theta'})$, *then* set $j = \theta'$
(* j will never go beyond γ, because $\mathcal{R}(v_\gamma)$ intersects e_i *).

Algorithm **Computation-of-Array-$\mathcal{B}$**

Input: The polygon P and the array $\mathcal{A}$.

Output: The array $\mathcal{B}$ generated in anticlockwise order.

Procedure: Traverse the array $\mathcal{A}$ to extract the chains $\{\psi_i, i = 1, 2, \ldots, k\}$. Each ψ_i is attached with the corresponding v'_i.

For each $i = 1, 2, \ldots, k$ *do*

For each edge/edge-segment $e \in \psi_i$ *do*

Draw the perpendicular projection of v'_i on e. Let it be w.

If w lies inside e, *then* add it in $\mathcal{B}$ with the vertex v'_i.

Lemma 2.6 *The elements of set $\mathcal{A}$ and $\mathcal{B}$ can be located in $O(n)$ time.*

Proof : We use the monotone matrix searching technique to compute the farthest neighbor $f(v_i)$ for every vertex v_i of the convex polygon P in $O(n)$ time [7]. Note that, the vertices $f(v_0), f(v_1), \ldots, f(v_{n-1})$ are in anticlockwise order, and the chains $\psi_0, \psi_1, \ldots, \psi_{k-1}$ are also in anticlockwise order. There may exist some Voronoi cell $\mathcal{R}(v)$ that does not contain any vertex of P. If in addition, $P \cap \mathcal{R}(v) \neq \emptyset$, then the chain in the cell $\mathcal{R}(v)$ is a segment of an edge of P, and that segment is bounded by two consecutive members in $\mathcal{A}$. As, $v_\gamma = f(v_i)$ and $v_\delta = f(v_{i+1})$ (as defined in *Computation-array-*$\mathcal{A}$), we have $P \cap \mathcal{R}(v_j)$ is either empty or a segment of edge e_i for each $j = \gamma + 1, \ldots, \delta - 1$ (see Lemma 2.4). We identify these members of $\mathcal{A}$ (on e_i) by observing the boundary of the Voronoi cell corresponding to vertices $v_\gamma, v_{\gamma+1}, \ldots, v_{\delta-1}$ in that order. Step 3 indicates the case where $P \cap \mathcal{R}(v_j)$ is empty and therefore the newly enumerated u appears to the left of the last element of array $\mathcal{A}$. We execute Steps 1-3 for each of the n vertices of P, and each iteration produces an u in $\mathcal{A}$. In addition, for the deletion of each element in $\mathcal{A}$ (in Step 3) we execute Steps 1-3 once. If the final length of $\mathcal{A}$ is k, $n - k$ elements of $\mathcal{A}$ will be deleted. Thus, the result follows. □

Now, we are in a position to present the algorithm for identifying the optimum location of the center of the minimum enclosing circle of P on boundary of the P.

Algorithm Problem_$P1(P)$

Input: Polygon P with n vertices.

Output: The point α on boundary of P.

Step 1: Compute the set $\mathcal{A} = \{u_0, u_1, \ldots\}$ using algorithm *Computation-of-Array-*$\mathcal{A}$.

Assign $d_{min} = \infty$.

For each $u_i \in \mathcal{A}$ *do*

(* Let v_i' be the one of the farthest neighbors of u_i, and is attached to u_i *)

If $d(u_i, v_i') < d_{min}$, *then* assign $\alpha = u_i$ and $d_{min} = d(u_i, v_i')$.

Step 2: Compute the set $\mathcal{B} = \{w_0, w_1, \ldots\}$ using algorithm *Computation-of-Array-*$\mathcal{B}$.

For each $w_i \in \mathcal{B}$ *do*

(* Let v_j be the vertex attached to w_i *)

If $d(w_i, v_j) < d_{min}$, *then* set $d_{min} = d(w_i, v_j)$, and set $\alpha = w_i$.

We now have the following theorem stating the time complexity of our proposed algorithm for problem P1.

Theorem 2.1 *Algorithm Problem_P1(P) computes the location* α *in* $O(n)$ *time.*

2.2 Problem $P2$

The obvious extension of Problem P1 is placing two base stations for covering P where only one edge of P is available for placing the base stations. Without loss of generality, we may assume that the given edge e lies on the x-axis, it joins the vertices v_0 and v_1, and $x(v_0) < x(v_1)$. We use the term *constrained circle* to denote a circle whose center lies on e.

The problem of computing a single constrained circle $\mathcal{C}$ of minimum radius which covers P, can be computed in $O(n)$ time by considering the ψ_is' which share the edge e. The procedure is very much similar to that of Problem P1. Our objective to compute two constrained equal circles for covering the entire region P with minimum radius. We propose an $O(n)$ time algorithm for this problem.

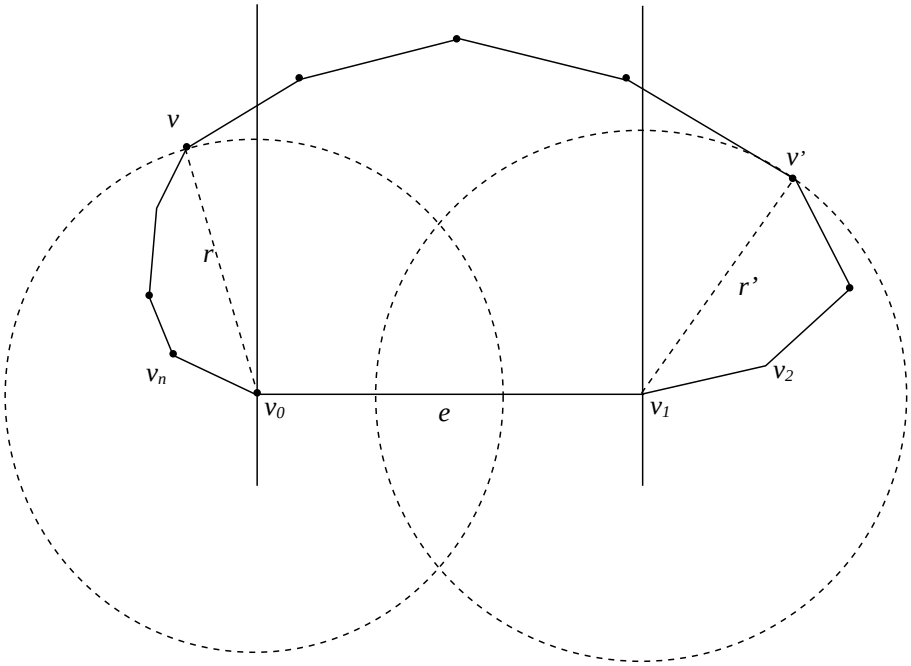

Figure 2.4: Illustrating the proof of Observation 2.4

Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two constrained equal circles of minimum radius that cover P. Let α and β be the centers of $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively, and $x(\alpha) < x(\beta)$. Now, we have the following simple observations.

Observation 2.4 *If v is the vertex of P such that $d(v, v_0) = Max\{d(v_i, v_0) \mid x(v_i) \leq x(v_0)\}$, and v' is the vertex such that $d(v', v_1) = Max\{d(v_i, v_1) \mid x(v_i) \geq x(v_1)\}$, then the radius of each circle $\mathcal{C}_1$ and $\mathcal{C}_2$ is greater than or equal to* $\max(d(v, v_0), d(v', v_1))$ *(see Figure 2.4).*

Observation 2.5 *If a vertex v is inside $\mathcal{C}_1$ but not inside $\mathcal{C}_2$ and a vertex v' is inside $\mathcal{C}_2$ but not inside $\mathcal{C}_1$, then $x(v) < x(v')$.*

Note that, the radius of $\mathcal{C}$ (the constrained circle of minimum radius enclosing P) is greater than or equal to the maximum among the radii of $\mathcal{C}_1$ and $\mathcal{C}_2$. If $\mathcal{C}$ passes through only one vertex of P, then its radius is exactly equal to the maximum radius among

the circles $\mathcal{C}_1$ and $\mathcal{C}_2$. This situation can be handled in linear time as mentioned for the problem P1. From now onwards, we assume that the circles $\mathcal{C}_1$ and $\mathcal{C}_2$ are smaller than the circle $\mathcal{C}$.

Let $v_k = \max_i\{v_i \notin \mathcal{C}_1\}$ and $v_{k'} = \min_i\{v_i \notin \mathcal{C}_2\}$. The vertices $\{v_0, v_{n-1}, v_{n-2}, \ldots, v_{k+1}\}$ are all in $\mathcal{C}_1$ and the vertices $\{v_1, v_2, \ldots, v_{k'-1}\}$ are all in $\mathcal{C}_2$. From Observation 2.5, we can conclude that $x(v_{k'}) < x(v_k)$. We first compute two constrained circles which cover the vertices $\{v_0, v_{n-1}, v_{n-2}, \ldots, v_{k+1}\}$ and $\{v_1, v_2, \ldots, v_{k'-1}\}$ respectively. These help in computing $\mathcal{C}_1$ and $\mathcal{C}_2$.

2.2.1 Algorithm

We will use $P_1(s)$ to denote the polygon with vertices $\{v_0, v_{n-1}, v_{n-2}, \ldots, v_s\}$, where s may assume values $0, n-1, n-2, \ldots, 1$. Similarly, $P_2(s)$ denotes the polygon with vertices $\{v_1, v_2, \ldots, v_s\}$, where s may assume values $1, 2, \ldots, n-2, n-1, 0$.

Lemma 2.7 *Let $\mathcal{C}^*$ and $\mathcal{C}^{**}$ be two minimum radius constrained circles enclosing the vertices of $P_2(s)$ and $P_2(s')$ respectively.*

1. *If $s' > s$ then the radius of $\mathcal{C}^*$ is less than or equal to the radius of $\mathcal{C}^{**}$.*
2. *If $s' > s$ then the x-coordinate of the center of $\mathcal{C}^*$ is greater than or equal to the x-coordinate of the center of $\mathcal{C}^{**}$.*
3. *Suppose both the circles pass through exactly two vertices of the polygon P and $s' > s$. If $\mathcal{C}^*$ passes through the vertices v_a and v_b with $a < b$, and $\mathcal{C}^{**}$ passes through the vertices v_z and v_w with $z < w$, then $z \leq a < b \leq w$.*

Proof : Suppose the radius of the circle $\mathcal{C}^*$ is r and it is centered at a point p on edge e. Since $\mathcal{C}^*$ encloses up to the vertex v_s, and p is on the left of v_1, then $x(v_s) \leq x(p)$, and also $x(v_i) < x(p)$ for all $i = s+1, s+2, \ldots, s'$. Since $\mathcal{C}^{**}$ encloses the points in $P_2(s)$ and also the points $v_{s+1}, v_{s+2}, \ldots, v_{s'}$, we have r less than or equal to the radius of $\mathcal{C}^{**}$. If r

is strictly less than the radius of $\mathcal{C}^{**}$, then at least one point v_i ($\in \{v_{s+1}, v_{s+2}, \ldots, v_{s'}\}$) is outside $\mathcal{C}^*$ and $\max_{i=s+1}^{s'} d(p, v_i)$ is greater than or equal to the radius of $\mathcal{C}^{**}$. Hence the lemma follows. □

Lemma 2.7 says that, if the minimum radius constrained circle $\mathcal{C}'$ covers the vertices of $P_1(s)$, and $s \geq k+1$, then its center is either at α or at the left side of α, where α is the center of $\mathcal{C}_1$. Similarly, for any $s' \leq k'-1$, if the minimum radius constrained circle $\mathcal{C}''$ encloses the vertices of $P_2(s')$, then its center is either at β or at the right side of β, where β is the center of $\mathcal{C}_2$.

Again from Lemma 2.1, we can conclude that the center of circle $\mathcal{C}''$ is either on edge e with x-coordinate $x(v_i)$, $1 \leq i \leq s'$ or it is at the intersection point of e with an edge of the farthest point Voronoi diagram of the vertices $\{v_1, v_2, \ldots, v_{s'}\}$ of $P_2(s')$. Let $\mathcal{A}_2(s') = \{u_0, u_1, u_2, \ldots, u_m\}$ be the set of intersection points of the furthest point Voronoi diagram of $P_2(s')$ with the edge e. With each element of $\mathcal{A}_2(s')$, the corresponding pair of vertices is attached as in problem P1. The center of the desired circle $\mathcal{C}''$ to cover the vertices of $P_2(s')$ is either the perpendicular projection of $v_{s'}$ on e or one of the members in $\mathcal{A}_2(s')$ for which the radius is minimum. Similarly $\mathcal{A}_1(s)$ is also computed for the polygon $P_1(s)$. Therefore we are interested about the intersection points of e with the edges of two farthest point Voronoi diagrams with the vertices of $P_1(s)$ and $P_2(s')$ respectively.

Initially while preprocessing the point set, we do not have any prior information of k and k' that determines $\mathcal{C}_1$ and $\mathcal{C}_2$. Initially, we start with $P_1(n-1) = \{v_0, v_{n-1}\}$. Given $\mathcal{A}_1(l)$ for the polygon $P_1(l)$, we incrementally compute $\mathcal{A}_1(l-1)$ for the polygon $P_1(l-1)$ by adding the next vertex v_{l-1}. The same procedure is followed for computing $\mathcal{A}_2(l)$ for all $l = 2, 3, \ldots, n-1, 0$. The following lemma describes an important relationship between $\mathcal{A}_2(l)$ and $\mathcal{A}_2(l+1)$, computed for $P_2(l)$ and $P_2(l+1)$ respectively.

Lemma 2.8 *Let $\mathcal{A}_2(l) = \{u_0, u_1, u_2, \ldots, u_m\}$, and $x(u_0) \leq x(u_1) \leq x(u_2) \leq \ldots \leq x(u_m)$. After introducing the next vertex v_{l+1}, $\mathcal{A}_2(l+1) = \{u'_0, u'_1, u'_2, \ldots, u'_t\}$, and $x(u'_0) \leq x(u'_1) \leq x(u'_2) \leq \ldots \leq x(u'_t)$. Then,*

(i) $m+1 \geq t$,
(ii) $u_0 = u'_0, u_1 = u'_1, \ldots, u_{t-1} = u'_{t-1}$.
Furthermore, if u_{t-1} *is generated by the perpendicular bisector of the line segment* $[v_i, v_{i+j}]$ $(i, j > 0)$*, then the point* u'_t *is generated due to intersection of edge* e *and the perpendicular bisector of line segment* $[v_{i+j}, v_{l+1}]$.

Proof : Follows from the property of the farthest point Voronoi diagram and from Lemmata 2.4 and 2.5. □

The same incremental result for $\mathcal{A}_1(l)$ is as follows:

Lemma 2.9 *Let* $\mathcal{A}_1(l) = \{u_0, u_1, u_2, \ldots, u_{m'}\}$*, and* $x(u_0) \geq x(u_1) \geq x(u_2) \geq \ldots \geq x(u_{m'})$*. After introducing the next vertex* v_{l-1}*,* $\mathcal{A}_1(l-1) = \{u'_0, u'_1, u'_2, \ldots, u'_{t'}\}$*, and* $x(u'_0) \geq x(u'_1) \geq x(u'_2) \geq \ldots \geq x(u'_{t'})$*. Then,*
(i) $m'+1 \geq t'$,
(ii) $u_0 = u'_0, u_1 = u'_1, \ldots, u_{t'-1} = u'_{t'-1}$.
Furthermore, if $u_{t'-1}$ *is generated by the perpendicular bisector of the line segment* $[v_i, v_{i-j}]$ $(i, j > 0)$*, then the point* $u'_{t'}$ *is generated due to intersection of edge* e *and the perpendicular bisector of line segment* $[v_{i-j}, v_{l-1}]$.

Lemma 2.10 *The total time complexity for computing* $\{\mathcal{A}_1(l), l = n-1, n-2, \ldots, 1\}$ *is* $O(n)$ *time. Another pass of same time complexity computes* $\{\mathcal{A}_2(l), l = 2, 3, \ldots, n-1, 0\}$.

Proof : Follows from Lemma 2.8, and similar arguments of the proof of Lemma 2.6. □

We now describe an iterative method for computing two minimum radii constrained circles $\mathcal{C}'$ and $\mathcal{C}''$ for covering all the vertices of P. Initially, we take $P_1(s) = \{v_0, v_{n-1}, v_{n-2}, \ldots, v_s\}$, where $x(v_i) \leq x(v_0)$ for all $i = n-1, n-2, \ldots, s$, and $P_2(s') = \{v_1, v_2, \ldots, v_{s'}\}$, where $x(v_i) \geq x(v_1)$ for all $i = 2, 3, \ldots, s'$. Let $\widehat{\mathcal{C}'}$ and $\widehat{\mathcal{C}''}$ are the minimum radii constrained circles for covering $P_1(s)$ and $P_2(s')$ respectively. Note that, the center of $\widehat{\mathcal{C}'}$

and $\widehat{\mathcal{C}''}$ are v_0 and v_1 respectively. Let the radii of $\widehat{\mathcal{C}'}$ and $\widehat{\mathcal{C}''}$ are r' and r'' respectively. At each iteration we do the following:

If $r' \geq r''$ and $\widehat{\mathcal{C}'}$ and $\widehat{\mathcal{C}''}$ do not cover all the vertices of P, then we execute the following steps to update $\widehat{\mathcal{C}''}$ such that it covers all the vertices of $P_2(s'+1) = \{v_1, v_2, \ldots, v_{s'}, v_{s'+1}\}$. Note that, the center of $\mathcal{C}''$ lies in the region $\mathcal{R}(v_{s'+1}) \cap e$. The center may be (i) the perpendicular projection of $v_{s'+1}$ on e, or (ii) a member of $\mathcal{A}_2(s'+1)$ which is computed as follows:

Prior to considering the vertex $v_{s'+1}$, let the circle $\widehat{\mathcal{C}''}$ passes through the vertex v_{s^*}, for some $s^* \leq s'$. If it passes through more than one vertex, then v_{s^*} is the rightmost one among them in the sequence $\{v_1, v_2, \ldots, v_{s'}\}$. We also have $\mathcal{A}_2(s^*) = \{u_0, u_1, \ldots, u_m\}$, where u_i is the intersection point of e and the perpendicular bisector of $[v_{s^i}, v_{s^{i-1}}]$ where $s^{i-1} < s^i$. We compute the point of intersection (say u) of e and perpendicular bisector of $(v_{s'+1}, v_{s^{i-j}})$ for $j = 0, 1, \ldots$ until we have $x(u) > x(u_m)$. Here, u is the desired center of circle $\widehat{\mathcal{C}''}$ as mentioned in Case (ii). We get $\mathcal{A}_2(s'+1)$ by removing all the members of $\mathcal{A}_2(s')$ whose x coordinate is less than $x(u)$, and finally add u in $\mathcal{A}_2(s'+1)$.

We set $r'' = d(u, v_{s'+1})$, where u is the center of $\widehat{\mathcal{C}''}$. Finally, if the updated $\widehat{\mathcal{C}''}$ covers some more vertices, namely $v_{s'+2}, v_{s'+3}, \ldots, v_{s'+j}$, then set $s^* = s'+1$, and $s' = s'+j$. In this case, we need not have to compute $\mathcal{A}_2(s'+2), \mathcal{A}_2(s'+3), \ldots, \mathcal{A}_2(s'+j)$.

If $r' < r''$ and $\widehat{\mathcal{C}'}$ and $\widehat{\mathcal{C}''}$ do not cover all the vertices of P, then we use $\mathcal{A}_1(s)$ to compute $\mathcal{A}_1(s-1)$, and follow the similar method as described above to update $\widehat{\mathcal{C}'}$ such that it covers all the vertices of $P_1(s-1) = \{v_0, v_{n-1}, v_{n-2}, \ldots, v_s, v_{s-1}\}$. After updating $\widehat{\mathcal{C}'}$, we compute its radius r'.

While considering the last uncovered vertex, a typical situation may arise. Let $\widehat{\mathcal{C}'}$ and $\widehat{\mathcal{C}''}$ be obtained after q iterative steps which cover all the vertices of P excepting only

one vertex, say v, and their radii are r' and r'' respectively. Without loss of generality, assume that $r' \leq r''$. In the $q+1$-th iteration, we include v in both $\widehat{\mathcal{C}'}$ and $\widehat{\mathcal{C}''}$. Let the corresponding radii are r^* and r^{**} respectively. If $r^* \geq r^{**}$, then $\mathcal{C}'$ is $\widehat{\mathcal{C}'}$ without including v, and $\mathcal{C}''$ is $\widehat{\mathcal{C}''}$ after including v. On the other hand, if $r^* < r^{**}$, then v is included in $\widehat{\mathcal{C}'}$. We also need to execute one more iteration as follows: let the $\widehat{\mathcal{C}''}$ is observed to pass through v_{s^*} in the q-th iteration and $r^* < r^{**}$. We include v_{s^*} in $\widehat{\mathcal{C}'}$, and observe its updated radius r^*. If $r^* < r''$, then $\mathcal{C}'$ is set to the updated $\widehat{\mathcal{C}'}$ in the $q+2$-th iteration (after including v_{s^*}). Otherwise, $\mathcal{C}'$ is set to the $\widehat{\mathcal{C}'}$ obtained in the $q+1$-th iteration. After fixing $\mathcal{C}'$, the remaining points will be covered by $\mathcal{C}''$.

Theorem 2.2 *Optimum size circles $\mathcal{C}'$ and $\mathcal{C}''$ with centers on edge e that cover all the vertices of polygon P can be located in $O(n)$ time.*

Proof : Result follows from above discussion and from Lemmata 2.7, 2.8 and 2.10. □

But there is no guarantee that the two circles $\mathcal{C}'$ and $\mathcal{C}''$ will cover the entire polygonal region.

Observation 2.6 *If $\mathcal{C}'$ and $\mathcal{C}''$ together do not cover the polygon P completely, then there exists exactly one edge e', ($e' \neq e$), which is not fully covered by $\mathcal{C}'$ and $\mathcal{C}''$ (see Figure 2.5(a)).*

If the situation stated in Observation 2.6 does not occur, then $\mathcal{C}_1 = \mathcal{C}'$ and $\mathcal{C}_2 = \mathcal{C}''$. Otherwise, the uncovered edge e' can be detected while computing $\mathcal{C}'$ and $\mathcal{C}''$. We now compute the optimum constrained circles $\mathcal{C}_1$ and $\mathcal{C}_2$.

Note that, $\mathcal{C}_1$ and $\mathcal{C}_2$ are of same size, and they must intersect at some point, say π, on edge e'. Let the equation of the line containing e' be $y = m \cdot x + c$. Let v_{s^*} be the vertex of P on the boundary of circle $\mathcal{C}'$, and having the least x-coordinate value, whereas $v_{s^{**}}$ be the vertex of P on the boundary of circle $\mathcal{C}''$, and having the maximum x-coordinate value among all such vertices. From Lemma 2.7, we can conclude that $\mathcal{C}_1$ passes through

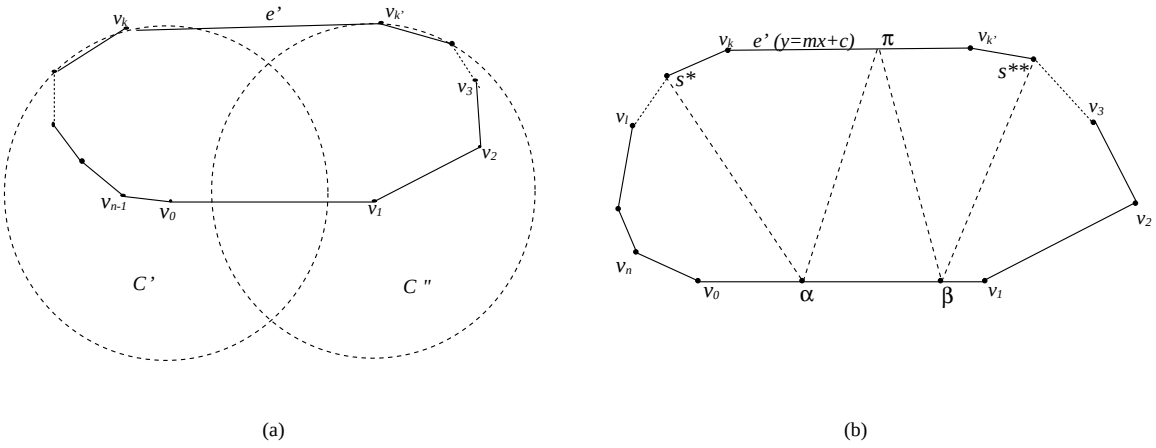

Figure 2.5: Illustrating the proof of Observation 2.6

either vertex v_{s^*} or some other vertex to the left side of v_{s^*}. Similarly, $\mathcal{C}_2$ passes through either vertex $v_{s^{**}}$ or some other vertex to the right side of $v_{s^{**}}$. Our objective is to locate the point π and hence the centers α and β. The following observation guides us to detect these points.

Observation 2.7 *The point π is the intersection of the perpendicular bisector of the line segment $[\alpha, \beta]$ with the edge e'.*

Assume that, the coordinate of π is (x_π, y_π) and the coordinate of α is $(x_\alpha, 0)$. From Observation 2.7, we can say that the coordinate of β is $(2x_\pi - x_\alpha, 0)$ (see Figure 2.5(b)). Initially, let us assume that $\mathcal{C}_1$ and $\mathcal{C}_2$ pass through v_{s^*} and $v_{s^{**}}$ respectively, whose coordinates are known. As both the circles pass through π, and have centers at α and β respectively, we can obtain a degree four polynomial equation involving x_π from the following constraints: (i) $d(v_{s^*}, \alpha) = d(\alpha, \pi)$, (ii) $d(v_{s^{**}}, \beta) = d(\beta, \pi)$, (iii) $y_\alpha = m \cdot x_\alpha + c$, and (iv) $x_\beta = 2x_\pi - x_\alpha$ (see Figure 2.5(b)). If $\alpha \notin \mathcal{R}(v_{s^*})$ then $\mathcal{C}_1$ passes through a vertex to the left of v_{s^*}. Without loss of generality, we choose v_{s^*-1}, and repeat the same procedure. Similarly, if $\beta \notin \mathcal{R}(v_{s^{**}})$, we apply the same procedure assuming that $\mathcal{C}''$ passes through the vertex $v_{s^{**}+1}$.

Theorem 2.3 *The minimum radii constrained circles $\mathcal{C}_1$ and $\mathcal{C}_2$ covering the region P can be computed in $O(n)$ time.*

2.3 Conclusion

In real life, finding the location for placing mobile base station avoiding the forbidden region is an important problem in facility location. Suppose P be a polygonal region which is forbidden in order to place a base station in the context of mobile communication. Here, we consider the problem of placing one base station at any point on the boundary of P and assign a range such that every point in the region is covered by that base station and the range assigned to that base station for covering the region is minimum among all such possible choices of base stations. Here we consider the forbidden region P as convex and a base station can be placed on the boundary of the region. We present optimum linear time algorithm for that problem. We also consider the placement problem for a pair of base stations on a specified side of the boundary such that the range assigned to those base stations in order to cover the region is minimum among all such possible choices of a pair of base stations on that side. We also present a linear time algorithm to solve this problem.

Chapter 3

Constrained Minimum Enclosing Circle Problem: a Query Version

In this chapter, we will study the query version of the problem mentioned in Chapter 2. Here, the point set P is given, and the query problem is to report the smallest enclosing circle of P whose center lies on a line segment L, supplied at the query time. During the preprocessing phase, we compute the convex hull of P, and then preprocess the convex polygon in $O(n \log n)$ time and using $O(n)$ space. The time complexity of our query algorithm is $O(\log^2 n)$. We will use this method for solving the following problem considered by Bose and Wang in [36] - *r simple polygons with a total of m vertices are given along with the point set P; the objective is to compute the smallest enclosing circle of P whose center lies in one of the r polygons.* This problem can be solved in $O(n \log n + m\log^2 n)$ time using our method in a much simpler way than [36]. It is an improved algorithm in the sense that the time complexity of its existing algorithm was $O((m+n)\log n + (n\sqrt{r}+m)\log m + n\sqrt{r} + r^{\frac{3}{2}} \log r)$ [36].

To the best of our knowledge, this is the first attempt on studying the query version of the smallest enclosing circle problem, where the center of the circle is constrained to lie on a query object.

3.1 Basic Results

It can be easily observed that the smallest circle enclosing the vertices of the convex hull of a point set P will also enclose all the points in P. So, from now onwards, we assume that the points in $P = \{p_1, p_2, \ldots, p_n\}$ are the vertices of a convex polygon in anticlockwise order. We first describe the role of farthest point Voronoi diagram in computing the smallest enclosing circle of P, and then use it to solve our online query problem. We reintroduce the notations for avoiding confusion with the notations used in Chapter 2.

Let $\mathcal{V}(P)$ denote the farthest point Voronoi diagram of P. It partitions the plane into n unbounded convex regions, namely $\mathcal{R}(p_1), \mathcal{R}(p_2), \ldots, \mathcal{R}(p_n)$, such that for any point $p \in \mathcal{R}(p_j)$, $d(p, p_j) > d(p, p_k)$ for all $k = 1, 2, \ldots, n$, and $k \neq j$. As mentioned in the earlier chapter, $d(.,.)$ denotes the Euclidean distance between a pair of points. $\mathcal{V}(P)$ can be computed in $O(n\log n)$ time and $O(n)$ space [131]. Given a query point q, this data structure can report the region $\mathcal{R}(p_i)$ containing q in $O(\log n)$ time.

Result 3.1 *[110] In the unconstrained situation, the smallest enclosing circle always passes through at least two points of P.*

Result 3.1 says that, the center c of the unconstrained smallest enclosing circle of P always lies on an edge e of $\mathcal{V}(P)$. In [110], an algorithm is proposed for computing c; this runs in $O(n\log n)$ time. We will use this information in designing the constrained smallest enclosing circle of P whose center c' lies on a given query line segment L. We first develop the algorithm for the case where the query object L is a line. Next, we show that a minor modification of that algorithm works when L is a line segment.

In our discussion, we will use C and C' to denote the unconstrained and constrained smallest enclosing circle respectively; c and c' denote the center of C and C' respectively (see Figure 3.1(a)). If ρ and ρ' denote the radius of C and C' respectively, then $\rho \leq \rho'$. As opposed to the fact that C must pass through at least two points of P (see Result

3.1), C' may pass through only one point in P. From Observation 2.2 and 2.3, we list distinct cases to be considered for computing c'.

For a convex polygon P,

(a) if C' passes through only a single point $p \in P$ then p is the farthest point from the query line L, and the perpendicular projection of p on the line L (denoted by p') lies inside $\mathcal{R}(p)$, and vice-versa. Here, $p' = c'$ is the center of the circle C'.

(b) if C' passes through exactly two points $p_1, p_2 \in P$ then its center c' is the intersection point of L with an edge e of $\mathcal{V}(P)$, where e is on the perpendicular bisector of p_1 and p_2.

(c) if C' passes through more than two points of P, then c' must be a vertex of $\mathcal{V}(P)$, which lies on the line L.

In order to test whether C' passes through a single point in P, we perform the following steps:

1. Identify the point $p \in P$ which is farthest from L.
2. Compute the perpendicular projection of p on the line L. Let this point be $\hat{c}$.
3. Identify the point $q \in P$ whose Voronoi region $\mathcal{R}(q)$ contains the point $\hat{c}$.
4. If $q = p$ then C' passes through p, and hence $c' = \hat{c}$.

The query time in step 1 is $O(\log n)$ [52]. Step 2 can be done in constant time. For step 3, the point location can be performed in $O(\log n)$ time using an $O(n)$ size data structure for $\mathcal{V}(P)$, which can be constructed in $O(n \log n)$ time [131]. Thus, we have the following lemma.

Lemma 3.1 *If C' passes through only a single point P, then its center c' can be computed in $O(\log n)$ time using a preprocessed data structure of size $O(n)$, which can be constructed in $O(n \log n)$ time.*

If the test in Step 4 fails, then C' passes through two or more vertices of P. Here, the center c' of the circle C' will be an intersection point of L and an edge of $\mathcal{V}(P)$. In the degenerate case, this point may be a vertex of $\mathcal{V}(P)$, indicating that C' passes through more than two points of P.

Let L intersect the edges $e_1, e_2, \ldots, e_m$ of $\mathcal{V}(P)$ in order, and $a_1, a_2, \ldots, a_m$ denote the corresponding intersection points. The above discussions say that c' will coincide with a member in the set $A = \{a_1, a_2, \ldots, a_m\}$. We will use $\rho(a_i)$ to denote the radius of the smallest enclosing circle of P with center at a_i.

Lemma 3.2 *The sequence* $\{\rho(a_1), \rho(a_2), \ldots, \rho(a_m)\}$ *is unimodal.*

Proof : Let $\rho(a_k) = \min_{i=1}^{m} \rho(a_i)$. We show that if $k < j \leq m-1$ then $\rho(a_k) < \rho(a_j) < \rho(a_{j+1})$. Similarly, it can be shown that if $1 \leq j' < k$, then $\rho(a_k) < \rho(a_{j'}) < \rho(a_{j'-1})$ (see Figure 3.1(a)).

Let us draw a line ℓ perpendicular to L at the point a_k on the Voronoi edge e_k. If e_k corresponds to the pair of points $p, p' \in P$, then p and p' lie on different sides of ℓ, and $d(p, a_k) = d(p', a_k)$. Let p (resp. p') be to the right (resp. left) of ℓ. Now, for any point α in the interval $[a_k, a_{k+1}]$ (on the line L), $\alpha \in \mathcal{R}(p')$ and $d(\alpha, p') > d(a_k, p')$. Thus, we have $\rho(a_{k+1}) > \rho(a_k)$.

Next we prove that $\rho(a_{k+1}) < \rho(a_{k+2})$. Let e_{k+1} be the Voronoi edge corresponding to p' and p''. Choose a point $\gamma \in [a_{k+1}, a_{k+2}]$ on the line L. Using the same argument, it can be shown that $\rho(a_{k+1}) = d(p', a_{k+1}) = d(p'', a_{k+1}) < d(p'', \gamma)$. Thus, $\rho(\gamma) > \rho(a_{k+1})$ since the circle enclosing P with center at γ has to enclose p''. Proceeding similarly, the result follows. □

It is already mentioned that each cell of $\mathcal{V}(P)$ is an unbounded convex region and the center c of the unconstrained smallest enclosing circle C lies on an edge of $\mathcal{V}(P)$. Thus, $\mathcal{V}(P)$ may be viewed as a directed tree $\mathcal{T}$ with c as root node, and all the Voronoi vertices are the internal nodes of $\mathcal{T}$ (see Figure 3.1(b)). The leaf nodes are hypothetical in the sense that these are at the open end-points of the half-line edges. In order to

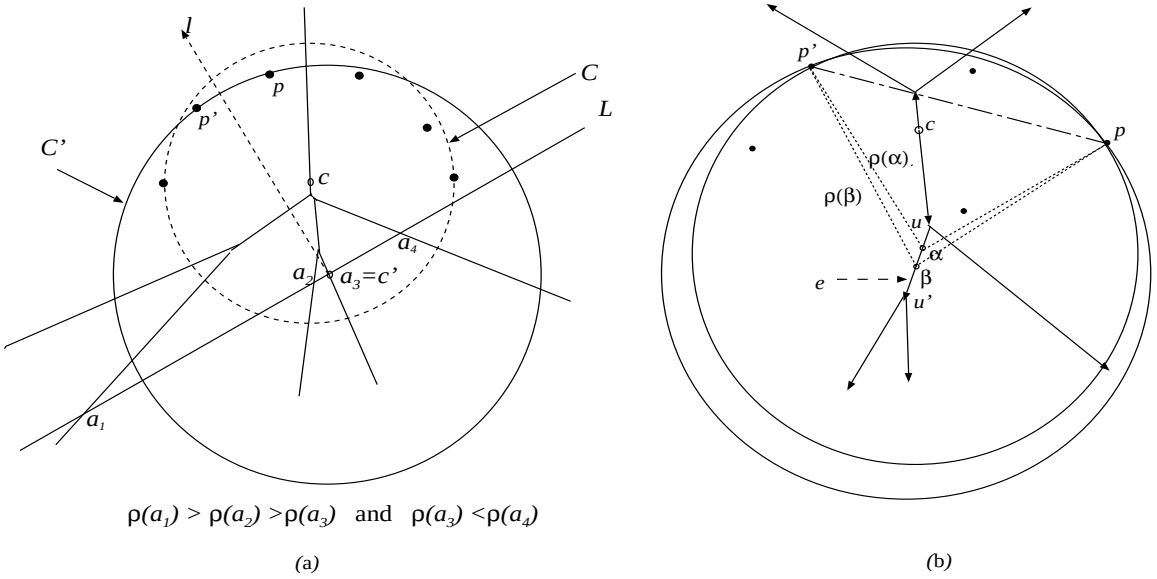

Figure 3.1: (a) Proof of Lemma 3.2, and (b) Proof of Lemma 3.3

clearly define the leaf-nodes of $\mathcal{T}$, we consider an axes-parallel square which contains all the vertices of $\mathcal{V}(P)$, and observe its intersections with all the unbounded edges of $\mathcal{V}(P)$. Let these be $E = \{\eta_1, \eta_2, \ldots, \eta_n\}$ in anticlockwise order along the boundary of the square, which will serve the role of leaf nodes in $\mathcal{T}$. For a point v on an edge of $\mathcal{T}$, we will use $\pi(v)$ to denote the directed path from c to v following the edges in $\mathcal{T}$. We will also use $depth(v)$ to denote the number of vertices of $\mathcal{T}$ on the path $\pi(v)$.

Lemma 3.3 *If T_u denotes the subtree rooted at an internal node $u \in \mathcal{T}$, then $\rho(v) > \rho(u)$ for each vertex $v \in T_u$, $u \neq v$.*

Proof : Let u' be a successor of u in $\mathcal{T}$. We need to prove $\rho(u) < \rho(u')$. Consider the directed edge $e = (u \rightarrow u')$. The regions adjacent to e are $\mathcal{R}(p)$ and $\mathcal{R}(p')$, where $p, p' \in P$. Thus, the edge e is the perpendicular bisector of the line segment $[p, p']$. At any point $\alpha \in e$, $\rho(\alpha) = d(\alpha, p) = d(\alpha, p')$. Moreover, for a pair of points $\alpha, \beta \in e$, if $\delta(\alpha, u) < \delta(\beta, u)$, then $\rho(\alpha) < \rho(\beta)$ because of the fact that both the triangles $\Delta pp'\alpha$ and $\Delta pp'\beta$ are isosceles, having the same base, and the other vertex is moving away

from the base along the perpendicular bisector of the base. Thus, $\Delta pp'\alpha$ is inside $\Delta pp'\beta$ (see Figure 3.1(b)). Thus, $\rho(u') > \rho(u)$. Applying the same technique recursively, the lemma follows. □

Lemma 3.3 says that as we go far from c along a path in $\mathcal{T}$, the ρ value of the nodes along that path increases monotonically. Thus, if a Voronoi region $\mathcal{R}$ contains c as its vertex, then $\rho(c)$ is minimum among the ρ values of all other vertices in $\mathcal{R}$. On the contrary, if $\mathcal{R}$ does not have c as one of its vertices, and $\rho(w) = min_{v \in \mathcal{R}}\ \rho(v)$, then for every vertex $v \in \mathcal{R}$, the path from c to v in $\mathcal{T}$ passes through node w. In addition, the following result is an implication of Lemma 3.3.

Corollary 3.3.1. *If L intersects a path $\pi(\eta_i)$ more than once, then the intersection point having minimum depth is the candidate for being c'.*

3.2 Constrained Smallest Enclosing Circle Problem with Center on Query Line

In our actual problem, the vertices of a convex polygon P are given in anticlockwise order. We propose two methods for this problem. The first one is simple but it takes $O(n\log n)$ space. It gives a clear idea about our method. In the next one, we adopt a complicated pointer structure to reduce the space complexity to $O(n)$. For both the methods, the preprocessing time complexity is $O(n\log n)$, and the query time complexity is $O(\log^2 n)$.

Suppose the farthest point Voronoi diagram $\mathcal{V}(P)$ of the polygon P is already computed, and is stored in the form of a directed tree $\mathcal{T}$ with root at c. Each node v is attached with its $\rho(v)$ value, which can be easily computed by observing the cell of $\mathcal{V}(P)$ in which v belongs. In addition, each node is attached with a *parent_pointer* which points to its predecessor in $\mathcal{T}$. The set E of leaf nodes in $\mathcal{T}$ are also stored in an array, and each element in E points to its corresponding element in $\mathcal{T}$.

3.2.1 Method-1

Preprocessing

In this method, we attach a few pointers with each node in $\mathcal{T}$ by performing a depth first search on the tree $\mathcal{T}$ in the preprocessing phase. Let v be a node with $depth(v) = m$. We attach a secondary structure B_v with node v which is an array of size $\lceil \log m \rceil$. These contain the address of the nodes at depth $\lceil \frac{m}{2} \rceil$, $\lceil \frac{3m}{4} \rceil$, $\lceil \frac{7m}{8} \rceil$, ... respectively along the path $\pi(v)$ in the mentioned order (see Figure 3.2). The computation of these pointers are described below.

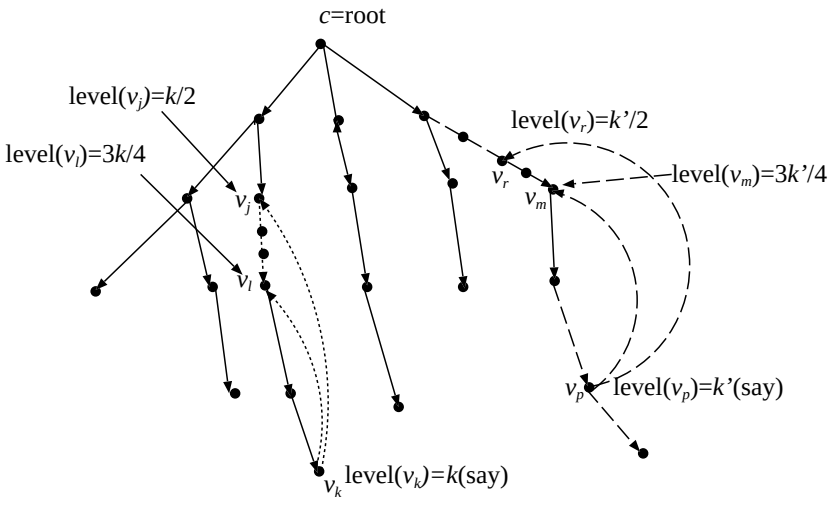

Figure 3.2: Intuitive idea of the secondary structures

We implement the stack required for the depth first search using an array. During the depth first search when the path follows a forward edge, we push it in the stack. While backtracking from a node v, we create the secondary structure B_v containing the pointers as mentioned above.

Lemma 3.4 *The Preprocessing phase can be completed in* $O(n\log n)$ *time and using* $O(n\log n)$ *space.*

Proof : The farthest point Voronoi diagram $\mathcal{V}(P)$ can be computed in $O(n\log n)$ time [131]. The computation of c needs $O(n)$ time [116]. Assigning the direction of the edges in $\mathcal{T}$ needs another $O(n)$ time. Finally, the depth first search needs $O(n)$ time for setting the pointers. While processing a node v during the backtrack, the creation of the array B_v of pointers needs $O(\log n)$ time in the worst case because $depth(v) \leq n$. The space complexity also follows from the same argument. □

Query answering

Given a query line L, we identify two paths $\pi(\eta_k), \pi(\eta_{k'})$, where $\eta_k, \eta_{k'} \in E$ are such that the leaf nodes of all the paths $\Pi_1 = \{\pi(\eta_{k+1}), \pi(\eta_{k+2}), \ldots, \pi(\eta_{k'-1})\}$ lie in the opposite side of c with respect to L, and those for $\Pi_2 = \{\pi(\eta_{k'}), \pi(\eta_{k'+1}), \ldots \pi(\eta_{k-1}), \pi(\eta_k)\}$ lie in the same side of c with respect to L. Note that, all the paths in Π_1 are intersected by the line L. The paths $\pi(\eta_k)$ and $\pi(\eta_{k'})$ can be identified in $O(\log n)$ time using the array E.

Lemma 3.5 *The center c' of the circle C' must be an intersection point of L with one of the paths in Π_1.*

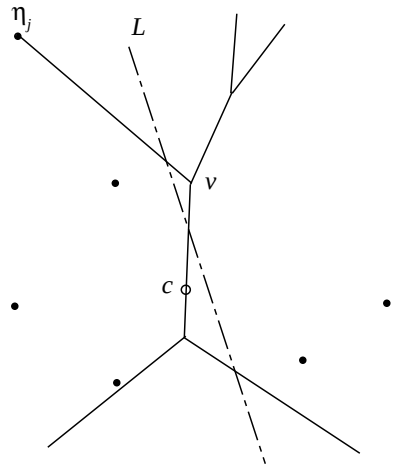

Figure 3.3: Illustration of Lemma 3.5

Proof : It is already mentioned in Observation 3.1 (b) & (c) that the point c' lies on a path in the set $\Pi_1 \cup \Pi_2$. We need to prove that if c' is observed on a path in Π_2, then it must also lie on some path in Π_1.

Consider an η_j such that the path $\pi(\eta_j) \in \Pi_2$ and c' lies on $\pi(\eta_j)$. By the definition of Π_2, the points c and η_j are in the same side of L. Since L intersects the path $\pi(\eta_j)$, there exists a vertex (say $v \in \mathcal{T}$) which lies in the opposite side of η_j. Now, consider the path $\pi(v)$, where c and v lie in different sides of L (see Figure 3.3). This path has also been intersected by L. Since the Voronoi regions are convex, there exists at least one path other than $\pi(\eta_j)$ which also passes through v. Consider one such path $\pi(\eta_i)$ $(\neq \pi(\eta_j))$. If it is in Π_1, then the proof is complete. Otherwise, an edge (v', v'') of $\pi(\eta_i)$ $(\in \Pi_2)$ will be intersected by L, and v' lies in the same side of v with respect to L. Proceeding with the same argument, we can reach to the leaf node of a path in Π_1. Thus the lemma follows. □

Lemma 3.6 *If the line L intersects a path $\pi(\eta_j)$ multiple times, and the center c' lies on $\pi(\eta_j)$, then c' will be the intersection point which is closest to c along the path $\pi(\eta_j)$.*

Proof : Follows from Corollary 3.3.1. □

Searching for an intersection of L with a path

The following lemma says that the links attached to the nodes in a path (say $\pi(\eta_j)$) helps us in searching for an intersection point of that path with the line L.

Lemma 3.7 *The worst case time complexity of searching for an intersection point of L with the path $\pi(\eta_j)$ is $O(\log n)$ in the worst case.*

Proof : Let $depth(\eta_j) = m$. We refer to the nodes on the path $\pi(\eta_j)$ as $v_1(= c), v_2, \ldots, v_m(= \eta_j)$. We first consider the first link in B_{v_m}. This points to the $v_{\lceil \frac{m}{2} \rceil}$-th node on that path. If c and $v_{\lceil \frac{m}{2} \rceil}$ are in the same side of L, then L has at least one intersection in the sub-path from $v_{\lceil \frac{m}{2} \rceil}$ and v_m; otherwise it has intersected $\pi(v_{\lceil \frac{m}{2} \rceil})$. In the former

case, we need to observe the next link in B_{v_m}; in the latter case, we need to consider $v_{\lceil \frac{m}{2} \rceil}$ and observe the first link in $B_{v_{\lceil \frac{m}{2} \rceil}}$. Proceeding this way, we can easily identify an edge on the path $\pi(\eta_j)$ of $\mathcal{T}$ which has been intersected by L. This needs $O(\log m)$ time. □

Searching for c' along L

Let $A = \{a_1, a_2, \ldots, a_\mu\}$ be a sequence of intersection points of L with the paths in Π_1, where a_j lies on path $\pi(\eta_j)$. Let $c' = a_i$. As the sequence of ρ values of the members of A is unimodal (see Lemma 3.2), both the sub-sequences $\{\rho(a_i), \rho(a_{i-1}), \ldots \rho(a_1)\}$ and $\{\rho(a_i), \rho(a_{i+1}), \ldots \rho(a_\mu)\}$ are monotonically increasing. So, we can identify c' by performing a binary search among the members in A. After observing a_j (by searching $\pi(\eta_j)$), we can decide whether $c' = a_j$ or have to move towards left (resp. right) from a_j along L by computing $\rho(q)$ and $\rho(q')$ for a pair of points q and q' at a distance ϵ from a_j on the line L, where ϵ is a very small positive constant which is less than $\min(d(a_j, a_{j-1}), d(a_j, a_{j+1}))$. Thus we have obtained one intersection point of L with the path $\pi(\eta_j)$. This may or may not be the point c'. This case happens if L intersects $\pi(\eta_j)$ more than once, $c' \in \pi(\eta_j)$, but our search identified some other intersection point. The following lemma says that c' will also lie on some other path, say $\pi(\eta_k)$. Finally, we will show that our algorithm will also explore the path $\pi(\eta_k)$, and will identify c'.

Lemma 3.8 *If L intersects the path $\pi(\eta_j)$ more than once, say at the points q_1 and q_2, and $depth(q_1) < depth(q_2)$, then (i) there is another intersection point q (may be q_1 itself) which lies on some other path, say $\pi(\eta_k) \in \Pi_1$, on which q_2 does not lie, and (ii) $\rho(q) \leq \rho(q_1) < \rho(q_2)$ (equality holds if $q = q_1$).*

Proof : Similar to the proof of Lemma 3.5. □

As the paths in Π_1 are non-crossing, the portion of the path of $\pi(\eta_k)$ from q to η_k lies in one side of the path $\pi(\eta_j)$. Hence this portion of $\pi(\eta_k)$ does not intersect with L (excluding the point q). This leads to the fact that the leaf nodes (members in E) corresponding to the paths in Π_1 are in the same order as the order of the intersection

points of L with the corresponding paths.

We perform binary search among the members in E. For each choice η_j, we compute the intersection point a_j on $\pi(\eta_j)$, and then check whether $a_j = c'$ or we move towards left (resp. right) on L by computing ρ values of two points on L in the ϵ neighborhood of a_j. Thus, we have the following theorem stating the complexity results of Method-1.

Theorem 3.1 *Method-1 correctly computes c' with $O(n \log n)$ preprocessing time and space complexities; the worst case query time complexity is $O(\log^2 n)$.*

Proof : The correctness of our proposed algorithm follows from Lemma 3.8. The preprocessing time and space complexity results follow from Lemma 3.4. The query time complexity follows from Lemma 3.7 and the fact that we may need to choose $O(\log n)$ elements among the members of Π_1 for searching the intersection point on the corresponding path. □

3.2.2 Method-2

In this method, we describe a different method of creating the secondary structure for each node in $\mathcal{T}$. This will reduce the space complexity of the problem to $O(n)$ keeping the preprocessing and query time complexities unchanged.

Revised secondary structure

Instead of keeping $O(\log n)$ pointers as the secondary structure of each node in $\mathcal{T}$, we store only two link fields, namely ptr_1 and ptr_2 with each node in $\mathcal{T}$. The ptr_1 pointer attached to a node $v \in \pi(\eta_j)$ indicates that while searching for an intersection of L with the path $\pi(\eta_j)$, if node v is reached, then such an intersection point must be observed in the path segment between the nodes v and $v.ptr_1$. The ptr_2 pointer of node v points to the middlemost node in the path segment between v and $v.ptr_1$. In order to set

these two pointers of the nodes in $\mathcal{T}$, we have to create a temporary data structure as mentioned below.

Let the maximum depth of a leaf node in $\mathcal{T}$ be m^*. We first create a temporary array A of size 2^α, where $2^{\alpha-1} < m^* \leq 2^\alpha$. Each entry of the array A consists of two fields, which are also named as ptr_1 and ptr_2 respectively. This array will be used to set the ptr_1 and ptr_2 pointers of each node in $\mathcal{T}$.

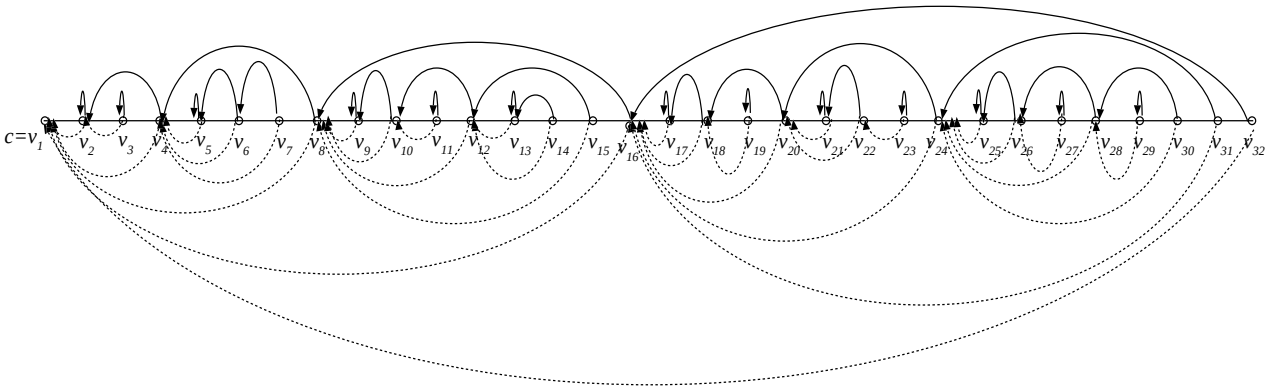

Figure 3.4: Secondary structures of the nodes in $\mathcal{T}$

Consider a path of length 2^α whose nodes are $c = v_1, v_2, v_3, \ldots, v_{2^\alpha} = \eta$. As mentioned earlier, the search in a path of $\mathcal{T}$ starts from its leaf node, L may intersect any edge on that path, and we will perform binary search on this path to identify its intersection with L. In order to perform the binary search, we set the pointers ptr_1 and ptr_2 as follows.

We will denote the portion of the path between a pair of nodes having node-indices a and b by the interval $[a, b]$. Thus, initially we have the interval $[2^0, 2^\alpha]$. In order to set the ptr_1 and ptr_2 pointers of each node on this path, we use a stack. Each element of this stack is a tuple of the form (I, i), where I is an interval of node-indices, and i is an integer $(0 \leq i \leq \log n)$. Initially, we push the tuple $([2^0, 2^\alpha], 0)$ onto the stack. Each time we pop an element $([a, b], i)$ from stack, and set ptr_1 and ptr_2 of $A[b - i]$ to a and $\frac{a+b}{2}$ respectively. The motivation is as follows: (i) we are searching in the interval

$[a, b]$, (ii) the pointers of $i-1$ nodes starting from b are already set, and now we will set the pointers of $A[b-i]$, and (iii) the split of the interval will be at node $\frac{a+b}{2}$.

Observe that, if $a+1 = b-i$, then ptr_2 of $A[b-i]$ is set to $b-i$ itself (see Figure 3.4), and no more split of this interval is needed. If $a+1 \neq b-i$, the interval $I = [a, b]$ gets split into two sub-intervals of node indices, namely $I_1 = [a, \frac{a+b}{2}]$ and $I_2 = [\frac{a+b}{2}, b]$. We push both the tuples $(I_1, 0)$ and $(I_2, i+1)$ in the stack. The process continues until the stack becomes empty. The creation of the array A is clearly illustrated in Figure 3.4. Here ptr_1 and ptr_2 pointers of that node are represented using dashed and solid edges respectively.

After the creation of the array A, we will set ptr_1 and ptr_2 of each node v by performing a depth first search as in Method-1. While popping a node v from the i-th position of the stack, we will set $v.ptr_1$ and $v.ptr_2$ by $A[i].ptr_1$ and $A[i].ptr_2$ respectively.

Lemma 3.9 *If a path $\pi(\eta)$ from c to a leaf node η in $\mathcal{T}$ is of length 2^β ($\beta \leq \alpha$), then the aforesaid link setting can report an intersection point of L with $\pi(\eta)$ in $O(\beta)$ time.*

Proof : We will use only ptr_2 to prove this lemma. The role of pointer ptr_1 will be clear in the next subsection.

Let $depth(\eta) = 2^\alpha$. As in Method-1, α pointers are available to η, which are stored in the ptr_2 fields of α nodes from η towards the root. So, here we can use these pointers moving upwards from η. In addition, while visiting these nodes, it has checked whether L has intersected the edges attached to these α nodes. The pointers which were present in the secondary structures of these $\alpha - 1$ nodes in Method-1, are not necessary in this method.

Suppose, after processing β ($\leq \alpha$) nodes starting from η, we could identify a node having depth $2^{\alpha-1} + 2^{\alpha-2} + \ldots + 2^{\alpha-\beta}$ such that L has intersected an edge in the sub-path from $v_{(2^{\alpha-1}+2^{\alpha-2}+\ldots+2^{\alpha-\beta-1})}$ to $v_{(2^{\alpha-1}+2^{\alpha-2}+\ldots+2^{\alpha-\beta})}$. The search interval is further pruned using the $\alpha-\beta$ pointers attached to $v_{(2^{\alpha-1}+2^{\alpha-2}+\ldots+2^{\alpha-\beta})}$. The process continues until the edge

of $\pi(\eta)$ is identified which has been intersected by L. The result follows from the fact that, the total number of link traversals in this process is at most α.

Next, consider the case where $depth(\eta) = 2^\beta$, $\beta < \alpha$. Here, observe that the similar link structure is maintained among the nodes in $v_1, v_2, \ldots, v_{2^\beta}$ (see Figure 3.4). Thus, the result follows. □

Searching in a path

Consider a path $\pi(\eta)$, where η is a leaf node and $depth(\eta) = m$, where $2^{\beta-1} < m \leq 2^\beta$, and $\beta \leq \alpha$. Without loss of generality, assume that the nodes are named as $v_1(= c), v_2, \ldots, v_m(= \eta)$, where $m = 2^{\beta-1} + 2^{\beta-2} + \ldots + 2^{\beta-j} + m'$, and $m' < 2^{\beta-j-1}$. Suppose the query line L intersects the edge $(v_\ell, v_{\ell+1}) \in \pi(\eta)$. Here, we need to consider two cases depending on whether (1) L intersects the path $\pi(\eta)$ between v_0 and $v_{2^{\beta-1}+2^{\beta-2}+\ldots+2^{\beta-j}}$, or (2) L intersects the path $\pi(\eta)$ below $v_{2^{\beta-1}+2^{\beta-2}+\ldots+2^{\beta-j}}$.

Case 1: We use the following notations to describe our search algorithm. Let $\theta = 2^{\beta-1} + 2^{\beta-2} + \ldots + 2^{\beta-k}$ and $\theta' = \theta - 2^{\beta-k} = 2^{\beta-1} + 2^{\beta-2} + \ldots + 2^{\beta-k-1}$.

Our search starts from v_m, and it consists of two major tasks: (i) identify θ such that v_ℓ lies in the interval of node-indices $[v_\theta, v_{\theta'}]$, and (ii) search for the intersection in the interval of node indices $[v_\theta, v_{\theta'}]$.

The task (i) is performed using ptr_1. Task (ii) is done using a binary search using ptr_2. Let us now observe Figure 3.4 to understand the search technique. Consider a typical instance where the path length is $m = 29$, and assume that L has intersected the edge $(v_\ell, v_{\ell+1})$, where $\ell = 2$. Thus, here $\theta = 16$ and $\theta' = 1$. The query involves the following link traversals (i) $v_{29} \rightarrow v_{28} \rightarrow v_{24} \rightarrow v_{16}$ using pointer ptr_1, and then (ii) apply binary search (as mentioned in Lemma 3.9) in the interval $[v_{16}, v_1]$ using ptr_2 to reach the node v_3. The intersection point of L with the edge (v_1, v_2) will then be identified.

Case 2: If L intersects an edges below $v_{2^{\beta-1}+2^{\beta-2}+\ldots+2^{\beta-j}}$, then it can also be detected in $O(\log n)$ time by expressing m' as that of m.

Thus, we have the following theorem stating the complexity results of the problem.

Theorem 3.2 *The preprocessing time and space complexities of Method-2 are* $O(n\log n)$ *and* $O(n)$ *respectively, and the worst case query time complexity is* $O(\log^2 n)$.

Proof : In addition to the preprocessing steps of Method-1, we had to create the array A containing the node indices on a path. This step needs $O(2^\alpha)$ time, if the length of the longest path m^* lies in $2^{\alpha-1} < m^* \leq 2^\alpha$. As $m^* \leq n$, the creation of array A needs $O(n)$ time in worst case. The secondary structure of each node of $\mathcal{T}$ consists of only ptr_1 and ptr_2, and during the depth first search, these can be set using the array A in $O(1)$ time. Thus both the preprocessing time and space complexity results follow.

In the worst case query time complexity analysis, we will study the search time on a path only. The location of c' among the possible paths remain same as in Method-1. The searching in a path in this method consists of two parts: (i) the number hops needed to reach from v_m to v_θ, and (ii) the time needed for the binary search to reach from v_θ to v_ℓ using ptr_2. Both the steps need $O(\alpha)$ hops which may be $O(\log n)$ in worst case. Since, we may need to inspect $O(\log n)$ paths in the worst case (see Lemma 3.7), the result follows. □

3.3 Constrained Smallest Enclosing Circle Problem with Center on Query Line Segment

We now consider that the query object L is a line segment. Let $\hat{L}$ be the line containing the line segment L. We apply Method-2 to identify the center $\hat{c} \in \hat{L}$ of the constrained smallest enclosing circle. If $\hat{c}$ is observed to be inside L, then report $c' = \hat{c}$. Otherwise, by Lemma 3.2, the center c' of the desired constrained smallest enclosing circle is one of the endpoints of L which is closest to $\hat{c}$ (with respect to the Euclidean distance). Finally, the radius of the desired smallest enclosing circle is $d(p, c')$, where $p \in P$ is the point whose corresponding Voronoi cell contains c'.

3.4 Constrained Smallest Enclosing Circle Problem with Center in a given Set of Polygons

Here the query objects are r simple polygons with a total of m edges. We first compute the farthest point Voronoi diagram $\mathcal{V}(P)$, to identify the center c of the unconstrained smallest enclosing circle. If it is inside one of these polygons, we report the answer. Otherwise, the center will be on the boundary of one of these polygons. For each edge (line segment), we compute the center of the constrained smallest enclosing circle with center on that edge, and report the radius of the smallest one. Thus, the overall time complexity becomes $O(n\log n + m\log^2 n)$, where $|P| = n$. In [35], it is mentioned that there may exist more than one such circle attaining the smallest radius. Our algorithm can report all these circles with the same time complexity.

3.5 Conclusion

In this chapter, we have considered the query version of the minimum enclosing circle problem. Here the query object is a line or a line segment, and the center of the desired minimum enclosing circle of the point set lies on that line/segment. The preprocessing and the query time complexities are $O(n \log n)$ and $O(\log^2 n)$ respectively. To the best of our knowledge this is the first attempt on handling the query version of minimum enclosing circle problem. Finally we have shown that our algorithm can be used for solving the generalized version of the constrained minimum enclosing circle problem, where the center lies inside one of the *r simple polygons with a total of m vertices*. Our technique also improves the time complexity of this constrained problem over its existing result [36], where this problem was initially posed.

Chapter 4

Guard Placement under L-visibility

In this chapter we consider a different guard placement problem. The guard can be viewed as a robot, which can sense any movement or sound in its *territory* and can be able to reach the source of the problem in short time. We address the location problem of the guard on the boundary of the given convex polygonal region P for guarding the exterior of the polygon under L-visibility. The term L-visibility is defined by Gewali et al. [72] as follows:

> Given a plane with polygonal obstacles, two points a and b are said to be *L-visible* if the length of the shortest path from a to b avoiding the obstacles is less than or equal to L.

The *external L-visibility* problem for a convex polygon P was first addressed in [72]. Here L is a constant, and the visibility power of the guard is L under the notion of L-visibility. The guard is assumed to be placed on the boundary of P, and the territory of a guard located at a point p is the region outside P, such that each point u in that region is L-visible from p. In other words, there is a path from p to u avoiding the interior of P whose length is less than or equal to L. The objective of our problem is to locate the position p on the boundary of P that maximizes the area of the territorial

region of p outside P. Gewali et al. [72] proposed an algorithm for locating the position of the guard on the boundary of the convex polygon P that maximizes the territory of the guard, where L is less than or equal to half the perimeter of P. The running time of the algorithm is $O(n)$, where n is the number of vertices of P. According to Gewali et al. [72], developing a polynomial time algorithm for the external L-visibility problem with L greater than half of the perimeter of convex polygon P is an open problem.

In this thesis, we address this open problem and present a linear time algorithm for the external L-visibility problem for any value of L.

4.1 Computation of the Territory of a Guard

Let P be a convex polygon, whose vertices $v_0, v_1, \ldots, v_{n-1}$ are in clockwise order; an edge connecting the vertices v_i and v_{i+1} is named as e_i. The length of an edge $e_i = (v_i, v_{i+1})$ is denoted by l_i. The angle between e_{i-1} and the extended line of e_i (as shown in Figure 4.1) is referred to as the *sector angle* at v_i, and will be denoted by θ_i. In addition, we will use the following notations throughout this chapter:

$\overline{pq}$ A straight line containing the points p and q.

$[p, q]$ A line segment joining points p and q.

$\overrightarrow{pq}$ A half-line that originates from the point p and passes through the point q.

$\widetilde{pq}$ A circular arc from point p to point q in clockwise direction.

$d(p, q)$ The Euclidean distance between the pair of points (p, q) without considering the obstacle P.

$\delta_c(p, q)$ The shortest distance from p to q in clockwise direction avoiding polygon P.

$\delta_a(p, q)$ The shortest distance from p to q in anticlockwise direction avoiding polygon P.

We will now consider the problem of placing a guard at a point p on the boundary of P, where the visibility power of the guard is L, and L is greater than half of the perimeter of P. Throughout this chapter, we will assume that L is supplied as the input parameter.

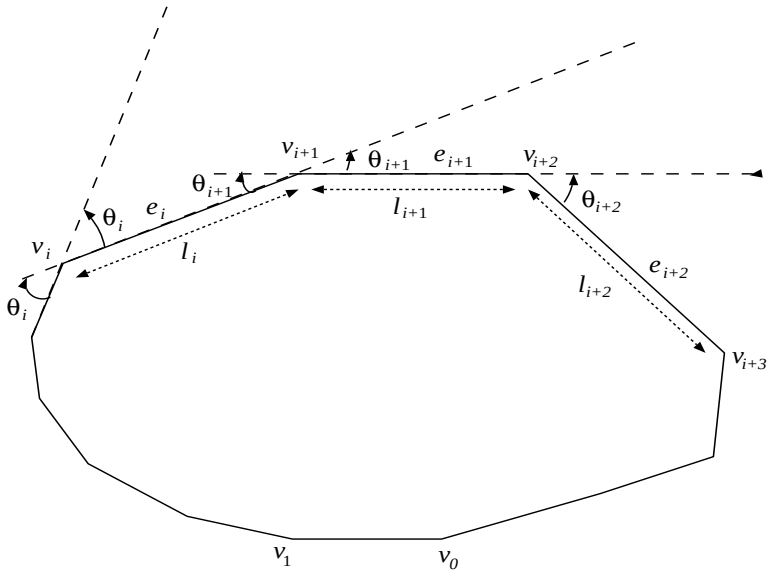

Figure 4.1: Vertices, edges and sector angles of a convex polygon

Definition 4.1 The *apex* point of a guard positioned at a point p (on the boundary of P) is a point a on the territory of p such that the length of the shortest paths from p to a avoiding obstacle P in both clockwise and anticlockwise directions are distinct and the length of both these paths are equal to L.

Note that, for a given placement of the guard on the boundary of P and for the given length L ($\geq$ half of the perimeter of P), the apex point a is unique. Both the paths from p to a pass through at least one vertex of P. The vertex of P, which is nearest to a along the path from p to a in clockwise (resp. anticlockwise) direction, is called the *supporting vertex* of a in clockwise (resp. anticlockwise) direction. Let v_c and v_a be two supporting vertices for the apex point a in clockwise and anticlockwise directions

respectively. We will use θ_c and θ_a to denote the sector angle at the supporting vertices v_c and v_a respectively. Let α_c be the angle formed by the line segment $[v_c, a]$ and the edge $e_c = (v_c, v_{c+1})$ (see Figure 4.2). Similarly, α_a is the angle formed by the line segment $[v_a, a]$ with the edge $e_{a-1} = (v_{a-1}, v_a)$. These two angles α_c and α_a are termed as *clockwise supporting angle* and *anticlockwise supporting angle* of the apex point a respectively. The observations below help to design an algorithm for finding the apex point of the guard positioned at a point p under L-visibility, for a fixed length L. Throughout this chapter, we will assume that L is an input parameter.

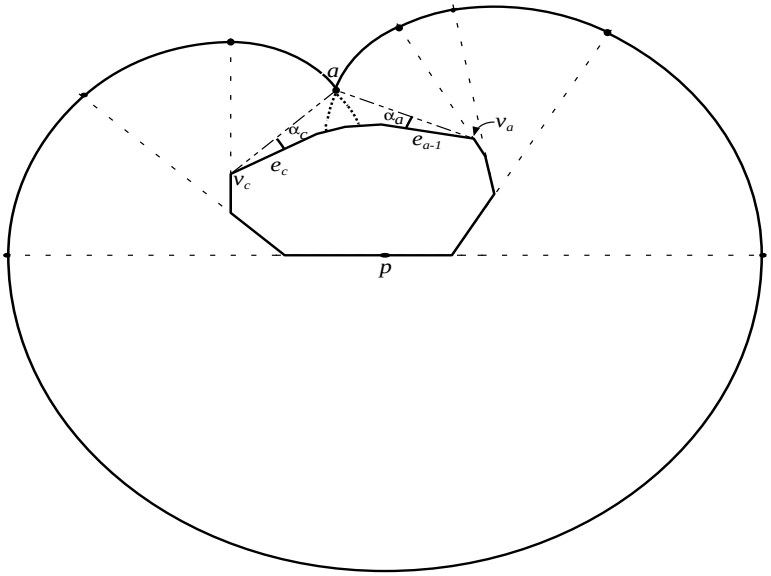

Figure 4.2: Territory of a guard located at p

Observation 4.1 *If v_c and v_a are the supporting vertices of an apex point a in clockwise and anticlockwise directions respectively, then $\alpha_c \leq \theta_c$ and $\alpha_a \leq \theta_a$.*

Proof : If $\theta_c < \alpha_c$, then there always exists a point u on the edge e_{c-1} which lies on the L-visibility path from p to a, and a is straight line reachable from u (see Figure 4.3). Thus, the length of the line segment $[u, a]$ must be less than the length of the path from u to a via the vertex v_c. Thus we have a contradiction, since v_c is the clockwise supporting vertex. Similarly, the other part of the lemma can be proved. □

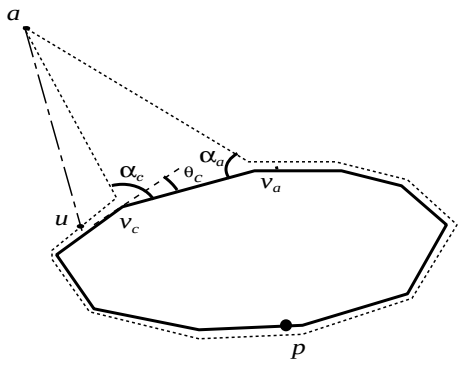

Figure 4.3: Illustrating the proof of Observation 4.1

Observation 4.2 *If the apex point a is given, then the location of the guard on the boundary of P that generates the apex point a, can be identified uniquely.*

Proof : Consider the vertices of the polygon P and the apex point a, and compute the convex hull of these points. Locate the point p on the convex hull such that the length of path from p to a in both clockwise and anticlockwise directions along the convex hull are same. Such a point p on the boundary of P must exist and it is unique. □

Observation 4.3 *(a) If v_c is the supporting vertex of an apex point a in clockwise direction and $\alpha_c = \theta_c$, then a can be identified uniquely.*

(b) If v_a is the supporting vertex of an apex point a in anticlockwise direction and $\alpha_a = \theta_a$, then a can be identified uniquely.

Proof : We will prove Part (a) of this observation. The proof of Part (b) is similar.

Let l be the half-line obtained by extending the edge $e_{c-1} = (v_{c-1}, v_c)$ from v_c onwards. Let it hit the lines containing the edges $e_c, e_{c+1}, \ldots, e_k$ at the points $\pi_c, \pi_{c+1}, \ldots, \pi_k$ respectively, and it does not hit the lines containing edges $e_{k+1}, e_{k+2}, \ldots e_{c-2}$. We assume $\pi_{k+1} = \infty$, and it corresponds to the edge e_{k+1}. Note that, the distances $d(v_c, \pi_c), d(v_c, \pi_{c+1}), \ldots$ are in increasing order.

Since $\alpha_c = \theta_c$, the apex point a must lie on l, and v_c is its clockwise supporting vertex. Its anticlockwise supporting vertex is computed as follows: Choose $\pi_c, \pi_{c+1}, \ldots, \pi_k$ in order. For each π_i, compute the convex hull of the vertices of P and the point π_i. If the half-perimeter of this convex polygon is greater than L, then a lies in $[\pi_{i-1}, \pi_i]$, and this implies that the anticlockwise supporting vertex of a is v_i. Otherwise, we repeat the same computation with π_{i+1}.

Let χ denote the length of the path from v_c to v_i along the boundary of P without passing through v_{i-1}. The sum of distances of a from v_c and v_i is $2L - \chi$, which is known since both L and P are given. Now, let us consider an ellipse with foci at v_c and v_i such that the sum of distances from its foci to any point on the boundary of the ellipse is equal $2L - \chi$. The intersection point of l with the ellipse is the position of the apex point a. □

Lemma 4.1 *The apex point for the guard positioned at a point p on the boundary of P can be located in $O(n)$ time.*

Proof : Given p, we can compute a point q on the boundary of P such that $\delta_c(p, q) = \delta_a(p, q) \leq L$. If $\delta_c(p, q) = \delta_a(p, q) = L$, then q is the apex point. Otherwise, we proceed as follows:

Step 1: Assume that q appear on edge $e_i = (v_i, v_{i+1})$.
Set $\mu = i$ and $\nu = i + 1$.

Step 2: Let $\overrightarrow{v_{\mu-1}v_\mu}$ and $\overrightarrow{v_{\nu+1}v_\nu}$ intersect at q'. Here three cases may arise:

Case 1 - $L > \delta_c(p, q') > \delta_a(p, q')$**:** Here set $\nu = \nu + 1$, and go to Step 2.

Case 2 - $L > \delta_a(p, q') > \delta_c(p, q')$**:** Here set $\mu = \mu - 1$, and go to Step 2.

Case 3 - $L < max(\delta_a(p, q'), \delta_c(p, q'))$**:** Here the clockwise and anticlockwise supporting vertices are determined, and these are v_μ and v_ν respectively.

Now, compute the apex point as described in the proof of Observation 4.3. □

Definition 4.2 If a guard is located at a point p on the boundary of P that generates an apex point with $\theta_c = \alpha_c$ or $\theta_a = \alpha_a$, then the point p is called *Steiner point*. Here θ_c and θ_a are the sector angles of the clockwise and anticlockwise supporting vertices v_c and v_a respectively, and α_c and α_a are the pair of supporting angles.

Lemma 4.2 *The number of possible Steiner points is at most* $2n$.

Proof : As L is given, for a given supporting vertex v_c in clockwise direction, we can uniquely determine the apex point a and the corresponding supporting vertex v_a in anticlockwise direction using the method described Observation 4.3(a). This defines a Steiner point (possible location of a guard) uniquely. Considering all the vertices as supporting vertices in clockwise direction, we can generate n Steiner points. Similarly, considering each vertex as the supporting vertex in anticlockwise direction, we can generate n Steiner points (see Observation 4.3(b)). □

Lemma 4.3 *All the Steiner points can be identified in* $O(n)$ *time.*

Proof : Let $v_c = v_1$, i.e., v_1 is the supporting vertex in clockwise direction; the sector angle θ_1 (at v_1) is equal to the supporting angle α_c in clockwise direction. As mentioned in Observation 4.3, we can identify the apex point and the location of guard in $O(n)$ time. This also reports the supporting vertex v_α in anticlockwise direction. We set two pointers pt_1 and pt_2 to point v_1 and v_α respectively. After computing v_α, we move pointer pt_1 to vertex v_2 for computing the next *Steiner point* whose supporting vertex is v_2, and the sector angle θ_2 is equal to the corresponding supporting angle. For this Steiner point, the supporting vertex in the anticlockwise direction must be v_α or some vertex in clockwise direction from v_α. That vertex can be identified by moving the pointer pt_2 along the vertices of P incrementally in clockwise direction. Note that, each move of the pointer pt_2 takes constant time. The process of moving pt_1 continues until pt_1 reaches v_n. During this process, pointer pt_2 moves from v_α to $v_{\alpha+1}, \ldots, v_n, v_1, \ldots v_\alpha$. Hence the lemma follows. □

We now describe the method of computing the area of the territory of a guard positioned at a point p on the boundary of P. Next, we will show that, in order to identify the optimal position of the guard, we need to consider only the Steiner points.

Let p be the position of the guard on an edge of P. If p coincides with a vertex v_j, then we assume that p lies on the edge $e_j = (v_j, v_{j+1})$. We will use d_i to denote the shortest distance of a vertex v_i from p along the boundary of P. Consider the vertices on the path from p to v_c in clockwise direction. At each vertex v_i, extend the edge (v_{i-1}, v_i) by a dotted line beyond v_i until it hits the boundary of the territory of p. Similarly, traverse the vertices that appear along the path from p to v_a in anticlockwise direction. At each vertex v_i on this path, extend the edge (v_{i+1}, v_i) by a dotted line beyond v_i until it hits the boundary of the territory of p. Finally, v_c and v_a are joined with the apex point a by a dotted line. See Figure 4.2 for the demonstration. Observe that, two dotted line segments of length $(L - d_i)$ are attached with each vertex v_i considered above. If $R_i(p)$ denote the portion of the territory of p bounded by two dotted lines at a vertex v_i, then

- for all the vertices v_i on the path from p to v_{c-1}, $area(R_i(p)) = (L - d_i)^2 \times \frac{\theta_i}{2}$,
- for all the vertices v_i on the path from p to v_{a+1}, $area(R_i(p)) = (L - d_i)^2 \times \frac{\theta_i}{2}$
- for the vertex v_c, $area(R_c(p)) = (L - d_c)^2 \times \frac{\theta_c - \alpha_c}{2}$.
- for the vertex v_a, $area(R_a(p)) = (L - d_a)^2 \times \frac{\theta_a - \alpha_a}{2}$.

We now split the territory into five zones. Let the dotted line $\overrightarrow{v_{c-1}v_c}$ meet the boundary of the territory at s and the dotted line $\overrightarrow{v_{a+1}v_a}$ meet the boundary of the territory at s' (see Figure 4.4).

Zone-I: Region bounded by arc $\widetilde{sa}$ and line segments $[s, v_c]$ and $[a, v_c]$. The area of this region is denoted by $A_1(p)$.

Zone-II: Region bounded by arc $\widetilde{as'}$ and line segments $[s', v_a]$ and $[a, v_a]$. The area of this region is denoted by $A_2(p)$.

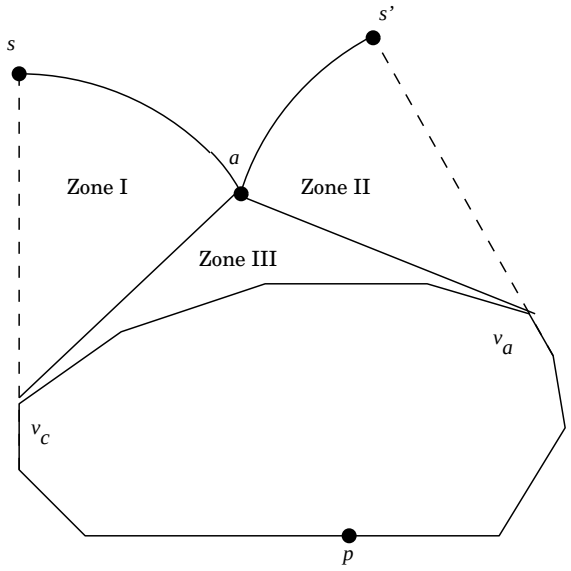

Figure 4.4: *Type I-III* zones for guard located at p

Zone-III: Region in the triangle $\triangle av_av_c$ outside polygon P. The area of this region is denoted by $A_3(p)$.

Zone-IV: $\cup_{v_i \in U} R_i(p)$, where U is the set of vertices that appear on the boundary of P from v_{c-1} to v_{a+1} in anticlockwise direction. The area of this region is denoted by $A_4(p)$.

Zone-V: The half-circle of radius L, centered at point p and to the other side of the polygon P with respect to the edge containing p. The area of this region is denoted by $A_5(p)$.

Thus the area of the entire territory of the point p can be written as $\mathcal{A}(p) = A_1(p) + A_2(p) + A_3(p) + A_4(p) + A_5(p) = \mathcal{A}_{123}(p) + A_4(p) + A_5(p)$, where $\mathcal{A}_{123}(p) = A_1(p) + A_2(p) + A_3(p)$.

Note that, if p is a Steiner point, then one of the two supporting angles must be equal to the corresponding sector angle. While computing each Steiner point, its one of the supporting vertices is known; thus, the other supporting vertex, and the apex point

can be determined using Observation 4.3. The areas of *Zone-I* and *Zone-II* can be determined immediately. The area of *Zone-III* can be obtained from a global counter μ maintained during the execution using a pair of pointers pt_1 and pt_2 as described in the proof of Lemma 4.3. While identifying a new Steiner point, if there is a move of pt_1 from v_c to v_{c+1}, $area(\Delta v_c v_{c+1} v_a)$ is subtracted from the existing value of μ, and if pt_2 moves from v_a to v_{a+1}, then $area(\Delta v_a v_{a+1} v_c)$ is added with μ.

Therefore we can conclude the following lemma.

Lemma 4.4 *The area of Zone-I, Zone-II and Zone-III for all the* $2n$ *Steiner points can be computed in O(n) time.*

Proof : Follows from the same technique for generating the *Steiner points* as described in proof of Lemma 4.3. □

Let S be a set consisting of of n vertices and at most $2n$ *Steiner points*. Lemma 4.5, stated below, indicates that the optimal position of the guard must be one of the points in S that maximizes the territorial region under L-visibility. Gewali et al. [72] conjectured that, one may have to investigate a set of $O(n)$ Steiner points in order to identify the optimal location of the guard.

Lemma 4.5 *An optimal location of the guard must be one of the points in* S *that maximize the area of the territory under L-visibility.*

Proof : Let p_m be a point on an edge e of P which is not in S. We can always find two points p_l and p_r to the left and right side of p_m respectively on the same edge e such that p_l and p_r are at equal distance from p_m, and there is no point of S in the interval $[p_l, p_m]$. Let a_l, a_m and a_r be the apex points of p_l, p_m and p_r respectively. Observe that, for such a choice of p_l and p_r, the supporting vertices of the apex points a_l, a_m and a_r will be the same, and the set of vertices of P that lie on the boundary of the *Zone-III* of the apex points a_l, a_m and a_r will also remain same. We prove that p_m cannot be an optimum location for placing the guard, by showing $area(\mathcal{A}(p_l))+area(\mathcal{A}(p_r)) \geq 2\times area(\mathcal{A}(p_m))$.

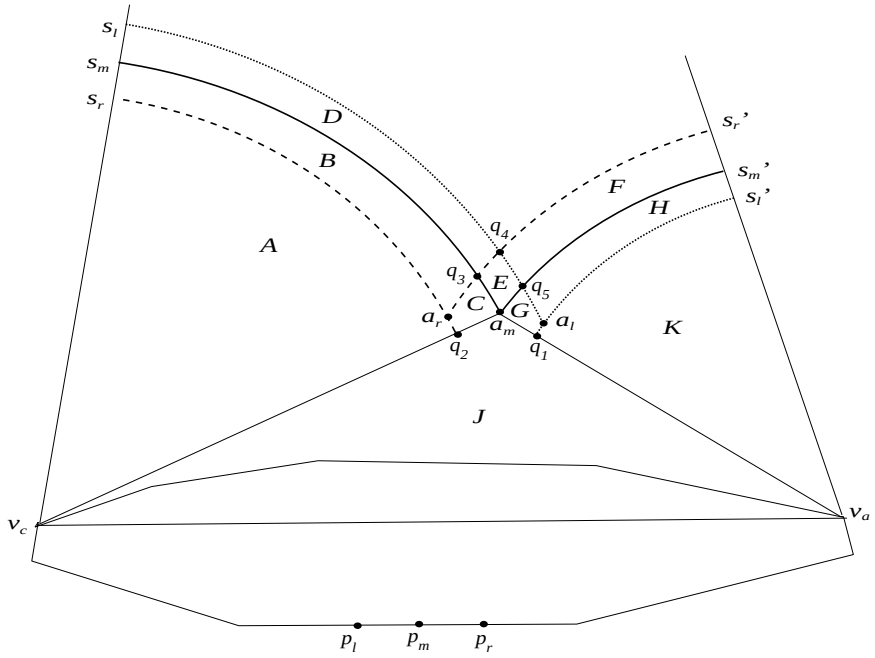

Figure 4.5: Illustrating the proof of Lemma 4.5

It is easy to observe that $A_5(p_l)+A_5(p_r) = 2A_5(p_m)$. Let v_i be a vertex on the path from v_{c-1} to v_{a+1} in anticlockwise direction. The distance of v_i from p_m, p_l and p_r are d_i, $d_i-\epsilon$ and $d_i+\epsilon$ respectively. Simple algebraic inequality says that $2(L-d_i)^2\frac{\theta_i}{2} < ((L-d_i-\epsilon)^2+(L-d_i+\epsilon)^2)\frac{\theta_i}{2}$. Adding over all such vertices, we have $A_4(p_l)+A_4(p_r) > 2A_4(p_m)$.

We now prove that $\mathcal{A}_{123}(p_l) + \mathcal{A}_{123}(p_r) > 2\mathcal{A}_{123}(p_m)$.

Since v_c and v_a are the supporting vertices of the apex points a_l, a_m and a_r corresponding to the positions of the guard at p_l, p_r and p_m respectively, the points a_l, a_m and a_r lie on the ellipse with foci at v_c and v_a. We extend the edge $(v_{c-1}v_c)$ from v_c onwards; the half-line $\overrightarrow{v_{c-1}v_c}$ hits the territory of p_l, p_m and p_r at s_l, s_m and s_r respectively. Similarly, the half-line $\overrightarrow{v_{a+1}v_a}$ hits the territory of p_l, p_m and p_r at s'_l, s'_m and s'_r respectively.

Note that, apex point for guard located at $p \in \{p_l, p_m, p_r\}$ is not in the *Zone-III* of the guard located at a point other than p on the boundary of P. We further partition the union of *Zone-I*, *Zone-II* and *Zone-III* generated by the guards located at p_l, p_m and p_r as follows (see Figure 4.5).

A: Region bounded by the arc $\widetilde{s_r q_2}$ and the line segments $\overline{s_r v_c}$ and $\overline{q_2 v_c}$.

B: Region bounded by the arcs $\widetilde{s_r a_r}$, $\widetilde{s_m q_3}$, $\widetilde{a_r q_3}$ and the line segment $\overline{s_r s_m}$.

C: Region bounded by the arcs $\widetilde{a_r q_3}$, $\widetilde{q_3 a_m}$, $\widetilde{a_r q_2}$ and the line segment $\overline{q_2 a_m}$.

D: Region bounded by the arcs $\widetilde{s_l q_4}$, $\widetilde{s_m q_3}$, $\widetilde{q_3 q_4}$ and the line segment $\overline{s_l s_m}$.

E: Region bounded by the arcs $\widetilde{q_3 q_4}$, $\widetilde{q_4 q_5}$, $\widetilde{q_3 a_m}$, and $\widetilde{a_m q_5}$.

F: Region bounded by the arcs $\widetilde{q_4 s'_r}$, $\widetilde{q_5 s'_m}$, $\widetilde{q_4 q_5}$ and the line segment $\overline{s'_r s'_m}$.

G: Region bounded by the arcs $\widetilde{a_m q_5}$, $\widetilde{q_5 a_l}$, $\widetilde{q_1 a_l}$ and the line segment $\overline{a_m q_1}$.

H: Region bounded by the arcs $\widetilde{q_5 s'_m}$, $\widetilde{q_5 a_l}$, $\widetilde{a_l s'_l}$ and the line segment $\overline{s'_m s'_l}$.

K: Region bounded by the arc $\widetilde{q_1 s'_l}$ and the line segments $\overline{q_1 v_a}$, $\overline{s'_l v_a}$.

Thus, we have
$\mathcal{A}_{123}(p_l) = area(A) + area(B) + area(C) + area(D) + area(E) + area(G) + area(K) + area(J)$,
$\mathcal{A}_{123}(p_m) = area(A) + area(B) + area(C) + area(G) + area(H) + area(K) + area(J)$, and
$\mathcal{A}_{123}(p_r) = area(A) + area(C) + area(E) + area(G) + area(F) + area(H) + area(K) + area(J)$.

Note that, (i) a_l and a_m lie on the circumference of an ellipse, where a_l appears in the anticlockwise direction from a_m, and (ii) $\widetilde{s_l a_l}$ is a circular arc whose center is at v_c and radius greater than the length of the line segment $[v_c, a_m]$. Thus the half-line $\overrightarrow{v_c a_m}$ intersects the arc $\widetilde{s_l a_l}$ and splits the region G. Therefore, $area(A \cup B \cup C \cup D \cup E \cup G)$ is greater than the area bounded by the arc $\widetilde{s_l a_l}$ with lines $\overrightarrow{s_l v_c}$ and $\overrightarrow{a_m v_c}$. Again if the distance from p_m to v_c is d and $\theta = \angle s_m v_c a_m$, then $2(L-d)^2 \frac{\theta}{2} < ((L-d-\epsilon)^2 + (L-d+\epsilon)^2)\frac{\theta}{2}$. Hence, we can conclude that $area(A) + area(A \cup B \cup C \cup D \cup E \cup G) > 2.area(A \cup B \cup C)$ and $area(K) + area(C \cup E \cup F \cup G \cup H \cup K) > 2.area(G \cup H \cup K)$. Adding these two inequalities, we have $\mathcal{A}_{123}(p_l) + \mathcal{A}_{123}(p_r) > 2\mathcal{A}_{123}(p_m)$. □

Using the above observations, we can identify a point p (the position of the guard) for which the area of the territory is maximum, by computing $\mathcal{A}(p)$ for every point $p \in S$. For each point $p \in S$, the time required for calculating $\mathcal{A}(p)$ (or more specifically, $A_4(p)$) is $O(n)$. Since the cardinality of the set S is $O(n)$, we can detect the optimum location of the guard in $O(n^2)$ time. In the next section, we will describe an incremental way of computing $A_4(p)$, which leads to a linear time algorithm for computing the optimum location of the guard.

4.2 Efficient Computation of the area of Zone-IV

It is already mentioned that the area of Zone-V is same for all the Steiner points $p \in S$. By Lemmata 4.4 and 4.5, $A_{123}(p)$ can be computed for all $p \in S$ in $O(n)$ time. In this section, we describe an accelerated method of computing the area of Zone-IV for all the members in S.

We redefine P assuming all the $3n$ points in the set S as the vertices of P. Therefore, the vertices of P are $v_0, v_1, \ldots, v_{3n-1}$ in clockwise direction with edges $e_i = (v_i, v_{i+1})$ having length l_i. The angle of the polygon at the original Steiner vertices are equal to π and the corresponding sector angles are equal to 0. Since L is strictly greater than half of the perimeter of P, the apex point cannot lie on the boundary of the polygon for any position of the guard on the boundary of P. Thus, none of the supporting vertices can be an original *Steiner point*. It is already proved in Lemma 4.5 that, the optimal position of the guard can be one of the vertices of the redefined polygon P. From now onwards, by vertices we mean the vertices of the redefined polygon.

We assume that each vertex v_i of P is attached with (i) the sector angle (θ_i) of v_i, (ii) length (l_i) of the edge $e_i = (v_i, v_{i+1})$, (iii) the supporting vertices v_c and v_a when the guard is positioned at v_i, and (iv) the following six parameters:

$sum_1^i = l_{i+1} + l_{i+2} + \cdots + l_{c-1}$,
$sum_2^i = l_{i-1} + l_{i-2} + \cdots + l_{a+1}$,

$\Theta_1^i = \theta_{i+1} + \cdots + \theta_{c-1}$,
$\Theta_2^i = \theta_{i-1} + \theta_{i-2} + \cdots + \theta_{a+1}$,
$\Gamma_1^i = l_{i+1}(\theta_{i+2} + \theta_{i+3} + \cdots + \theta_{c-1}) + l_{i+2}(\theta_{i+3} + \theta_{i+4} + \cdots + \theta_{c-1}) + \cdots + l_{c-2}(\theta_{c-1})$
$\Gamma_2^i = l_{i-1}(\theta_{i-1} + \theta_{i-2} + \cdots + \theta_{a+1}) + l_{i-2}(\theta_{i-2} + \theta_{i-3} + \cdots + \theta_{a+1}) + \cdots + l_{a+1}(\theta_{a+1}))$,
where the indices in the above expressions are written in modulo n form. These parameters will be used for the computation of $A_4(v_i)$.

For each vertex v_i of P, these parameters can be computed in an amortized $O(1)$ time as follows:

The time needed for computing the parameters attached to v_1 is $O(n)$. While computing the parameters of v_{i+1}, we assume that the parameters of v_i is already available. The sum_1^{i+1}, sum_2^{i+1}, Θ_1^{i+1} and Θ_2^{i+1} can be easily computed observing its corresponding supporting vertices in $O(1)$ time, since at most one of the two supporting vertices of v_{i+1} may differ from that of v_i. In order to compute Γ_1^{i+1} and Γ_2^{i+1}, we need to consider three cases depending on whether (i) clockwise supporting vertex of v_{i+1} has changed to v_{c+1}, where v_c is the clockwise supporting vertex of v_i, or (ii) anticlockwise supporting vertex of v_{i+1} has changed to v_{a+1}, where v_a is the anticlockwise supporting vertex of v_i, or (iii) the supporting vertices of v_{i+1} are same as those of v_i.

Case (i) Here $\Gamma_1^{i+1} = \Gamma_1^i - l_{i+1}(\theta_{i+2} + \theta_{i+3} + \cdots + \theta_{c-1}) + \theta_c(l_{i+2} + l_{i+3} + \cdots + l_{c-1}) = \Gamma_1^i - l_{i+1}(\Theta_1^i - \theta_{i+1}) + \theta_c(sum_1^i - l_{i+1})$, and
$\Gamma_2^{i+1} = \Gamma_2^i + l_i(\theta_i + \theta_{i-1} + \cdots + \theta_{a+1}) = \Gamma_2^i + l_i(\Theta_2^i + \theta_i)$,

Case (ii) Here $\Gamma_1^{i+1} = \Gamma_1^i - l_{i+1}(\theta_{i+2} + \theta_{i+3} + \cdots + \theta_{c-1}) = \Gamma_1^i - l_{i+1}(\Theta_1^i - \theta_{i+1})$, and
$\Gamma_2^{i+1} = \Gamma_2^i + l_i(\theta_i + \theta_{i-1} + \cdots + \theta_{a+2}) - \theta_{a+1}(l_{i-1} + l_{i-2} + \cdots + l_{a+1}) = \Gamma_2^i + l_i(\Theta_2^i + \theta_i - \theta_{a+1}) - \theta_{a+1}sum_2^i$,

Case (iii) Here $\Gamma_1^{i+1} = \Gamma_1^i - l_{i+1}(\theta_{i+2} + \theta_{i+3} + \cdots + \theta_{c-1}) = \Gamma_1^i - l_{i+1}(\Theta_1^i - \theta_{i+1})$, and
$\Gamma_2^{i+1} = \Gamma_2^i + l_i(\theta_i + \theta_{i-1} + \cdots + \theta_{a+1}) = \Gamma_2^i + l_i(\Theta_2^i + \theta_i)$.

We now concentrate on the computation of $A_4(v_i)$ (the area of Zone-IV) for all vertices v_i of P. Suppose $A_4(v_i)$ is already computed (considering v_c and v_a as its supporting

vertices), and is equal to

$A_4(v_i) = \frac{1}{2}[\{\theta_{i+1}(L-l_i)^2 + \theta_{i+2}(L-l_i-l_{i+1})^2 + \theta_{i+3}(L-l_i-l_{i+1}-l_{i+2})^2 + \cdots + \theta_{c-1}(L-l_i-l_{i+1}-l_{i+2}-\cdots-l_{c-2})^2\} + \{\theta_i(L)^2 + \theta_{i-1}(L-l_{i-1})^2 + \theta_{i-2}(L-l_{i-1}-l_{i-2})^2 \cdots + \theta_{a+1}(L-l_{i-1}-l_{i-2}-\cdots-l_{a+1})^2\}]$

While computing $A_4(v_{i+1})$, if $v_{c'}$ and $v_{a'}$ are the supporting vertices, then

$A_4(v_{i+1}) = \frac{1}{2}[\{\theta_{i+2}(L-l_{i+1})^2 + \theta_{i+3}(L-l_{i+1}-l_{i+2})^2 + \theta_{i+4}(L-l_{i+1}-l_{i+2}-l_{i+3})^2 + \cdots + \theta_{c'-1}(L-l_{i+1}-l_{i+2}-l_{i+3}-\cdots-l_{c'-2})^2\} + \{\theta_{i+1}(L)^2 + \theta_i(L-l_i)^2 + \theta_{i-1}(L-l_i-l_{i-1})^2 + \theta_{i-2}(L-l_i-l_{i-1}-l_{i-2})^2 + \cdots + \theta_{a'+1}(L-l_i-l_{i-1}-l_{i-2}-\cdots-l_{a'+1})^2\}].$

$= \frac{1}{2}[\{\theta_{i+1}(L-l_i)^2 + \theta_{i+2}(L-l_i-l_{i+1})^2 + \theta_{i+3}(L-l_i-l_{i+1}-l_{i+2})^2 + \cdots + \theta_{c'-1}(L-l_i-l_{i+1}-l_{i+2}-l_{i+3}-\cdots-l_{c'-2})^2\} + \{\theta_i(L)^2 + \theta_{i-1}(L-l_{i-1})^2 + \theta_{i-2}(L-l_{i-1}-l_{i-2})^2 + \theta_{i-3}(L-l_{i-1}-l_{i-2}-l_{i-3})^2 + \cdots + \theta_{a'+1}(L-l_{i-1}-l_{i-2}-l_{i-3}-\cdots-l_{a'+1})^2\}] - \frac{1}{2}[\theta_{i+1}(L-l_i)^2 + l_i^2(\theta_{i+2}+\theta_{i+3}+\cdots+\theta_{c'-1}) - 2l_i\{\theta_{i+2}(L-l_{i+1}) + \theta_{i+3}(L-l_{i+1}-l_{i+2}) + \cdots + \theta_{c'-1}(L-l_{i+1}-l_{i+2}-\cdots-l_{c'-2})\}] + \frac{1}{2}[\theta_{i+1}L^2 + l_i^2(\theta_i + \theta_{i-1} + \cdots + \theta_{a'+1}) - 2l_i\{\theta_i L + \theta_{i-1}(L-l_{i-1}) + \cdots + \theta_{a'+1}(L-l_{i-1}-l_{i-2}-\cdots-l_{a'+1})\}].$

$= \frac{1}{2}[T_1 - T_2]$, where
$T_1 = \{\theta_{i+1}(L-l_i)^2 + \theta_{i+2}(L-l_i-l_{i+1})^2 + \theta_{i+3}(L-l_i-l_{i+1}-l_{i+2})^2 + \cdots + \theta_{c'-1}(L-l_i-l_{i+1}-l_{i+2}-l_{i+3}-\cdots-l_{c'-2})^2\} + \{\theta_i(L)^2 + \theta_{i-1}(L-l_{i-1})^2 + \theta_{i-2}(L-l_{i-1}-l_{i-2})^2 + \theta_{i-3}(L-l_{i-1}-l_{i-2}-l_{i-3})^2 + \cdots + \theta_{a'+1}(L-l_{i-1}-l_{i-2}-l_{i-3}-\cdots-l_{a'+1})^2\}$, and
$T_2 = \theta_{i+1}(L-l_i)^2 + l_i^2\Theta_1^{i+1} - 2l_i\{\Theta_1^{i+1}(L-l_{i+1}) - \Gamma_1^{i+1}\} - \theta_{i+1}L^2 - l_i^2\Theta_2^{i+1} + 2l_i\{(L+l_i)\Theta_2^{i+1} - \Gamma_2^{i+1}\}$.

Observe that, both T_1 and T_2 can be computed in $O(1)$ time, but while computing T_1, we need to consider the following three cases: (i) $v_{c'} = v_{c+1}$ and $v_{a'} = v_a$, (ii) $v_{c'} = v_c$ and $v_{a'} = v_{a+1}$, and (iii) $v_{c'} = v_c$ and $v_{a'} = v_a$.

In Case (i), $T_1 = A_4(v_i) + \theta_c(L - sum_1^i - l_i)^2$,
in Case (ii), $T_1 = A_4(v_i) - \theta_{a+1}(L - sum_2^i)^2$, and in Case (iii), $T_1 = A_4(v_i)$ (here the necessary changes are captured in T_2). Thus we have the following lemma:

Lemma 4.6 *The total time required for computing $A_4(v_i)$ for all the vertices v_i in P is $O(n)$.*

This leads to the final result of this chapter as follows:

Theorem 4.1 *The location of the guard on the boundary of the given convex polygon that yields the maximum territorial region can be computed in $O(n)$ time.*

Proof : Follows from Lemmata 4.3, 4.4, 4.5 and 4.6. □

4.2.1 Conclusion

We consider the problem of finding the location of a guard on the boundary of a convex polygon P such that the external area that can be covered under *L-visibility* is maximized. A restricted version of this problem where the length of L is less than half of the perimeter of P was solved by Gewali et al. [72]. The time complexity of their algorithm is $O(n)$. Here we present a linear time algorithm for the general version of this problem. An interesting open problem is to determine the location of the guard anywhere outside the polygon P such that the guard can cover the boundary of the polygon and can optimize its territory under L-visibility.

Chapter 5

Approximate Shortest Path in Weighted Polyhedra

Locating shortest path on weighted polyhedral surface in 3D are sometimes essential in facility location in the context of geographical information system and robotics. A polyhedral surface can be represented by a set of faces, edges and vertices, where the vertices are the end points of the edges, each edge is the intersection of two faces, and each face is a plane in 3D. Without loss of generality, we assume that a polyhedron consists of n triangular faces. The shortest path problem in weighted polyhedral surfaces is in general NP-hard, and several approximation algorithms are already available. The first work on approximating the minimum cost path between two points on the surface of an weighted polyhedron appeared in [122]. It exploits the fact that the minimum cost path follows *Snell's law of refraction*, and uses *continuous Dijkstra* method [121] to compute the minimum cost path.

Recently some interesting algorithms for this problem are developed, which are based on the method of introducing Steiner points on the edges of the triangulated polyhedron. Lanthier et al. [109] proposed an approximation algorithm for the minimum cost path problem, which adds equally spaced Steiner points on the edges of the polyhedron. It

approximates the cost of the optimum path to $opt' = opt + LW$, where opt is the cost of the optimum path, W is the maximum weight among the faces of the triangulated polyhedron P, and L is the longest edge of the polyhedron. The running time of the algorithm is $O(n^5)$. In the same paper, another algorithm was presented using graph spanners; it runs in $O(n^3 \log n)$ time to report an approximation of the optimum path whose cost is no more than $\beta(opt + LW)$, where $\beta > 1$. Lanthier in his Ph.D. thesis [107] proposed an algorithm that runs in O(n log n) time and produces a solution with worst case length $\frac{2}{\sin(\theta)} \times opt$. The latest algorithm for this problem [13] produces $(1 + \epsilon) \times opt$ solution and it runs in $O(C(P)\frac{n}{\sqrt{\epsilon}} \log \frac{n}{\epsilon} \log \frac{1}{\epsilon})$. Here $C(P)$ captures the geometric parameters and weights of the faces of the given weighted polyhedron. As shown in Lemma 2.3 of [13], $C(P)$ is bounded by $4.83\Gamma \log_2(2L)$ where Γ is the average of the reciprocal of the sinuses of angles in the polyhedron P and L is the maximum of the ratios $\frac{|\ell(v)|}{r(v)}$ among all vertices $v \in P$. Here $|\ell(v)|$ is the length of the bisector of the angle incident at v in a triangular face and $r(v) = \frac{w_{min}(v)}{7w_{max}(v)}d(v)$, where $w_{max}(v)$ and $w_{min}(v)$ are the maximum and the minimum weights of the faces incident to v, respectively. The distance $d(x)$ is the minimum Euclidean distance from a point $x \in P$ to the boundary of the union of the faces containing x.

Here we propose an alternative scheme of approximating the minimum weight path of the polyhedron P. Our algorithm terminates in finite time and the approximation bound is $(1 + \frac{1}{\sin\theta}) \times opt + \epsilon nW$. Although it has some additive factor, the factor can be reduced by decreasing the value of ϵ, which in turn increases the time complexity. Detailed analysis shows that, the time complexity of our algorithm is $O(n(\log^2 \frac{L}{\epsilon}) + n(\log \frac{L}{\epsilon}) \log n)$. Note that, the factor $(1 + \frac{1}{\sin\theta})$ is always strictly less than $\frac{2}{\sin\theta}$, since θ (the minimum angle incident to the vertices of all triangular faces of P) is less than or equal to $\frac{\pi}{3}$. Furthermore, the time complexity of our algorithm does not depend on the geometric parameter θ. Finally we became able to show that for a restricted polyhedra where each triangular face is non-obtuse and the perpendicular distance of each side from its opposite vertex in each triangular face is less than the length of that side, then our algorithm achieves a solution path having length at most $2 \times opt + \epsilon nW$. Our

algorithm is very easy to implement. Thus it can be considered as one of the possible alternatives for solving the weighted shortest path problem with reasonable time and with reasonable approximation factor.

5.1 Our Proposed Algorithm

We propose an alternative scheme for placing Steiner points on the edges of the polyhedron which guarantees the termination of the algorithm, but the approximation factor depends on the fatness of the triangular faces of the polyhedron. Throughout this chapter, we will use (i) (u, v) to denote an edge joining the vertices u and v of the polyhedron P, (ii) $[\alpha, \beta]$ to denote a line segment joining points α and β, and (iii) $d(\alpha, \beta)$ to denote the Euclidean distance between the pair of points α and β.

We consider each edge of P separately and put Steiner points as follows: let (u, v) be an edge and μ be its middle point. We put a Steiner point at μ, and assume $p_0 = q_0 = \mu$. Next, we put two sets of Steiner points $p_1, p_2, \ldots p_k$ and $q_1, q_2, \ldots, q_k$ on the segments $[u, \mu]$ and $[\mu, v]$ respectively, such that p_i (resp. q_i) is the middle point of $[u, p_{i-1}]$ (resp. $[v, q_{i-1}]$), for $i = 1, 2, \ldots, k$, where k is such that $d(u, p_k) = d(v, q_k) \leq \epsilon$, for an user-defined constant ϵ (see Figure 5.1). We denote the points u and v by p_{k+1} and q_{k+1} respectively. Given the value of ϵ, we will choose k such that around each vertex v of the polyhedron, the portion of each edge (adjacent to v) inside the ϵ-ball contain exactly one Steiner point excepting the vertex v itself. Next, we form a graph G in the same manner as was done in [10]. The vertices of this graph are all the vertices of P and are the Steiner points generated on all the edges in P. The edges of the graph are drawn as follows:

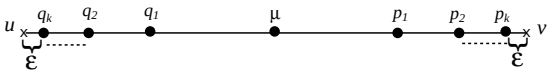

Figure 5.1: Our scheme of placing Steiner points

Consider each face f_i, $1 \leq i \leq n$ of P, connect a pair of vertices by an edge if and only if the points corresponding to v_a and v_b (i) appear on two different edges of f_i or (ii) are adjacent on an edge of f_i. The weight of an edge (v_a, v_b) is equal to $w(f_i) \times d(v_a, v_b)$ where $w(f_i)$ is the weight attached to the face f_i.

Lemma 5.1 *The number of vertices and edges of the graph G are $O(N \log(\frac{L}{\epsilon}))$ and $O(N(\log(\frac{L}{\epsilon}))^2)$ respectively in the worst case, where N denotes the number of faces of the polyhedron and L is the length of its longest edge.*

Proof : Follows from the fact that (i) the number of Steiner points on an edge of length λ is equal to $2\log(\lceil\frac{\lambda}{\epsilon}\rceil) - 1$, and (ii) a vertex on a particular edge is connected with the vertices on at most 4 edges. □

We apply Dijkstra's algorithm for computing the minimum cost path in the weighted graph G. The running time of our algorithm is $O(E + V \log V)$, where E denotes the number of edges and V denotes the number of vertices in the graph G.

5.1.1 Analysis of approximation factor

In this section, we estimate the approximation factor of the path produced by our algorithm with respect to the optimum solution. Let θ be the minimum angle among all angles incident to the vertices of all the triangular faces in P. Suppose Δuvw is a triangular face of the polyhedron P and let $a_1, a_2, \ldots, a_k$ be the set of Steiner points on the edge (v, u) of Δuvw when observed from v towards u. Now we have the following results.

Lemma 5.2 *If a_i and a_{i+1} are two consecutive Steiner points on an edge (v, u) of the polyhedron then $d(a_i, a_{i+1}) \leq d(v, a_i)$.*

Proof : Here we need to consider the following three cases:

$d(v, a_i) < \frac{d(u,v)}{2}$ and $d(v, a_{i+1}) \leq \frac{d(u,v)}{2}$: Here $d(a_i, a_{i+1}) = d(v, a_i)$.

$d(v, a_i) = \frac{d(u,v)}{2}$ and $d(v, a_{i+1}) > \frac{d(u,v)}{2}$: Here we have $d(a_i, a_{i+1}) = \frac{d(u,v)}{4}$.

$d(v, a_i) > \frac{d(u,v)}{2}$ and $d(v, a_{i+1}) > \frac{d(u,v)}{2}$: Here $d(a_i, a_{i+1}) < \frac{d(u,v)}{4}$. □

Lemma 5.3 *$d(a_i, a_{i+1}) \leq \frac{1}{\sin(\theta)} \times pdist(a_i, vw)$. where $pdist(a_i, vw)$ denotes the perpendicular distance of point a_i from a line containing the line segment $[v, w]$.*

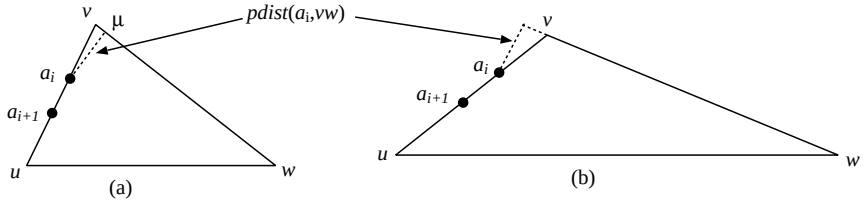

Figure 5.2: Proof of Lemma 5.3

Proof : Here we need to consider the following two cases:

$\angle uvw \leq \frac{\pi}{2}$**:** We draw a line perpendicular from a_i on the line segment $[v, w]$ (see Figure 5.2(a)). Let it meets $[u, v]$ at a point μ. Now, $\frac{d(v,a_i)}{d(a_i,\mu)} = \frac{1}{sin(\angle uvw)}$. Using Lemma 5.2, we have $\frac{d(a_i,a_{i+1})}{d(a_i,\mu)} \leq \frac{1}{sin(\angle uvw)}$. As $\angle uvw \geq \theta$, the result follows.

$\angle uvw > \frac{\pi}{2}$**:** By Lemma 5.2, $d(a_i, a_{i+1}) \leq d(v, a_i)$. Again, $\frac{pdist(a_i,vw)}{d(v,a_i)} = sin(\pi - \angle uvw)$ (see Figure 5.2(b)). As $\pi - \angle uvw = \angle vuw + \angle vwu \geq \theta$, and $sin(.)$ is an increasing function in $[0, \frac{\pi}{2}]$, we have the result in this case.

□

In order to compute the worst case approximation factor for the cost of the path from s to t along the surface of P, consider a path $\Pi_1(s, t)$ corresponding to a graph-theoretic path from s to t in G which strictly passes through the same sequence of faces and edges of the polyhedron P as in the optimal path $\Pi(s, t)$. Note that, this may not be the output $\Pi_2(s, t)$ of Dijkstra's algorithm on finding the minimum cost graph-theoretic path from s to t in G. We will compute $\Pi_2(s, t)$ as an approximation to the optimal path $\Pi(s, t)$. But in order to prove the approximation factor, we will use $\Pi_1(s, t)$. For

each line-segment σ on the $\Pi(s,t)$, any one of the three cases may arise: (A) it may pass through a portion of an edge, (B) it completely lies inside the ϵ-ball attached to a vertex of the corresponding face, and (C) it crosses a face but does not satisfy case (B). Let us consider that $\Pi(s,t)$ is approximated by $\Pi_1(s,t)$. In other words, each segment $\sigma \in \Pi(s,t)$ is approximated by a path segment $\sigma' \in \Pi_1(s,t)$ passing through the same face of σ. Here σ' may be an edge in G or concatenation of more than one edges in G depending on the aforesaid three cases. In case both the end points of σ map to the same Steiner point, σ' is considered as zero-length segment. The approximation ratio $\frac{|\sigma'|}{|\sigma|}$ is calculated for all $\sigma \in \Pi(s,t)$. An upper bound of the overall approximation factor is the maximum of the approximation ratios corresponding to all the segments on the optimal path $\Pi(s,t)$. It needs to be mentioned that, if Dijkstra's shortest path algorithm on graph G outputs a path $\Pi_2(s,t)$ which is different from $\Pi_1(s,t)$, then its cost must be less than or equal to that of $\Pi_1(s,t)$. Now, we explain the nature of σ' in three different cases.

Case A: when a segment of the optimal path passes along a side

Let the optimal path $\Pi(s,t)$ enters in a face $f = \Delta uvw$ through a point $p \in (u,w)$ and exits from f through a point $q \in (v,w)$. But the weights of the other face f' adjacent to the edge (u,v) is small enough such that the optimal path has to pass through (u,v) along a line segment $[\alpha,\beta]$. If a_1 and a_3 be the Steiner points on (u,v) which are closest to α and β respectively, then we approximate $[\alpha,\beta]$ by $[a_1,a_3]$. If a_1, a_2, a_3 are three consecutive Steiner points on (u,v), the length of $[a_1,a_3]$ may be at most $2 \times d(\alpha,\beta)$ (as shown in Figure 5.3(b)). If the number of Steiner points between a_1 and a_3 is more than one, then the approximation factor will surely be less than 2.

Case B: when a segment of the optimal path passes through the ϵ-ball attached at a vertex of the polyhedron

Here we approximate the length of the optimal path segment σ by a pair of very small line segments inside the ϵ-ball which are along the sides of the polyhedron and incident at the said vertex. Thus, the total weight will be less than $2\epsilon \times w(f)$.

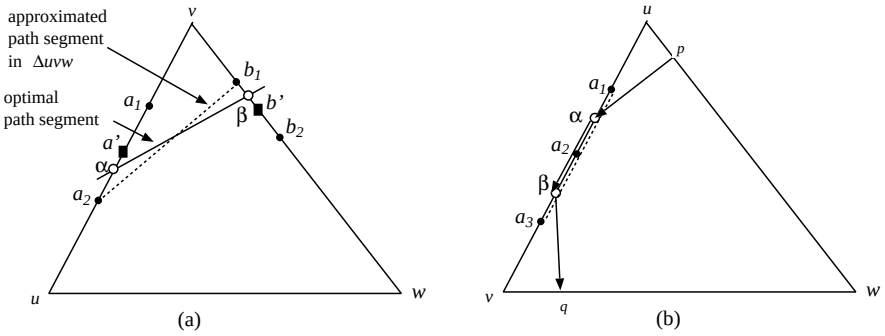

Figure 5.3: Calculation of approximation factor inside a face

Case C: when a segment of the optimal path crosses a face, but does not satisfy case B

Let a line segment σ of the path $\Pi(s,t)$ crosses a face $f = \Delta uvw$, and it intersects (u,v) and (v,w) at points α and β respectively (see Figure 5.3(a)). Let a_1 and a_2 be two Steiner points on the edge (u,v) which appear on two sides of α. Similarly, b_1 and b_2 be two Steiner points on (v,w) which appear on two sides of β. Let a' and b' be the mid-points of $[a_1,a_2]$ and $[b_1,b_2]$ respectively. Now, if α is closer to a_i, $i = 1$ or 2, and β is closer to b_j, $j = 1$ or 2, then we approximate the path segment $[\alpha,\beta]$ by $[a_i,b_j]$. The approximation factor analysis is as follows:

$d(a_i,b_j) \leq$ length of the path segment $(a_i \rightarrow \alpha \rightarrow \beta \rightarrow b_j)$

$= d(a_i,\alpha) + d(\alpha,\beta) + d(\beta,b_j))$

Thus approximation factor

$\frac{d(a_i,b_j)}{d(\alpha,\beta)} \leq \frac{d(a_i,\alpha)+d(\alpha,\beta)+d(\beta,b_j)}{d(\alpha,\beta)} \leq 1 + \frac{d(a_i,\alpha)}{d(\alpha,\beta)} + \frac{d(\beta,b_j)}{d(\alpha,\beta)} \leq 1 + \frac{\frac{d(a_i,a_{i+1})}{2}}{d(\alpha,\beta)} + \frac{\frac{d(b_j,b_{j+1})}{2}}{d(\alpha,\beta)} \leq 1 + \frac{\frac{d(a_i,a_{i+1})}{2}}{pdist(\alpha,vw)} + \frac{\frac{d(b_j,b_{j+1})}{2}}{pdist(\beta,uv)} \leq (1 + \frac{1}{\sin(\theta)})$ (from Lemma 5.3).

The analysis of the cases A, B, and C lead to the following theorem:

Theorem 5.1 *The length of the path produced by our algorithm is at most*

$$(1 + \tfrac{1}{\sin(\theta)})\Pi(s,t) + 2\epsilon \times (w(f_{\alpha_1}) + w(f_{\alpha_2}) + \ldots + w(f_{\alpha_m})),$$

where $f_{\alpha_1}, f_{\alpha_2}, \ldots, f_{\alpha_m}$ are the faces such that an ϵ-ball of each of these faces contains a complete segment of the optimal path $\Pi(s,t)$.

Remark 5.1 *Theorem 5.1 says that, the approximation factor depends upon the geometric parameter θ, which indicates the smallest angle among the triangles of the polyhedron. But by Lemma 5.1, the execution time and space requirement of our algorithm does not depend on this geometric parameter.*

5.1.2 A more restricted model with better approximation bound

We may get better approximation bound if the faces of the polyhedron can be triangulated satisfying the following *nice* property: *each triangular face Δ is non-obtuse and the perpendicular distance of each side of Δ from its opposite vertex is less than the length of that side.*

Lemma 5.4 *A triangle satisfying the aforesaid* nice *property, have each angle $\geq \frac{\pi}{4}$.*

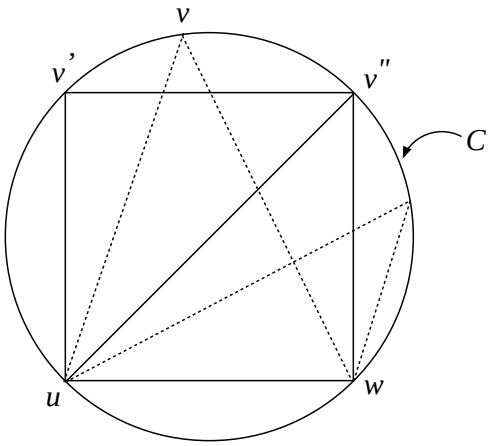

Figure 5.4: Proof of Lemma 5.4

Proof : Consider a non-obtuse triangle Δuvw, whose base is $[u, w]$. Consider a corridor in the plane containing Δuvw by drawing perpendiculars on $[u, w]$ at the points u and w respectively. If the point v is inside the corridor, the angles $\angle vuw$ and $\angle vwu$ are both non-obtuse. Now, if the perpendicular distance of v on $[u, w]$ is less than the length of $[u, w]$, the point v must be inside the square $\Box v'v''uw$ (see Figure 5.4). Note that if v is

aligned with v' or v'', then $\angle uvw = \frac{\pi}{4}$. Thus, we can consider a circle C passing through the four points v', v'', u and w. Note that, $v \in \Box v'v''uw$ implies $v \in C$, which in turn, implies $\angle uvw \geq \frac{\pi}{4}$. □

Lemma 5.4 has the following immediate consequence: If the faces of the polyhedron satisfy the *nice* property then a trivial upper bound on the weighted length of the path produced by our algorithm is $(1+\sqrt{2})\Pi(s,t) + 2n\epsilon \times W$ (see Theorem 5.1), where W is the maximum weight among all the n faces in P.

Below we show that the approximation bound of our algorithm is much better than this trivial bound.

Lemma 5.5 *Let $[\alpha, \beta]$ be a segment of the optimal path $\Pi(s,t)$, which passes through the interior of a face Δuvw satisfying* nice *property with $\alpha \in (u,v)$ and $\beta \in (v,w)$. If a and b are the two Steiner points appearing on (u,v) and (v,w) which are closest to α and β respectively, then, $\frac{d(a,b)}{d(\alpha,\beta)} < 1.5$.*

Proof : Let us align the side (u,w) of Δuvw with the X-axis. Let the coordinates of u, v and w be $u = (0,0)$, $v = (h,k)$, and $w = (h+\delta, 0)$. Here $h + \delta \geq k$ since Δuvw satisfies the *nice* property. Let μ and γ be the mid points of the edges (u,v) and (v,w) respectively.

Let $[\alpha, \beta]$ be the path segment inside $f = \Delta uvw$ and is approximated by $[a,b]$, where a and b are two Steiner points on (u,v) and (v,w) respectively. In Figure 5.5(a), the line segment $[a,b]$ is shown using dotted line, and $[\alpha,\beta]$ is shown using solid line.

We prove the lemma by showing $D = 9 \times (d(\alpha,\beta))^2 - 4 \times (d(a,b))^2 \geq 0$ considering the following four exhaustive cases :

Case 1: a is below μ on (u,v) and b is above ν on (v,w)
This situation is demonstrated in Figure 5.5(a). Let us assume that the coordinates of a and b be $a = (\frac{h}{2^{i+1}}, \frac{k}{2^{i+1}})$ and $b = (h + \frac{\delta}{2^{j+1}}, \frac{2^{j+1}-1}{2^{j+1}}k)$, $i, j \geq 1$. The two neighboring Steiner points of a on (u,v) are $a_1 = (\frac{h}{2^i}, \frac{k}{2^i})$ and $a_2 = (\frac{h}{2^{i+2}}, \frac{k}{2^{i+2}})$, and two neighboring

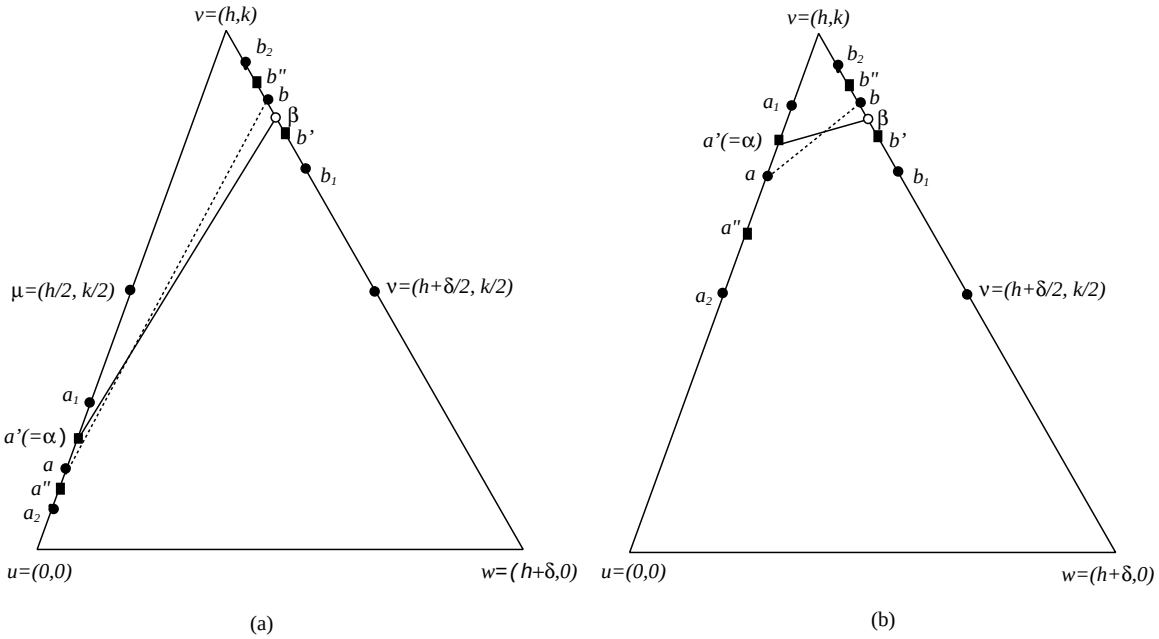

Figure 5.5: Proof of Lemma 5.5

Steiner points of b on (v, w) are $b_1 = (h + \frac{\delta}{2^j}, \frac{2^j-1}{2^j}k)$ and $b_2 = (h + \frac{\delta}{2^{j+2}}, \frac{2^{j+2}-1}{2^{j+2}}k)$. The mid-points of $[a_1, a]$ and $[a, a_2]$ are $a' = (\frac{3h}{2^{i+2}}, \frac{3k}{2^{i+2}})$ and $a'' = (\frac{3h}{2^{i+3}}, \frac{3k}{2^{i+3}})$ respectively, and mid-points of $[b_1, b]$ and $[b, b_2]$ are $b' = (h + \frac{3\delta}{2^{j+2}}, \frac{2^{j+2}-3}{2^{j+2}}k)$ and $b'' = (h + \frac{3\delta}{2^{j+3}}, \frac{2^{j+3}-3}{2^{j+3}}k)$ respectively. As $[a, b]$ is the approximation of the optimal path segment $[\alpha, \beta]$ inside Δuvw, α must lie in the interval $[a', a'']$ and β must lie in the interval $[b', b'']$.

If β is fixed at any point on $[b', b'']$, the minimum length of $[\alpha, \beta]$ is attained when $\alpha = a'$. Now, in order to prove the lemma, we need to consider the following two subcases:

Case 1.1: β lies on the line segment $[b, b']$

Here, assume that the minimum value of $d(\alpha, \beta)$ is achieved when $d(b', \beta) : d(\beta, b) = r : 1$ for some $r \geq 0$. Thus, the coordinate of β is $\beta = ((h + \frac{\delta}{2^{j+2}} + \frac{r+2}{r+1} \times \frac{\delta}{2^{j+2}}), (k - \frac{k}{2^{j+2}} - \frac{r+2}{r+1} \times \frac{k}{2^{j+2}}))$, and

$d(\alpha, \beta) = d(a', \beta)$

$= \sqrt{(h - \frac{3h}{2^{i+2}} + \frac{\delta}{2^{j+2}} + \frac{r+2}{r+1} \times \frac{\delta}{2^{j+2}})^2 + (k - \frac{3k}{2^{i+2}} - \frac{k}{2^{j+2}} - \frac{r+2}{r+1} \times \frac{k}{2^{j+2}})^2}$.

The length of the approximated path
$d(a,b) = \sqrt{(h - \frac{h}{2^{i+1}} + \frac{\delta}{2^{j+1}})^2 + (k - \frac{k}{2^{i+1}} - \frac{k}{2^{j+1}})^2}$.

Substituting the values of $d(\alpha,\beta)$ and $d(a,b)$ on the expression of D, we have
$D = 9 \times (d(\alpha,\beta))^2 - 4 \times (d(a,b))^2$,
$= (5h - \frac{13h}{2^{i+2}} + \frac{10\delta}{2^{j+2}} + \frac{3\delta}{(r+1)2^{j+2}}) \times (h - \frac{5h}{2^{i+2}} + \frac{2\delta}{2^{j+2}} + \frac{3\delta}{(r+1)2^{j+2}})$
$+ (5k - \frac{13k}{2^{i+2}} - \frac{10k}{2^{j+2}} - \frac{3k}{(r+1)2^{j+2}}) \times (k - \frac{5k}{2^{i+2}} - \frac{2k}{2^{j+2}} - \frac{3k}{(r+1)2^{j+2}})$.

If $i \geq 1$ and $j > 1$, $D \geq 0$ for all $r \geq 0$.

If $i = j = 1$, $D = \frac{1}{64}((27\times h+10\times\delta+\frac{3\delta}{r+1})(3\times h+2\times\delta+\frac{3\delta}{r+1})+(k)^2(17-\frac{54}{r+1}+\frac{9}{(r+1)^2}))$.
Here also, $D > 0$ for all $r \geq 0$, since the value of k can be at most $h + \delta$.

Case 1.2: β lies on the line segment $[b'', b]$

Here, assume that the minimum value of $d(\alpha,\beta)$ is achieved when $d(b'',\beta) : d(\beta,b) = r : 1$ for some $r \geq 0$. Thus,
$\beta = ((h + \frac{3\delta}{2^{j+3}} + \frac{r}{r+1} \times \frac{\delta}{2^{j+3}}), (k - \frac{4r+3}{r+1} \times \frac{k}{2^{j+3}}))$, and
and $d(\alpha,\beta) = \sqrt{(h - \frac{3h}{2^{i+2}} + \frac{3\delta}{2^{j+3}} + \frac{r}{r+1} \times \frac{\delta}{2^{j+3}})^2 + (k - \frac{3k}{2^{i+2}} - \frac{4r+3}{r+1} \times \frac{k}{2^{j+3}})^2}$.
Note that, in this case the optimal path segment is not shown in Figure 5.5(a).

Here, $D = 9 \times (d(\alpha,\beta))^2 - 4 \times (d(a,b))^2$,
$= (5 \times h - \frac{13\times h}{2^{i+2}} + \frac{17\times\delta}{2^{j+3}} + \frac{r}{r+1} \times \frac{3\delta}{2^{j+3}}) \times (h - \frac{5\times h}{2^{i+2}} + \frac{\delta}{2^{j+3}} + \frac{3r}{r+1} \times \frac{\delta}{2^{j+3}}) + (5 \times k - \frac{13\times k}{2^{i+2}} - \frac{17\times k}{2^{j+3}} - \frac{3r}{r+1} \times \frac{k}{2^{j+3}}) \times (k - \frac{5\times k}{2^{i+2}} - \frac{k}{2^{j+3}} - \frac{3r}{r+1} \times \frac{k}{2^{j+3}})$.

It can be shown that $D > 0$ for all $i, j \geq 1$ and $r \geq 0$.

Case 2: both a and b are above the line containing $[\mu,\nu]$, and $[a,b]$ is not parallel to the edge (u,w)

This case is demonstrated in Figure 5.5(b). Without loss of generality, assume that $a = (\frac{2^i-1}{2^i}h, \frac{2^i-1}{2^i}k)$ and $b = (h + \frac{\delta}{2^{j+1}}, \frac{2^{j+1}-1}{2^{j+1}}k)$, $1 \leq i \leq j$.

Using the same notation as in Case 1,
$a_1 = (\frac{2^{i+1}-1}{2^{i+1}}h, \frac{2^{i+1}-1}{2^{i+1}}k)$ and $a_2 = (\frac{2^{i-1}-1}{2^{i-1}}h, \frac{2^{i-1}-1}{2^{i-1}}k)$ are the neighboring Steiner points of a, and on (u,v),

$a' = (\frac{2^{i+2}-3}{2^{i+2}}h, \frac{2^{i+2}-3}{2^{i+2}}k)$ and $a'' = (\frac{2^{i+1}-3}{2^{i+1}}h, \frac{2^{i+1}-3}{2^{i+1}}k)$ are the mid-points of $[a, a_1]$ and $[a, a_2]$,

$b_1 = (h + \frac{\delta}{2^j}, \frac{2^j-1}{2^j}k)$ and $b_2 = (h + \frac{\delta}{2^{j+2}}, \frac{2^{j+2}-1}{2^{j+2}}k)$ are the adjacent Steiner points of b on (v, w),

$b' = (h + \frac{3\delta}{2^{j+2}}, \frac{2^{j+2}-3}{2^{j+2}}k)$ and $b'' = (h + \frac{3\delta}{2^{j+3}}, \frac{2^{j+3}-3}{2^{j+3}}k)$ are the mid-points of $[b, b_1]$ and $[b, b_2]$.

As the optimal path segment $[\alpha, \beta]$ is approximated by $[a, b]$, $\alpha \in [a', a'']$ and $\beta \in [b', b'']$.

Using similar argument as in Case 1, we can say that, for some fixed $\beta \in [b', b'']$, $d(\alpha, \beta)$ achieves minimum if α coincides with a'. As in Case 1, here also we need to consider the following two subcases:

Case 2.1: $\beta \in [b, b']$

Let the length of $[\alpha, \beta]$ becomes minimum when

$d(b, \beta) : d(\beta, b') = 1 : r$ for some $r \geq 0$.

Thus, $\beta = ((h + \frac{\delta}{2^{j+2}} + \frac{r+2}{r+1} \times \frac{\delta}{2^{j+2}}), (k - \frac{k}{2^{j+2}} - \frac{r+2}{r+1} \times \frac{k}{2^{j+2}}))$.

Now, $D = 9 \times (d(\alpha, \beta))^2 - 4 \times (d(a, b))^2$,

$= \frac{k^2}{2^{2i+4}}(17 - \frac{44}{2^{j-i}} - \frac{54}{(r+1)2^{j-i}} + \frac{36}{(r+1)2^{2(j-i)}} + \frac{20}{2^{2(j-i)}} + \frac{9}{(r+1)^2 2^{2(j-i)}}) + \frac{1}{2^{2i+4}}(17h^2 + \frac{44h\delta}{2^{j-i}} + \frac{54h\delta}{(r+1)2^{j-i}} + \frac{36\delta^2}{(r+1)2^{2(j-i)}} + \frac{20\delta^2}{2^{2(j-i)}} + \frac{9\delta^2}{(r+1)^2 2^{2(j-i)}})$.

It can be easily observed that this expression is positive when $j - i > 2$, and with a simple algebraic manipulation it can be shown that D is positive for $j - i = 0, 1, 2$ also.

Case 2.2: $\beta \in [b, b'']$

Let the length of $[\alpha, \beta]$ becomes minimum when

$d(b_2, \beta) : d(\beta, b) = r : 1$ for some $r \geq 0$.

Thus,

$\beta = ((h + \frac{3\delta}{2^{j+3}} + \frac{r}{r+1} \times \frac{\delta}{2^{j+3}}), (k - \frac{4r+3}{r+1} \times \frac{k}{2^{j+3}}))$, and

$d(\alpha, \beta) = \sqrt{(h - \frac{3h}{2^{i+2}} + \frac{3\delta}{2^{j+3}} + \frac{r}{r+1} \times \frac{\delta}{2^{j+3}})^2 + (k - \frac{3k}{2^{i+2}} - \frac{4r+3}{r+1} \times \frac{k}{2^{j+3}})^2}$.

It needs to be mentioned that in this case the optimal path segment is not shown in Figure 5.5(b).

Now, $D = 9 \times (d(\alpha,\beta))^2 - 4 \times (d(a,b))^2$
$= (\frac{17\times h}{2^{i+2}} + \frac{17\times\delta}{2^{j+3}} + \frac{3\times r\delta}{(r+1)2^{j+3}}) \times (\frac{h}{2^{i+2}} + \frac{\delta}{2^{j+3}} + \frac{3\times r\delta}{(r+1)2^{j+3}}) + (\frac{17\times k}{2^{i+2}} - \frac{17\times k}{2^{j+3}} - \frac{3rk}{(r+1)2^{j+3}}) \times$
$(\frac{k}{2^{i+2}} - \frac{k}{2^{j+3}} - \frac{3\times rk}{(r+1)2^{j+3}})$

It is easy to show that, $D > 0$ for all $j \geq i$ and for all $r \geq 0$.

Case 3: both the Steiner points a and b are below the line containing $[\mu,\nu]$
In this case, D can be shown to be positive in a manner similar to Case 2.

Case 4: ab is parallel to (u,w)
This situation is demonstrated in Figure 5.6(a). To analyze this situation, we use the same notations as in the earlier cases. Here, $[a',b']$ is also parallel to the edge (u,w). Let us draw two line segments parallel to (u,w) from α and β which intersects (u,v) and (v,w) at β^* and α^* respectively. Now, $d(\alpha,\beta) \geq min(d(\beta,\beta^*), d(\alpha,\alpha^*)) \geq d(a',b')$.

Thus, $\frac{d(a,b)}{d(\alpha,\beta)} \geq \frac{d(a,b)}{d(a',b')} = \frac{d(u,a)}{d(u,a')} = C$ (say).

When a is below μ (the mid-point of (u,v)) then the approximation ratio can can directly be shown to be $C = \frac{2^{i+2}-2}{2^{i+2}-3}$, assuming $a = (\frac{h}{2^{i+1}}, \frac{k}{2^{i+1}})$ and $b = (h + \frac{2^{i+1}-1}{2^{i+1}} \times \delta, \frac{k}{2^{i+1}})$,
If a is above μ, then it can be shown that C is at most $\frac{4}{3}$. □

Remark 5.2 *It needs to be mentioned that, the approximation factor $\frac{2\sqrt{5}}{3} \approx 1.4907$ is achievable for an instance satisfying Case 2.1 with $i = j = 1$, $\delta = 0$ and $h = k$ (see Figure 5.6(b)).*

Theorem 5.2 *The length of the path produced by our algorithm is at most*

$$2 \times \Pi(s,t) + 2\epsilon \times (w(f_{\alpha_1}) + w(f_{\alpha_2}) + \ldots + w(f_{\alpha_m})),$$

where $f_{\alpha_1}, f_{\alpha_2}, \ldots, f_{\alpha_m}$ are the faces such that an ϵ-ball of each of these faces contains a complete segment of the optimal path $\Pi(s,t)$ and is computable in $O(n(\log^2 \frac{L}{\epsilon}) + n(\log \frac{L}{\epsilon}) \log n)$ time, where n denotes the number of triangulated faces of the polyhedron and L is the length of its longest edge.

Proof : If each segment of the optimal path does not coincide with an edge of the polyhedron, and is not completely contained in the ϵ-ball of a vertex, then by Lemma 5.5 the approximation factor is bounded above by 1.5. But if there exists instance(s)

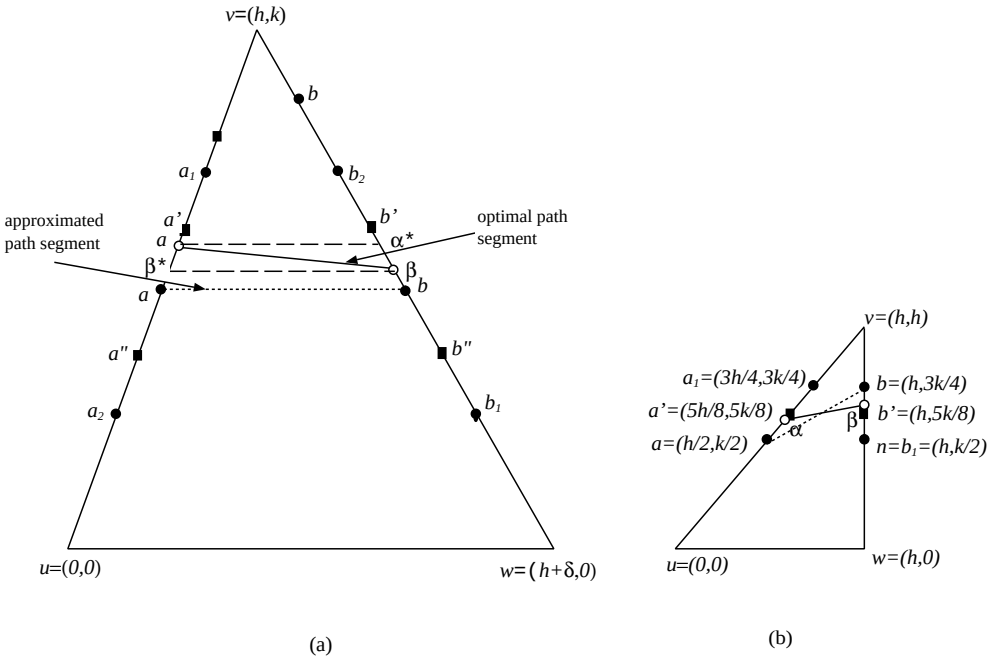

Figure 5.6: (a) Illustration of Case 4, and (b) An example achieving 1.4907 approximation factor

where the optimum path coincides with edge(s) of the polyhedron then by the analysis of Case A in Section 5.1.1, the approximation factor can be at most 2. The additive term appears if there exists instances of Case B as described in Section 5.1.1. □

5.2 Conclusion

An efficient and implementable algorithm for computing the shortest path between a pair of points on the surface of a weighted polyhedron is proposed. The approximation bound of the result produced by our algorithm is $(1 + \frac{1}{sin\theta})opt + \epsilon nW$, where θ is the smallest angle among the triangular faces of the polyhedron and W denotes the maximum weight among the faces of the polyhedron P. In a restricted case, where each triangular face is non-obtuse and the perpendicular distance of each side from its opposite vertex in each of the triangular faces is less than the length of that side, our algorithm achieves a solution whose length is at most $2 \times opt + \epsilon nW$, where opt is the length of the shortest path.

Chapter 6

Monotone Descent Path Problem on a Polyhedral Terrain

In this chapter, we will consider a constrained variation of the shortest path problem, on the surface of a polyhedral terrain. As in the earlier chapter, we will assume that the faces of the terrain are triangulated. We will use n to denote the number of vertices of the terrain. Here the desired shortest path must be monotone descent from the source point s to the destination point t, and it must pass through the surface of the terrain. Thus, for every pair of points $p = (x(p), y(p), z(p))$ and $q = (x(q), y(q), z(q))$ on the path, if $dist(s, p) < dist(s, q)$ then $z(p) \geq z(q)$, where $dist(s, p)$ denotes the distance of p from s along the aforesaid path. This problem is posed as an open problem by Berg and Kreveld [26], in the sense that no bound on the combinatorial or Euclidean length of the shortest monotone descent path between a pair of points on the surface of a polyhedral terrain is available in the literature. Some interesting observations of the problem lead us to design efficient polynomial time algorithm for solving the shortest monotone descent path problem in the following special cases:

1. there exists at least one monotone descent path from s to t through a sequence of faces such that each pair of consecutive faces are in convex position (see Section 6.3), and

2. given a sequence of pairwise adjacent faces having their boundaries parallel to each other (but the faces are not all necessarily in convex (resp. concave) position). The objective is to find the shortest monotone descent path from s to t through that sequence of faces (see Section 6.4).

In Case 1, if the terrain contains n triangulated faces, the preprocessing of those faces need $O(n^2 \log n)$ time and $O(n^2)$ space, and the shortest path query can be answered in $O(k + \log n)$ time, where k is the number of faces through which the optimum path passes. In Case 2, if a sequence of n faces with their boundaries in parallel position is given, the shortest monotone descent path from s to t through that edge sequence can be computed in $O(n \log n)$ time. The solution technique for this case indicates the difficulties of handling the general terrain.

The problem is motivated from the agricultural applications where the objective is to lay a canal of minimum length from the source of water at the top of the mountain to the ground for irrigation purpose. Another application of Case 2 can be observed in the design of fluid circulation systems in automobiles or refrigerator/air-condition machines.

6.1 Preliminaries

A terrain $\mathcal{T}$ is a polyhedral surface in $\mathbb{R}^3$ with a special property: the vertical line at any point on the xy-plane intersects the surface of $\mathcal{T}$ at most once. Thus, the projections of all the faces of a terrain on the xy-plane are mutually non-intersecting at their interior. Each vertex p of the terrain is specified by a triple $(x(p), y(p), z(p))$. More formally, a terrain $\mathcal{T}$ is the image of the real bivariate function ζ defined on a compact and connected domain Ω in the Euclidean plane, i.e, $\mathcal{T} = \{(x, y, \zeta(x, y)), (x, y) \in \Omega\}$ [135]. Without loss of generality, we assume that all the faces of the terrain are triangles, and the source point s is a vertex of the terrain.

Definition 6.1 *[121] Let f and f' be a pair of faces of $\mathcal{T}$ sharing an edge e. The* planar unfolding *of face f' onto face f is the image of the points of f' when rotated about the line e onto the plane of f such that the points in f and the points in f' lie in two different sides of the edge e respectively (i.e., the faces f' and f becomes coplanar and they do not overlap after unfolding).*

Let $\{f_0, f_1, \ldots, f_m\}$ be a sequence of adjacent faces. The edge common to f_{i-1} and f_i is e_i. We define the *planar unfolding with respect to the edge sequence* $\mathcal{E} = \{e_1, e_2, \ldots, e_{m-1}\}$ as follows: obtain the planar unfolding of face f_m onto face f_{m-1}, then get the planar unfolding of the resulting plane onto f_{m-2}, and so on; finally, get the planar unfolding of the entire resulting plane onto f_0. From now onwards, this event will be referred to as $U(\mathcal{E})$.

Definition 6.2 *[121] A geodesic path is a simple path (i.e., not self-intersecting) whose intersection with any face is the union of disjoint line segments and is locally optimal. Therefore it can not be shortened by slight perturbations.*

Definition 6.3 *A path $\pi(s,t)$ from a point s to a point t on the surface of the terrain is said to be a relaxed geodesic path if (i) it entirely lies on the surface of the terrain, (ii) it is not self-intersecting, and (iii) in each face through which it passes, the intersection of this path and the corresponding face is a straight line segment.*

The distance $dist(p,q)$ between a pair of points p and q on $\pi(s,t)$ is the length of the simple path from p to q along $\pi(s,t)$. The path $\pi_{geo}(s,t)$ is said to be the *shortest geodesic path* if the distance between s and t along $\pi_{geo}(s,t)$ is minimum among all possible geodesic paths from s to t. Note that, $\pi_{geo}(s,t)$ is the relaxed geodesic path between two points s and t having minimum length.

Lemma 6.1 *For a pair of points α and β, if $\pi_{geo}(\alpha,\beta)$ passes through the interior of each edge in an edge-sequence $\mathcal{E}$ of a polyhedron, then in the planar unfolding $U(\mathcal{E})$, the path $\pi_{geo}(\alpha,\beta)$ is a straight line segment.*

Proof : The proof follows from Lemma 3.3 in the paper by Mitchel et al. [121]. □

Definition 6.4 *A path $\pi(s,t)$ $(z(s) \geq z(t))$ on the surface of a terrain is a monotone descent path if for every pair of points $p, q \in \pi(s,t)$, $dist(s,p) < dist(s,q)$ implies $z(p) \geq z(q)$.*

We will use $\pi_{md}(p,q)$ and $\delta(p,q)$ to denote the shortest monotone descent path from p to q and its length, respectively. If $\pi_{geo}(p,q)$ corresponds to the line segment $[p^*, q^*]$ in the unfolded plane along an edge sequence and it satisfies monotone descent property, then q is said to be *straight line reachable* from p in the unfolded plane. In such a case, $\pi_{md}(p,q) = \pi_{geo}(p,q)$.

Remark 6.1 *A monotone descent path between a pair of points s and t may not exist (Figure 6.1(a)). Again, if monotone descent path from s to t exists, then $\pi_{md}(s,t)$ may not coincide with $\pi_{geo}(s,t)$ (Figure 6.1(b)).*

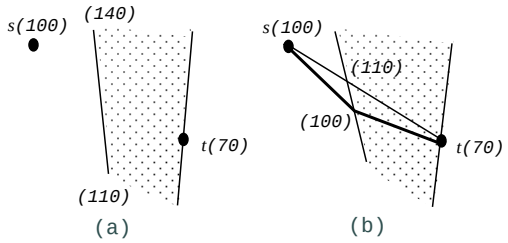

Figure 6.1: Justification of Remark 6.1

Lemma 6.2 *If the shortest monotone decent path $\pi_{md}(s,t)$ passes through a face f, then the intersection of $\pi_{md}(s,t)$ with the face f is a straight line segment.*

Proof : [By contradiction] Let the portion of $\pi_{md}(s,t)$, which lies in face f, is not a single line segment. Let us consider a pair of points $p_1, p_2 (\in f)$ on the path $\pi_{md}(s,t)$ (with $dist(s,p_1) < dist(s,p_2)$) such that their joining line segment does not coincide

with an edge of $\pi_{md}(s,t)$. Note that, the line segment $[p_1, p_2]$ satisfies the monotone descent property, and its length is less than the length of the path from p_1 to p_2 along $\pi_{md}(s,t)$. Hence we have a contradiction. □

Lemma 6.3 *Given a vertex s and a pair of points α and β on the terrain $\mathcal{T}$, $\pi_{md}(s,\alpha)$ and $\pi_{md}(s,\beta)$ can not intersect except at some vertex of $\mathcal{T}$. Moreover, if they intersect at a vertex v then the length of the subpath from s to v on both $\pi_{md}(s,\alpha)$ and $\pi_{md}(s,\beta)$ are same.*

Proof : Let $\pi_{md}(s,\alpha)$ and $\pi_{md}(s,\beta)$ intersects at a point γ, which is equidistant from s along both the paths $\pi_{md}(s,\alpha)$ and $\pi_{md}(s,\beta)$, otherwise one of these two paths can not be optimum. Thus, the second part of the lemma follows.

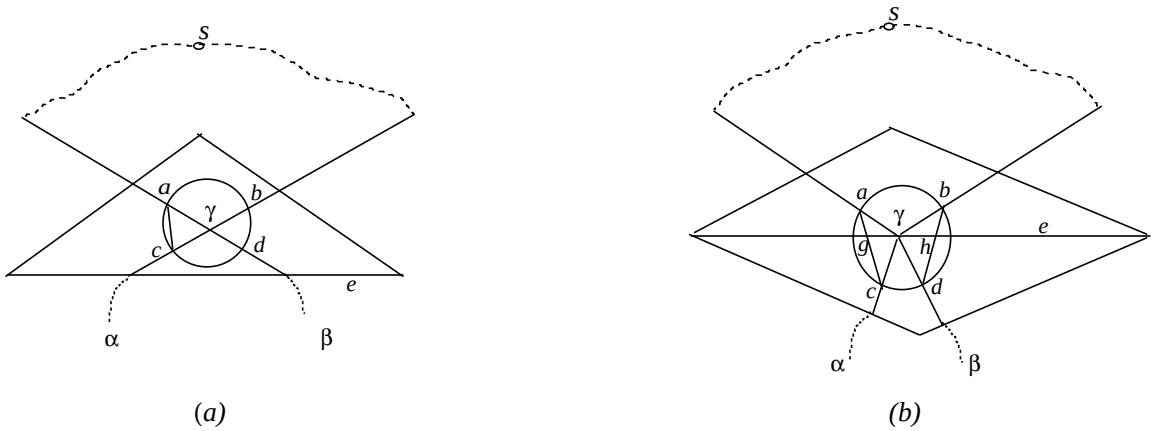

Figure 6.2: Proof of Lemma 6.3

We prove the first part by contradiction. We need to consider two cases: (i) γ is inside a face f, and (ii) γ lies on an edge e which is adjacent to a pair of faces f and f'.

In case (i) consider a very small circle centered at γ which completely lies inside the face f. The path $\pi_{md}(s,\alpha)$ intersects the circle at two points b and c, and the path $\pi_{md}(s,\beta)$ intersects the circle at a and d (see Figure 6.2(a)). From the second part of the Lemma, the length of the paths $s \sim b \rightarrow \gamma \rightarrow c \sim \alpha$ and $s \sim a \rightarrow \gamma \rightarrow c \sim \alpha$ are

same, and both are optimum paths (by the statement of the lemma). Now, consider the path $s \sim a \rightarrow c \sim \alpha$. Its length is less than both the paths mentioned above (by triangle inequality). Again, $z(a) \geq z(\gamma) \geq z(c)$ due to the fact that both $\pi_{md}(s, \alpha)$ and $\pi_{md}(s, \beta)$ are monotone descent. So, the path $s \sim a \rightarrow c \sim \alpha$ is monotone descent also. Thus, we have a contradiction.

The case (ii) can be handled by unfolding f onto f' and drawing the circle around γ in the unfolded plane. The path $\pi_{md}(s, \alpha)$ intersects the circle at two points b and c, and the path $\pi_{md}(s, \beta)$ intersects the circle at a and d. The edge e intersects the line segment $[a, c]$ at the point g and the line segment $[b, d]$ at the point h (see Figure 6.2(b)). If all the three values $z(a), z(b), z(\gamma)$ are equal then the plane is horizontal. In that case the length of the path $s \sim a \rightarrow c \sim \alpha$ is monotone descent and its length is less than the paths mentioned above. Thus, we have a contradiction. Similarly, we can argue for the case when $z(c) = z(d) = z(\gamma)$. Next, we consider the case where none of the planes are horizontal. Here we need to consider the following three cases:

$z(\gamma) \geq z(h)$**:** Here we can always get a point p on the interval $[h, \gamma)$ such that $z(b) \geq z(\gamma) \geq z(p) \geq z(d)$. Hence, the path $s \sim b \rightarrow p \rightarrow d$ is monotone decreasing, and is shorter that the path $s \sim b \rightarrow \gamma \rightarrow d$.

$z(\gamma) < z(h)$ **and** $z(b) > z(\gamma)$**:** Here we can identify a point p in the interval $[h, \gamma)$ such that $z(b) > z(p) \geq z(c)$. Hence, the path $s \sim b \rightarrow p \rightarrow d$ is monotone decreasing, and is shorter that the path $s \sim b \rightarrow \gamma \rightarrow d$.

$z(\gamma) < z(h)$ **and** $z(b) = z(\gamma)$**:** Here $z(a) < z(\gamma)$, and therefore no monotone descent path of the form $s \sim a \rightarrow \gamma$ exists.

Thus, in all the above three cases, we reached to the contradiction. □

Definition 6.5 *Given an arbitrary point p on the surface of the terrain $\mathcal{T}$, the descent flow region of p (called $DFR(p)$) is the region on the surface of $\mathcal{T}$ such that each point $q \in DFR(p)$ is reachable from p through a monotone descent path.*

Thus, given a polyhedral terrain $\mathcal{T}$ and a given point $s \in \mathcal{T}$, we study the following problems:

P1 Construct $DFR(s)$.

P2 For a given query point $t \in DFR(s)$ report $\pi_{md}(s,t)$, and its length.

Problem P2 seems to be difficult in general. We identified the following two special cases where it can be solved in polynomial time.

P2.1 Assuming that the source point s is in a face f_0, we construct a data structure such that given any query point t in face f_m, we can identify a monotone descent shortest path passing through a sequence of faces $\{f_0, f_1, \ldots, f_{m-1}, f_m\}$ that are in convex position (to be defined in Section 6.3), provided such a path exists.

P2.2 Given a sequence of faces $\{f_0, f_1, \ldots, f_m\}$ of a polyhedral terrain (not necessarily convex/concave), if e_i denotes the edge separating f_{i-1} and f_i, and the projections of the edges $e_1, e_2, \ldots, e_k$ on the XY-plane are parallel, then we can identify the monotone descent shortest path between a pair of points $s \in f_0$ and $t \in f_m$ through that sequence of faces.

6.2 Computation of $DFR(s)$

Given the source point s, if it lies on an edge e of a triangulated face, we connect s by adding edges with the vertices opposite to e in both the faces adjacent to e. If s lies inside a triangulated face then s is connected to all the three vertices of the face by adding edges. Thus s may always be considered as a vertex of the triangulated terrain.

Observation 6.1 *If r is reachable from s using a monotone descent path, then $DFR(r) \subseteq DFR(s)$.*

Observation 6.2 *Let Δspq be a triangular face adjacent to the source s with $z(p) \leq z(q)$. Now,*

(i) if $z(s) \geq z(q)$ *then* $\Delta spq \subseteq DFR(s)$,
(ii) if $z(s) < z(p)$ *then* $\Delta spq \cap DFR(s) = \phi$ *(empty region), and*
(iii) if $z(p) \leq z(s) < z(q)$ *then there exists a point* r *on the edge* (p, q) *(with* $z(r) = z(s)$*) such that* $\Delta spr \subseteq DFR(s)$*, and no point in the properly inside of* Δsrq *is in* $DFR(s)$*. In this case, if* $z(p) = z(r)$*, then* Δspr *degenerates to the line segment* $[s, p]$ *(or equivalently* $[s, r]$*).*

In Figure 6.3, faces B and F satisfy Cases (i) and (ii) of Observation 6.2 respectively; all other faces satisfy Case (iii) of Observation 6.2.

We consider all the faces adjacent to s, and compute the initial descent flow region inside those faces. The union of these regions is denoted as $IDFR(s)$. The projection of $IDFR(s)$ in the XY-plane is a connected region, which may be (i) a simple polygon with s inside it or (ii) a collection of simple polygons each having s as a vertex. The vertices of $IDFR(s)$ (excluding s itself) are said to be the descent flow neighbor of s, and is denoted as $DFN(s)$ (see Figure 6.3).

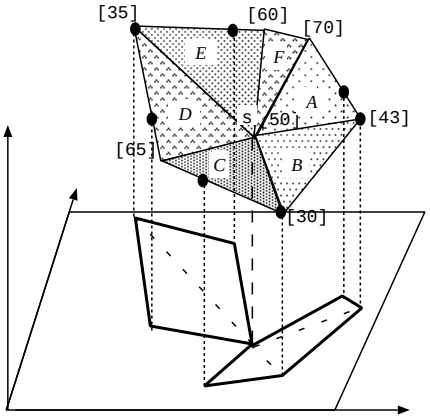

Figure 6.3: Illustration of $DFN(s)$

Lemma 6.4 $DFR(s) = (\cup_{r \in DFN(s)} DFR(r)) \bigcup IDFR(s)$.

Proof : From Observations 6.1 and 6.2,
$(\bigcup_{r \in DFN(s)} DFR(r)) \bigcup IDFR(s) \subseteq DFR(s)$. We now prove that
$DFR(s) \subseteq (\bigcup_{r \in DFN(s)} DFR(r)) \bigcup IDFR(s)$.

Let q be a point in $DFR(s)$ but not in $(\bigcup_{r \in DFN(s)} DFR(r)) \bigcup IDFR(s)$. Consider a monotone descent path from s to q. By Observation 6.2, it intersects a boundary edge $[a, b]$ of $IDFR(s)$ at a point c. Assume that $z(a) \geq z(b)$. The path from the point a to the point c along the boundary $[a, b]$ is a monotone descent path. Thus, $q \in DFR(a)$, which leads to the contradiction. □

We compute $IDFR(s)$ by considering the faces adjacent to s. The processing of the triangular faces which are not adjacent with the source s is discussed below.

Observation 6.3 *If more than one point on the boundary of a face Δabc are reachable from s, then for each pair of such points α and β, $z(\alpha) < z(\beta)$ implies $DFR(\alpha) \cap \Delta abc \subseteq DFR(\beta) \cap \Delta abc$.*

Observation 6.4 *The intersection of a face of $\mathcal{T}$ with $DFR(s)$ may be a vertex of that face, an edge of that face which is parallel to XY-plane, or a polygonal region.*

During the execution of the algorithm, we maintain a priority queue $\mathcal{Q}$ which is initialized with $DFN(s)$, and process these elements in an ordered manner (as discussed below). During the processing of a member $\alpha \in \mathcal{Q}$, the set of points $DFN(\alpha)$ are also inserted in $\mathcal{Q}$. The algorithm continues until all the members in $\mathcal{Q}$ are processed.

While processing an element $\alpha \in \mathcal{Q}$, if it is a vertex of a triangular face Δabc, then it is processed as in Lemma 6.4. If it appears in the middle of an edge (a, b) (assuming $z(a) > z(b)$) then two situations may arise.

• if $z(c) > z(\alpha)$ then there exists a point β on the edge (b, c) with $z(\beta) = z(\alpha)$. Here, the triangular region $\Delta b\alpha\beta$ is included in $DFR(s)$ (see Figure 6.4(a)).

• if $z(c) < z(\alpha)$ then there exists a point β on the edge (a, c) with $z(\beta) = z(\alpha)$. Here the quadrilateral $\Box b\alpha\beta c$ is included in $DFR(s)$ (see Figure 6.4(b)).

In order to explain the order of processing of the elements in $\mathcal{Q}$, let us consider a terrain in Figure 6.5, the z-coordinates of all the vertices are given in square bracket, and the DFN(s)'s are marked with dark circles. Now, consider the following situations:

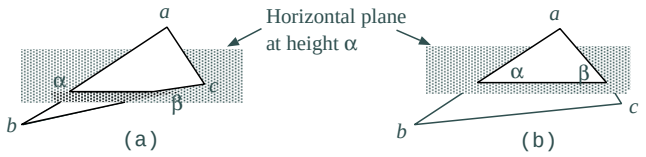

Figure 6.4: DFR of a point α on an edge (a, b) where (a) $z(c) > z(\alpha)$ and (b) $z(c) < z(\alpha)$

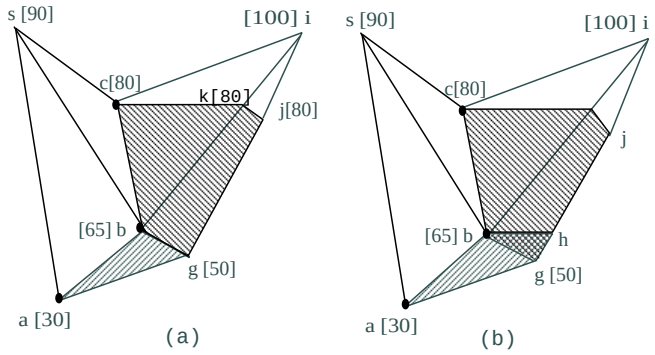

Figure 6.5: Order of processing of the DFNs'

c **is processed prior to** b**:** Here, after processing of c, the region Δcbk is included in $DFR(s)$, and generates the point k as a new DFN. After processing k, $\square kbgj$ is included in $DFR(s)$. Next, when b is processed, Δbga is included in $DFR(s)$ (see Figure 6.5(a)).

b **is processed prior to** c**:** Here, after processing of b, Δabg, and Δbgh ($\subset \Delta bgi$) are included in $DFR(s)$. Now, if we process the point c then, as mentioned above, the $\square cbgj$ is included in $DFR(s)$ (see Figure 6.5(b)).

In the latter situation, Δbgh will be included twice in $DFR(s)$. This situation can be avoided by (i) processing the DFNs' in decreasing order of their z-coordinates using the priority queue, and (ii) maintaining a *flag* with each face which will be set to the value "1" if it is considered during the processing of a DFN. Observation 6.3 along with the above discussion lead to the following algorithm which identifies $DFR(s)$ for a given source vertex s, and stores it in the form of a doubly connected edge list. We also maintain a data structure $PSLG$ for planar point location query using the

projections of the faces of $DFR(s)$ on the XY-plane and following the method described by Kirkpatrick [106].

Algorithm

Input: A triangulated polyhedral terrain, and the source s.

Output: DFR(s) in the form of doubly connected edge list $DCEL$, and
a planar point location data structure $PSLG$.

Data structure: A priority queue $\mathcal{Q}$ to store the unprocessed DFNs' in decreasing order of their z-coordinates.

begin
 put s in $\mathcal{Q}$;
 while $\mathcal{Q}$ is not empty **do**
  $p = \mathcal{Q}(1)$;
  for each face f attached to p **do**
   if $flag$ of face f is not equal to 1 **then**
    compute $DFN(p)$ in face f and insert them in $\mathcal{Q}$;
    set $flag$ of face f to 1;
   endif
  endfor
  compute the faces in $DFR(p)$ attached with point p;
  insert each of them in the data structure $DCEL$
 endwhile
 Use DCEL to construct the $PSLG$ data structure [106]
end.

Lemma 6.5 *The proposed algorithm processes each face at most once, and outputs $DFR(s)$ correctly.*

Proof : The first part of the lemma follows from the use of flag bit during the processing.

For the second part, consider a portion of a face f is in the descent flow region of s but is not included in $DFR(s)$ by our algorithm. Let p be a point in this region having maximum z-coordinate. Surely, p is on an edge of f, and p is not processed as a DFN from $\mathcal{Q}$. Note that, the flow can reach from s to p through a face f' (which is adjacent to f), which is also not included in $DFR(s)$. We can apply the same argument repeatedly to prove that s is not inserted in $\mathcal{Q}$. Thus, we have a contradiction. □

Theorem 6.1 *The proposed algorithm for computing $DFR(s)$ needs $O(n\log n)$ time and $O(n)$ space, and given an arbitrary point t, it searches in $DFR(s)$ data structure in $O(\log n)$ time to report whether a monotone descent path exists from s to t along the surface of $\mathcal{T}$.*

Proof : Each face is processed at most once for inclusion in $DFR(s)$ (by Lemma 6.5), and while processing each (triangulated) face at most three DFNs' are generated (see Figure 6.5). Thus, total number of DFNs' inserted in $\mathcal{Q}$ is $O(n)$ in the worst case. Inserting a part of a face in $DCEL$ requires $O(1)$ time. Since a single operation in a priority queue needs $O(\log n)$ time, $DCEL$ can be constructed in $O(n \log n)$ time. The same argument leads to the fact that $DCEL$ needs $O(n)$ space. Given the $DCEL$, the $PSLG$ data structure can also be constructed in $O(n \log n)$ time using $O(n)$ space [106]. For the query time complexity, see [106]. □

6.3 Shortest Monotone Descent Path on Convex DFR

We now study a restricted version of the descent flow problem where $DFR(s)$ is convex.

Definition 6.6 *Let f and f' be two adjacent faces of the terrain $\mathcal{T} = \{(x, y, \zeta(x, y)), (x, y) \in \Omega\}$ sharing an edge e. The faces f and f' are said to be in convex (resp. concave) position, if for any two points $p \in f$, $q \in f'$ and p, q not lying on e, and for a point κ on the straight line segment (p, q) with $(x(\kappa), y(\kappa), \zeta(x(\kappa), y(\kappa)))$ lying in f or f', then $\zeta(x(\kappa), y(\kappa)) > z(\kappa)$ (resp. $\zeta(x(\kappa), y(\kappa)) < z(\kappa)$).*

Definition 6.7 *Given a terrain $\mathcal{T}$ and a source point s, $DFR(s)$ is said to be convex if every two adjacent faces in $DFR(s)$ are in convex position. Similarly, $DFR(s)$ is said to be concave if its every pair of adjacent faces are in concave position.*

We now study the properties of shortest monotone descent path in a convex $DFR(s)$. The convexity of $DFR(s)$ can be tested very easily by observing the neighbors of its each face. From now onwards, we assume that the $DFR(s)$ on which we are working, is convex.

Observation 6.5 *If p_1, p_2 be two points on a face of $\mathcal{T}$, and p_3 be another point on the line segment $[p_1, p_2]$, then $z(p_1) > z(p_3)$ implies $z(p_2) < z(p_3)$.*

Lemma 6.6 *Let f and f' be two adjacent faces of a polyhedral terrain which are in convex position. f and f' are separated by an edge $e = (a, b)$ where $z(b) > z(a)$. Consider a pair of points p and q on faces f and f' respectively, and a point c on e with $z(p) = z(c)$.*

(a) Now the edge e can be partitioned into two parts $[a, c]$ and $(c, b]$ such that the descent flow from p to the face f' is possible through the portion $[a, c] \in e$ but not possible through the portion $(c, b]$.

(b) Let q^* denote the image of the point q in the planar unfolding of f' onto f. Now, (i) if the line segment $[p, q^*]$ intersects the line segment $[a, c]$ $(\in e)$ in the unfolded plane, then $q \in DFR(p)$ and the geodesic shortest path from p to q through the edge e is the shortest monotone decent path from p to q, and (ii) if $[p, q^*]$ intersects the line segment $(c, b]$ but $q \in DFR(p)$, then $[p, c] + [c, q]$ forms the shortest monotone descent path from p to q through the edge e.

Proof : Part (a) of the lemma is trivial. We now prove part (b) of the lemma.
Let $\pi_{geo}(p, q; e)$ denote the geodesic shortest path from p to q passing through the edge e. If the line segment $[p, q^*]$ (in the unfolded plane) intersects e (at a point, say η) in its

interior, then by Lemma 6.1, the image of $\pi_{geo}(p, q; e)$ in the unfolded plane coincides with the line segment $[p, q^*]$. Now, two cases need to be considered: (1) $z(\eta) \leq z(p)$ and (2) $z(\eta) > z(p)$.

Case 1: By Observation 6.5, $z(q^*) < z(\eta)$. As the two faces f and f' are in convex position, $z(q) \leq z(q^*)$. Thus both the line segments $[p, \eta]$ and $[\eta, q]$ are monotone descent (see Figure 6.6(a)), and part (i) of the lemma follows.

Case 2: Here the line segment $[p, \eta]$ is not monotone descent in the plane f. Consider any monotone descent path from p to q which intersects the line segment $[a, c]$ (at a point, say η'). Note that, the length of such a path remains same as that of its image in the unfolded plane, and it attains minimum when $\eta' = c$ as illustrated in Figure 6.6(b). This proves part (ii) of the lemma. □

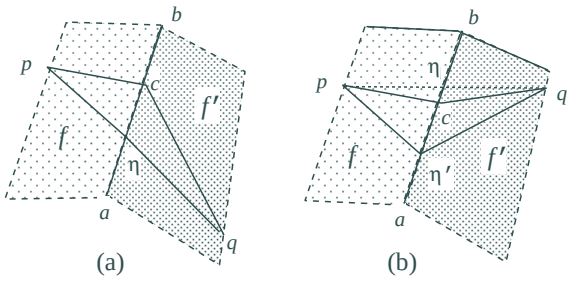

Figure 6.6: Proof of Lemma 6.6

Let v be a vertex of $\mathcal{T}$ and p be a point in $DFR(v)$ which is reachable from v through a sequence of edges $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$ of $DFR(v)$; the faces f_{i-1} and f_i, attached to edge e_i, are in convex position; $v \in f_0$, $p \in f_m$. Now, let R^* denote the region obtained by the planar unfolding $U(\mathcal{E})$. It is possible that R^* is self-overlapping. Now we have the following result:

Lemma 6.7 *If p^* denotes the image of the point p in R^*, and the line segment $[v, p^*]$ intersects the images of the edges $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$ in that order, then the path $\pi(v, p)$ on $\mathcal{T}$, whose image in R^* is the line segment $[v, p^*]$, is the shortest monotone descent path from v to p through the faces $\{f_0, f_1, f_2, \ldots, f_m\}$.*

Proof : From Lemma 6.1, the path $\pi(v, p)$ is a shortest geodesic path through $\{f_0, f_1, f_2, \ldots, f_m\}$. Next, we show that $\pi(v, p)$ is a monotone descent path.

Let c_i be the point of intersection of $\pi(v, p)$ with the edge e_i. Since $\pi(v, p)$ passes through $DFR(v)$, $z(c_i) < z(v)$ for all $i = 1, \ldots, m$. Rename $v = c_0$ and $p = c_{m+1}$. Now, by repeated application of the proof technique of Lemma 6.6, it can be shown that $z(c_0) > z(c_1) > z(c_2) > \ldots > z(c_m) > z(c_{m+1})$. □

Remark 6.2 *If the shortest monotone descent path from a vertex v to a point p is obtained as in Lemma 6.7, then the point p is straight-line reachable from the vertex v.*

Remark 6.3 *Let e be an edge of $\mathcal{T}$ separating the faces f and f'. A point p on e may be straight line reachable from a vertex v through different edge sequences. Thus, p may be straight line reachable from v through both f and f'.*

Definition 6.8 *A point α on a line segment $[a, b]$ (portion of an edge) is said to be the* frontier point *with respect to a vertex v if α is straight line reachable from v through an edge sequence $\mathcal{E}$ and it is the closest point of v on the line segment $[a, b]$.*

It is easy to see that α can be either a or b or the perpendicular projection of v on the line segment $[a, b]$ in the planar unfolding R^*.

The above discussions lead to a preprocessing step of $DFR(s)$ similar to [121]. It splits each face f of $DFR(s)$ into homogeneous partitions such that for every point p in a partition the shortest monotone descent path from s reaches p through the same edge sequence. Note that, each of these partitions must be maximal in the sense that it is not properly contained in some other homogeneous partition of f. We will refer the data structure storing this homogeneous partitions of $DFR(s)$ by $HDFR$.

Definition 6.9 *A segment $I = [a, b]$ on an edge $e \in DFR(s)$ is said to be a homogeneous segment (or h-segment in short) if for every point $\alpha \in I$, the shortest monotone descent path from s to α passes through the same edge sequence.*

6.3.1 Preprocessing

Our algorithm for finding shortest monotone descent path on convex $DFR(s)$ creates a data structure *HDFR* in two phases. In Phase 1, each edge $e = (a, b)$ of $DFR(s)$ is split into h-segments $\{I_i = [a_i, a_{i+1}], i = 0, \ldots, k-1\}$, $a_0 = a$, $a_k = b$, $\bigcup_{i=0}^{k-1} I_i = e$. The points $a_0, a_1, a_2, \ldots, a_k$ are referred to as *break-points*. In Phase 2, the interior of each face in $DFR(s)$ is split into homogeneous partitions (similar to Voronoi partitions). Below, we describe Phase 1 and Phase 2 in detail. The $HDFR$ data structure is similar to the data structure for storing $DFR(s)$ as defined in Section 6.2; but each edge e in the $DCEL$ data structure is attached with an associated structure $AVL(e)$ which is defined in Phase 1, and each face $f = \Delta abc$ in the $DCEL$ data structure is attached with three associated structures $VOR_{ab}(f)$, $VOR_{bc}(f)$ and $VOR_{ac}(f)$ which are defined in Phase 2.

Phase 1: Let p be a point on the surface of $\mathcal{T}$ which is straight-line reachable from a vertex $r \in DFR(s)$. The shortest monotone descent path from s to p passing through the vertex r, denoted by $\pi_r(p)$, is the concatenation of $\pi_{md}(s, r)$ and the line segment $[r, p]$. Its length is $\delta_r(p) = \delta(s, r) + d(r, p)$, where $d(r, p)$ denotes the length of the straight line segment $[r, p]$ in the unfolded plane.

Definition 6.10 *Let $I = [a, b]$ be an h-segment on an edge e such that $\pi_{md}(s, \alpha) = \pi_r(\alpha)$ for every point $\alpha \in I$, then the vertex r is said to be the link-vertex for the h-segment I.*

The end points of the h-segments are the break-points. If r is the link-vertex of a h-segment I, and $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$ be the edge sequence which are intersected by the line segment $[r, \alpha]$ for every point $\alpha \in I$, then the last edge e_m in $\mathcal{E}$ is called the predecessor of I in the *HDFR* data structure. If $[r, \alpha]$ does not intersect any edge, then the predecessor of I is r itself. Now, we have the following remarks.

Remark 6.4 *If a_i is a break-point on an edge e, and is shared by two h-segments $[a_{i-1}, a_i]$ and $[a_i, a_{i+1}]$ with link vertices r and u respectively, then $\delta_r(a_i) = \delta_u(a_i)$.*

Remark 6.5 *If $I_1 = [a_1, b_1]$ and $I_2 = [a_2, b_2]$ are two h-segments on an edge e (adjacent to faces f and f') with link-vertex r_1 and r_2 respectively, and both I_1 and I_2 are reachable from r_1 and r_2 respectively through the same face f, then I_1 and I_2 have mutually disjoint interiors.*

Thus, the h-segments generated on an edge e are non-overlapping, and can be orderly maintained in an AVL-tree, called $AVL(e)$. Each of these intervals is attached with its (i) predecessor and (ii) link-vertex. In addition, we need to compute the frontier-point on each h-segment $[a, b]$ with respect to its link-vertex. A vertex of $DFR(s)$ in the *HDFR* data structure is also attached with its predecessor and link-vertex; these can be defined in a manner similar to the h-segments in *HDFR*.

During the execution of the *HDFR* creation algorithm, we use a MIN_HEAP as the event queue. It contains all the vertices, break-points and frontier-points explored so far. Execution starts by putting s in MIN_HEAP with $\delta(s, s) = 0$, and proceeds in a manner similar to Dijkstra's shortest path algorithm. But, unlike Dijkstra's algorithm, here the event-points are generated during the execution and are inserted in the MIN_HEAP. The elements in the MIN_HEAP are the points where the monotone descent path from s exist, but for each of these points, there is a possibility of obtaining an alternate path of smaller length, except the one, say v, having minimum δ-value. Thus, each element α in the MIN_HEAP is attached with $\delta(s, \alpha)$ (explored so far), and a pointer field, called *self_ptr*. The self_ptr field points to the h-segment in the $HDFR$, which has introduced the point α in the MIN_HEAP data structure.

Each time, we choose a member v from the MIN_HEAP having minimum δ-value, and process its adjacent face(s) to include some more region in $DFR(s)$. The link-vertex attached to v is u, and the h-segment attached to v is $[\alpha, \beta]$, where $[\alpha, \beta]$ is on an edge $e_0 = (a, b)$. We process the face $f = \Delta abc$ which is adjacent to e_0, and is in the other side of the vertex u. Let us name the other two edges of face f as $e_1 = (a, c)$, $e_2 = (b, c)$ respectively. We need to consider two distinct cases: (i) v is not a vertex of $DFR(s)$, (see Figure 6.7) and (ii) v is a vertex of $DFR(s)$ (see Figure 6.9).

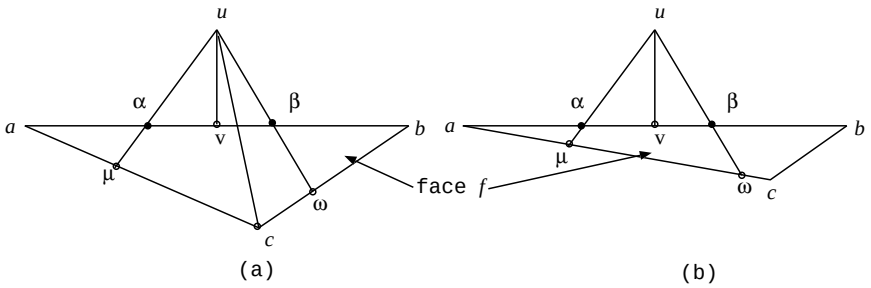

Figure 6.7: Processing a point v which is not a vertex of $DFR(s)$

v is not a vertex of $DFR(s)$

Let $\mathcal{R}$ be the planar unfolding of all the faces intersected by the path $\pi(u, v)$. We unfold face f onto $\mathcal{R}$, join (u, α) and (u, β) and extend them inside face f. These lines hit the boundary of f at μ and ω respectively. Let I be the portion of the boundary of f from μ to ω, which is straight-line reachable from u in the planar unfolding $\mathcal{R}$. I contributes one or two h-segments in $HDFR$ depending on whether it is a single interval (on either e_1 or e_2) or contains the vertex c (see Figures 6.7(a) and 6.7(b) respectively). We compute $\delta_u(\mu)$ and $\delta_u(\omega)$ and identify the interval $[\mu, \omega]$ in the $AVL(e)$ attached to edge e in the *HDFR* data structure.

If I does not overlap with the existing h-segments then we (i) insert the h-segment $[\mu, \omega]$ in $AVL(e)$, (ii) insert μ and ω in MIN_HEAP with respect to $\delta_u(\mu)$ and $\delta_u(\omega)$ respectively, (iii) insert the frontier-point $\phi \in [\mu, \omega]$ (with respect to $\delta_u(\phi)$) if it does not coincide with any of μ and ω, and (iv) set the self_ptr of μ, ω and ϕ to point $I = [\mu, \omega]$ in *HDFR* data structure.

If I overlaps with the h-segments $\{J_i = [\phi_i, \psi_i], i = 1, \ldots, k\}$, $k \geq 1$ on an edge e, $\mu \in J_1$ and $\omega \in J_k$ (see Figure 6.8(a)), and the link-vertex attached to J_i is r_i, then we consider each interval $J_i, i = 1, 2, \ldots, k$ in order. For each J_i, we compute $\delta_{r_i}(\phi_i)$, $\delta_{r_i}(\psi_i)$, $\delta_u(\phi_i)$ and $\delta_u(\psi_i)$. Depending on the relationship among these four quantities, we may have to replace J_i in $AVL(e)$ by some newly generated h-segments as described below. The same technique was followed in [121]. If $J_i = [\phi_i, \psi_i]$ needs to be replaced then we split

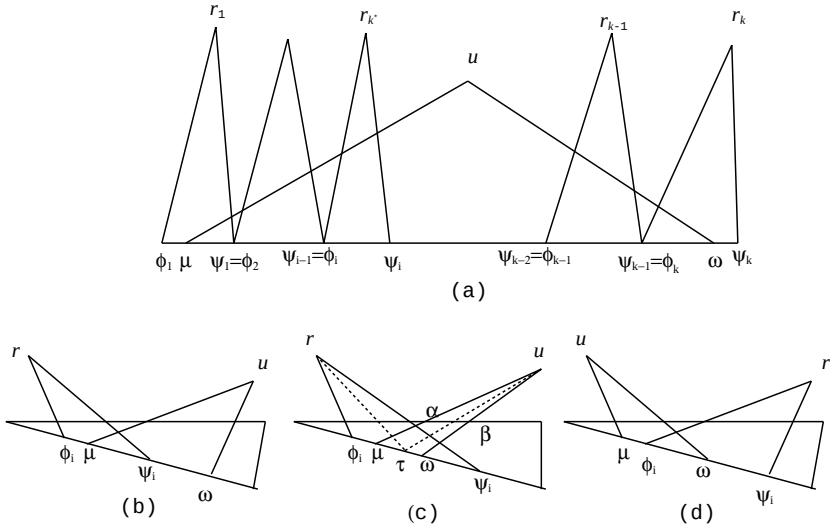

Figure 6.8: Overlaps of I with other h-segments

it two or three pieces depending on the following situations:

Case 1 ($\mu \in [\phi_i, \psi_i]$ but $\omega \notin [\phi_i, \psi_i]$): Here J_i splits into two pieces, namely $J_{1i} = [\phi_i, \mu]$ and $J_{2i} = [\mu, \psi_i]$ (see Figure 6.8(b)).

Case 2 ($\mu, \omega \in [\phi_i, \psi_i]$): Here J_i splits into three pieces, namely $J_{1i} = [\phi_i, \mu]$, $J_{2i} = [\mu, \omega]$ and $J_{3i} = [\omega, \psi_i]$ (see Figure 6.8(c)).

Case 3 ($\mu \notin [\phi_i, \psi_i]$ but $\omega \in [\phi_i, \psi_i]$): Here J_i splits into two pieces, namely $J_{2i} = [\phi_i, \omega]$ and $J_{3i} = [\omega, \psi_i]$ (see Figure 6.8(d)).

In either of these cases, we identify the *tie-point* $\tau \in J_{2i}$ (see p. 658 of [121]), where $\delta_{r_i}(\tau) = \delta_u(\tau)$ (if it exists). If the tie-point (τ) is found, then it splits J_{2i} into two intervals. Finally, we delete J_i and insert all the newly generated h-segments (with their corresponding predecessors and link-vertices) in $AVL(e)$.

After processing all the J_is' for $i = 1, 2, \ldots, k$, we replace all the h-segments in $AVL(e)$ having the same link-vertex u by a single h-segment which is obtained by merging them.

If the above steps are executed for at least one J_i, then we insert μ (or the corresponding tie-point) and ω (or the corresponding tie-point) as break-points in the MIN_HEAP along with their respective δ-values and self_ptr. For each newly inserted h-segment the corresponding frontier-point is also to be inserted in MIN_HEAP.

v is a vertex of $DFR(s)$

Let the h-segment attached to v be $[v, \alpha]$. We unfold the face f onto $\mathcal{R}$, and extend the straight lines $L_1 = (u, v)$ and $L_2 = (u, \alpha)$ beyond v and α respectively. If both L_1 and L_2 go outside f then both the edges e_1 and e_2 of f are straight line reachable from u, and they will be considered as h-segments (see Figure 6.9(a)). If one or both of L_1 and L_2 hit(s) the boundary of f, then one or two h-segments will be generated (see Figures 6.9(b), 6.9(c)). For each of them, link-vertex is u and predecessor is $[v, \alpha]$.

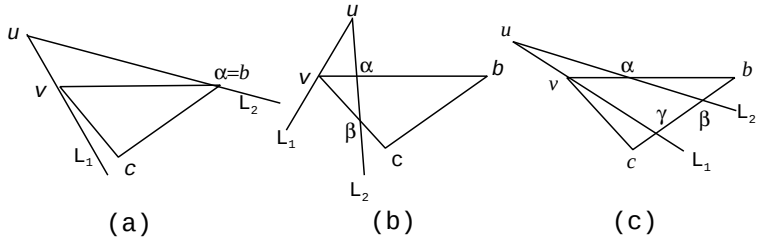

Figure 6.9: Processing a vertex

In addition, we need to consider each edge e in $IDFR(v)$ (defined in Section 6.2) which are not adjacent to v. If e (or a portion of e) lies to the other side of α with respect to the line L_1, then it is considered as h-segment with link-vertex and predecessor both equal to v. Similar technique is adopted in [121], but the monotone descent property from v was not required there.

Each of these newly generated h-segments may overlap on some existing h-segments, and it need to be tackled using the same technique as before. Finally, all newly generated h-segment are inserted in respective AVL-trees of the *HDFR* data structure, and all the break-points and a frontier-point are inserted in MIN_HEAP as in [121]. If v is observed to be a break-point of a h-segment, it is also to be inserted in MIN_HEAP for processing the other faces incident to v.

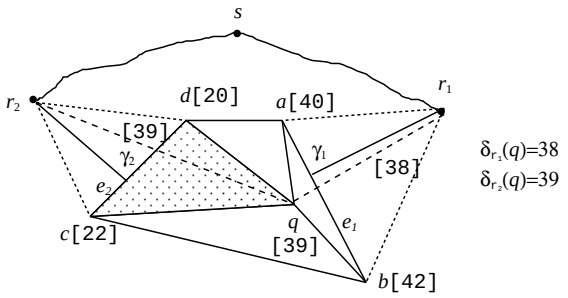

Figure 6.10: Demonstration of frontier point

Role of the frontier-points

Consider the planar unfolding along an edge sequence of the terrain $\mathcal{T}$. The break-points a, b, c and d are stored in MIN_HEAP and their distances are shown inside square brackets in Figure 6.10. If we do not consider the frontier-point then after processing d from MIN_HEAP, the point q will be pushed in the MIN_HEAP, with $\delta(s,q) = \delta_{r_2}(q) = 39$ (say). As $\delta(s,q) < min(\delta(s,a), \delta(s,b))$, after processing c from MIN_HEAP, q will be chosen for processing. This implies, the distance of q will not be reduced further by Lemma 6.8. But Figure 6.10 shows that q is also straight-line reachable from r_1 and $\delta_{r_1}(q) = 38$. Note that, if we consider frontier-points γ_1 and γ_2, then $\delta_{r_2}(q) > min(\delta(s,\gamma_2), \delta(s,\gamma_1))$. Thus, either γ_1 or γ_2 will be chosen prior to the processing of q, and $\delta(s,q)$ will be set correctly prior to its processing as described in the following lemma.

Lemma 6.8 *Every time the path-length attached to the top-most element of the MIN_HEAP is optimum.*

Proof : [By contradiction] Let the distance attached to q in MIN_HEAP represent the length of the path π_1 from s to q through the link-vertex r_1, and through the edge sequence $\mathcal{E}_1$ (see Figure 6.11). Suppose π_1 is not optimum; there exists another path π_2 from s to q through the link-vertex r_2 and passing through the edge sequence $\mathcal{E}_2$, such that $\delta_{r_1}(q) > \delta_{r_2}(q)$. Let $e_k \in \mathcal{E}_2$ and closest to q. Since $\delta_{r_2}(r_2) < \delta_{r_2}(q) < \delta_{r_1}(q)$, and q is

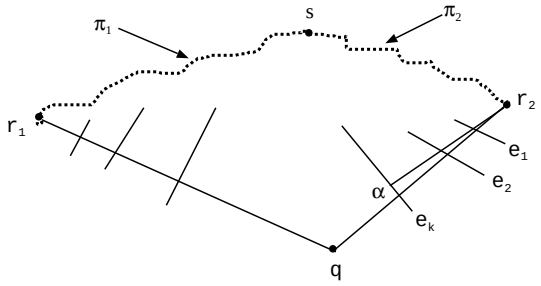

Figure 6.11: Proof of Lemma 6.8

currently being processed, r_2 is already processed. Moreover, since π_2 passes through e_k, there exists a frontier point, say α, on e_k with respect to r_2, and $\delta_{r_2}(\alpha) < \delta_{r_2}(q) < \delta_{r_1}(q)$. While processing α, we will discover q, and compute $\delta_{r_2}(q)$. Thus, the hypothesis that distance attached to $q = \delta_{r_1}(q)$ is not correct. □

Phase 2: In this phase, we compute the three homogeneous partitions inside each face separately. Let us consider a face $f = \Delta abc$. We compute the homogeneous partition with respect to all the three edges of face f as follows.

Consider the h-segments on edge $e = (a, b)$ of face f; and their link-vertices are considered as weighted points. The weight of a link-vertex v is $\delta(s, v)$. We unfold the faces of the terrain such that the link-vertices of all the h-segments on edge e belong to the same plane containing f. The homogeneous partition $VOR_{ab}(f)$ of the face f is the Voronoi partition of the interior of face f with respect to those weighted points. This can be computed in $O(K\log K)$ time, where K is the number of h-segments on the boundary of f. Each such partition points to its corresponding h-segment in the *HDFR* data structure. The detailed description of this technique is available in [121]. Similarly $VOR_{bc}(f)$ and $VOR_{ac}(f)$ are obtained.

6.3.2 Query answering

For a given query point t, we first locate the face $f = \Delta abc$ of $DFR(s)$ containing t. Next, we need to search in the three data structure $VOR_{ab}(f)$, $VOR_{bc}(f)$ and $VOR_{ac}(f)$

separately. The shortest path from s to t which enters face f through the edge (a, b) (of f) can be obtained by searching $VOR_{ab}(f)$. This actually gives the h-segment I_{ab} through which the shortest path from s to t has entered the face f crossing the edge (a, b), and the length of the corresponding path (see Theorem 6.3). Similarly, the other two data structures also report the corresponding h-segments and the distances. The minimum among these three distances will indicate the length of the optimum path. Let it correspond to the h-segment I_t. We use predecessor links of the h-segments to obtain the edge (to be precise, h-segment) sequence $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$, to reach the link-vertex r (a vertex of $DFR(s)$) attached to I_t. Thus, the shortest monotone descent path $\pi(r, t)$ passes through faces $f_0, f_2, \ldots, f_m = f$, where r is a vertex of f_0, and $t \in f_m$, and the edge e_i is shared by f_{i-1} and f_i. Finally, we obtain the entire path $\pi(s, t)$ using the following steps.

• Compute $\pi(r, t)$ through the edge sequence $\mathcal{E}$ as mentioned above.

• From r, reach its link-vertex r_1 through an edge sequence obtained using predecessor pointers. Compute the planar unfolding of the faces, adjacent to that edge sequence. The inverse image of the line segment $[r_1, r]$ in the unfolded plane is the shortest descent flow path from r_1 to r.

• Treat r_1 as r, and repeat step (ii) until the vertex s is reached.

6.3.3 Complexity analysis

The total number of vertices and edges in $DFR(s)$ are both $O(n)$. Let us consider the h-segments on an edge e coming through one of its adjacent faces. Each such h-segment is designated by two lines originating from its link vertex and is supported by some other vertex of $\mathcal{T}$. Note that, one vertex can not support more than one such lines. Thus, the number of such h-segments is $O(n)$ in the worst case. The final set of h-segments on an edge is obtained by merging the two sets of h-segments coming through its two adjacent faces, which is also $O(n)$. Each h-segment is attached with a frontier-point. Thus, the

total number of event-points pushed in the MIN_HEAP may be $O(n^2)$ in the worst case. Processing of all these event-points needs $O(n^2 \log n)$ time. Finally, in Phase 2, the time complexity of homogeneous partitioning of all the faces is $O(n^2 \log n)$ in total, and it produces $O(n^2)$ homogeneous partitions. The worst case size of the $HDFR$ is $O(n^2)$.

The query time needs $O(\log n + k)$, where $O(\log n)$ time is required for four point location queries for t as mentioned in Subsection 6.3.2, and k is the number of line segments on the shortest path from s to t. Thus we have the following theorem:

Theorem 6.2 *Given a polyhedral terrain $\mathcal{T}$ with n vertices, and a source point s, our algorithm (i) creates the $HDFR$ data structure in $O(n^2 \log n)$ time and $O(n^2)$space. (ii) For a given query point t, it outputs a monotone descent path from s to t through a sequence of faces in convex position (if exists) in $O(k + \log n)$ time, where k is the number of line segments on the optimal path.*

6.3.4 A simple variation: the distance query

A simpler version of the above problem is the *distance query*, where the objective is to compute the length of the monotone descent path from s to a given query point t. We show that, a minor tailoring of the *HDFR* data structure helps us to report the length in $O(\log n)$ time.

Recall the Phase 2 of the preprocessing. Here, we have assumed that the length of the shortest path of each link-vertex is already known. At each face, we have considered the link-vertices attached to the h-segments on its boundary as weighted points, where the weight attached to a link vertex is the length of the shortest monotone descent path from s to that point. Next, we computed the Voronoi diagram of those weighted points inside that face. At each partition, we attach two scalar information: (i) the coordinate of the image of its corresponding link-vertex in the unfolded plane, and (ii) the distance of that link-vertex from s.

During the distance query for a query point t, we identify the partition in which it

belongs by point location. Let r be the link-vertex attached to that partition. The length of the shortest monotone path from s to t is obtained by $\delta(s,r) + d(r,t)$, where $d(r,t)$ is the Euclidean distance of this point from the the image of the link-vertex r in the unfolded plane. Thus, we have the following result:

Theorem 6.3 *Given a polyhedral terrain $\mathcal{T}$ with n vertices, and a source point s, the revised $HDFR$ data structure can be created in $O(n^2\log n)$ time and $O(n^2)$ space, such that given any arbitrary query point t, the distance query can be answered in $O(\log n)$ time.*

6.4 Shortest Monotone Descent Path through Parallel Edge Sequence

In this section, we shall consider a slightly different problem on a general terrain where each pair of adjacent faces are not restricted to only in convex position. Here, along with the source (s) and destination (t) points, a sequence of faces $\mathcal{F} = \{f_0, f_1, \ldots, f_m\}$, $s \in f_0$, $t \in f_m$, is given. The objective is to find the shortest descent flow path through $\mathcal{F}$. This problem in its general form seems difficult. But we are proposing an efficient solution in a restricted setup. Let $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$ be the sequence of edges separating the consecutive faces in $\mathcal{F}$. We assume that the members in the edge sequence $\mathcal{E}$ are parallel to each other. Note that, here we are deviating from the assumption that the faces in the terrain are triangular.

The problem is very much similar to the *L_2-shortest path problem over walls*, defined in [123]. Here a set of n vertical walls parallel to x-axis are given. Each wall is positioned on the xy-plane. The i-th wall, is positioned at $y = a_i$, and its top boundary, denoted by e_i, is a line of the form $z = b_i x + c_i$, where a_i, b_i and c_i are given constants, $a_1 < a_2 < \cdots < a_n$. The objective is to report the L_2-shortest path between a given pair of query points s and t, where $s < a_1$ and $t > a_n$. They proved that, the shortest

path is always monotone with respect to the y-axis, and it bends on the edges e_i. It is also proved that, the shortest path from s to t is the concatenation of two sub-paths, one of them is monotone ascending and the second one is monotone descending with respect to z-coordinate. The standard method of solving this problem involves a preprocessing phase which splits each edge e_i into segments, and then defines the shortest path map [135]. The optimal L_2-shortest path from s to t can be obtained by following an appropriate path in that map. It is proved that the size of the shortest path map is $O(n^2)$ in the worst case, but finding a polynomial time algorithm for constructing the map is left as an open problem [123]. Our case is a simpler version, where the plane between two consecutive edges is known, and we are specifically searching for a monotone descent path which is constrained to lie on the plane.

6.4.1 Properties of parallel edge sequence

Lemma 6.9 *Let p and q be two points on two consecutive members e_i and e_{i+1} of $\mathcal{E}$ which bound a face f, and $z(p) = z(q)$. Now, if a line ℓ on face f intersects both e_i and e_{i+1}, and is parallel to the line segment $[p, q]$, then (i) the length of the portion of ℓ lying in face f is equal to the length of the line segment $[p, q]$, and (ii) all the points on ℓ have same z-coordinate.*

Proof : Part (i) of the lemma follows from the fact that as e_i and e_{i+1} are parallel, the portion of ℓ on f and the line segment $[p, q]$ appear as two parallel edges of a parallelogram on face f.

Part (ii) of the lemma trivially follows if the face f is horizontal. So, we prove it for the case where f is not horizontal.

Consider a horizontal plane h at altitude $z(p)$. The intersection of the face f and the plane h is the line segment $[p, q]$ (by part (i) of this lemma). Consider another horizontal plane h' through a point r on line ℓ. The intersection of f and h' must be parallel to $[p, q]$, and hence it coincides with the line ℓ. Thus, all the points on the line ℓ have the same z-coordinate. □

Lemma 6.10 *Let e_i and e_{i+1} be two edges in $\mathcal{E}$ bounding a face f. For a pair of points $p, p' \in e_i$ and a pair of points $q, q' \in e_{i+1}$, if $z(p) > z(p')$ and $z(q) > z(q')$, then the line segments $[p, q]$ and $[p', q']$ do not intersect on face f; but the line segments $[p, q']$ and $[p', q]$ must intersect on face f.*

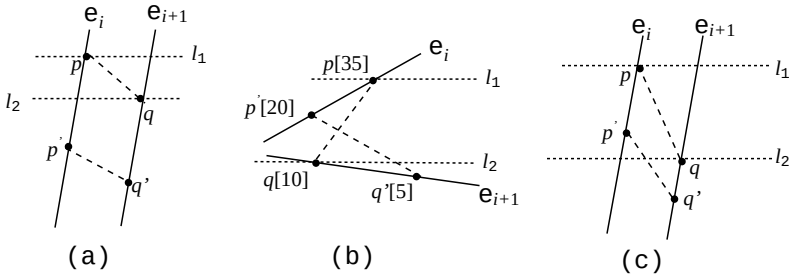

Figure 6.12: Proof of Lemma 6.10

Proof : Without loss of generality, assume that $z(p) > z(q)$. Draw two horizontal planes h_1 and h_2 through p and q respectively which intersect face f along lines ℓ_1 and ℓ_2 respectively. Note that, ℓ_1 is above ℓ_2 with respect to their z-coordinates. Here any one of the three cases may arise: (i) both p' and q' are above h_2, (ii) both p' and q' are below h_2, (iii) p' and q' appear in different sides of h_2. Case (i) is impossible since $z(q) > z(q')$. In Case (ii), the line segment $[p, q]$ and $[p', q']$ appear in different sides of the plane h_2, and hence they can not intersect (see Figure 6.12(a)). In Case (iii), $z(p') > z(q)$ and $z(q') < z(q)$. Here, if $[p, q]$ and $[p', q']$ intersect, then e_i and e_{i+1} can not be parallel (see Figure 6.12(b)). The reason is that, as e_i and e_{i+1} are parallel and $z(p) > z(p') > z(q)$ & $z(q) > z(q')$, then $[p, q]$ and $[p', q']$ are the sides (not the diagonals) of the trapezoid $\Box pp'q'q$ (see Figure 6.12(c)). □

Theorem 6.4 *Let f_1 be a non-horizontal face bounded by two parallel edges $e_1 = [a_1, b_1]$ and $e_2 = [a_2, b_2]$ ($z(a_i) < z(b_i)$, $i = 1, 2$); the point s appears in its adjacent face f_0 such that f_0 and f_1 are separated by the edge e_1. If there exist a pair of points $p \in e_1$ and $q \in e_2$ with $z(p) = z(q) < z(s)$, and the points s, p, q^* (q^* is the image of the point q in the planar unfolding $U(e_1)$) are collinear, then*

(i) for any point α in the interval $[q, a_2]$, shortest monotone descent path along e_1 is the inverse-image of the straight line segment $[s, \alpha^*]$ in the unfolded plane provided $[s, \alpha^*]$ intersects the edge e_1 in its interior.

(ii) for any point α in the interval $[b_2, q]$, shortest monotone descent path along e_1 is not an inverse-image of the straight line segment $[s, \alpha^*]$ in unfolded plane. Here $\pi_{md}(s, \alpha)$ will pass through a point $\beta \in [b_1, p]$ with $z(\beta) = z(\alpha)$ in the original terrain.

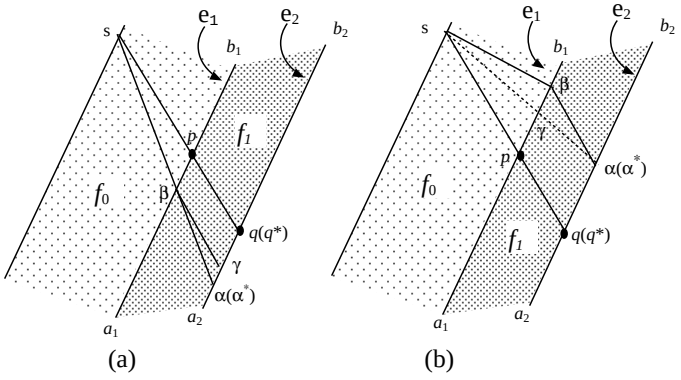

Figure 6.13: Proof of Theorem 6.4

Proof : The line segment $[p, q]$ partitions the face f_1 into two parts, and the points b_1 and b_2 belong to the same side of $[p, q]$ (by Lemma 6.10). Consider a point $\alpha \in [q, a_2]$ on the edge e_2 (see Figure 6.13(a)). In the planar unfolding $U(e_1)$, the straight line segment $[s, \alpha^*]$ intersects e_1 at a point, say β. By Lemma 6.10, the line segment $[\alpha, \beta]$ is below the line segment $[p, q]$. Thus, if β is in the interior of the edge e_1 then $\beta \in [p, a_1]$. Let us consider a line segment $[\beta, \gamma]$ on the face f_1 which is parallel to $[p, q]$, and γ is on the edge e_2. Now consider the triangle $\Delta s q^* \alpha^*$ in the unfolded plane, where the point β lies on $[s, \alpha^*]$. As the line segment $[\beta, \gamma^*]$ is parallel to $[s, q^*]$, γ lies on $[q^*, \alpha^*]$. So, $z(\alpha) < z(\gamma) < z(q)$. By Lemma 6.9, $z(\gamma) = z(\beta)$. Hence part (i) of the lemma follows.

The proof of part (ii) follows from the following argument. Consider a point $\alpha \in [q, b_2]$ (See Figure 6.13(b)); the line segment $[s, \alpha^*]$ intersects the edge e_1 at γ in the unfolded plane $U(e_1)$. Draw a line segment $[\alpha, \beta]$ on face f_1 which is parallel to $[p, q]$. As

$z(\beta) = z(\alpha)$ (by Lemma 6.9), we have $z(\gamma) < z(\alpha)$. Thus, the shortest monotone descent path from s to α can not be the geodesic shortest path between them. As $z(\beta) = z(\alpha) < z(s)$, all the monotone descent path from s to α through edge e_1 must pass through the line segment $[\beta, b_1]$. Hence, from triangle inequality, we can conclude that the shortest monotone descent path from s to α will be the concatenation of line segments $[s, \beta]$ and $[\beta, \alpha]$. □

In order to describe our algorithm, let us introduce the following terminology.

We obtain the planar unfolding of the faces $\mathcal{F} = \{f_1, f_2, \ldots, f_m\}$ onto face f_0, and use a two dimensional coordinate system for the entire unfolded plane such that the members in the edge-sequence $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$ are ordered from left to right, and each of them is parallel to the y-axis. The z-coordinate of a point in the unfolded plane indicates the z-coordinate of that point in the original terrain. In this planar unfolding, if an edge e_i of the terrain is represented as $[a_i, b_i]$, with $y(a_i) < y(b_i)$ then $z(a_i) \leq z(b_i)$ (see Lemma 6.9). The source s is in f_0, then flow passes through the edge sequence $\mathcal{E}$ to reach a point $t \in f_m$. If a path $\pi(s, t)$ enters into a face f_i along a line ℓ_i, then the *angle of incidence* of $\pi(s, t)$ in face f_i (with edge e_i) is denoted by θ_i, and henceforth will be referred to as *slope* of ℓ_i.

Let e_1 and e_2 be two parallel boundaries of a face f. The *translation event* for face f, denoted by $T(f)$ is a linear translation of e_2 on e_1 such that the entire face f is merged to the line e_1 as follows:

The points in the unfolded plane lying on the same side of s with respect to e_1 remain unchanged.

Each point p lying in the proper interior of the face f is mapped to a point $q \in e_1$ such that $z(p) = z(q)$.

Each point $p = (x_p, y_p)$ on the edge e_2 is mapped to a point $q = (x_q, y_q)$ on the edge e_1 such that $z(p) = z(q)$. Under this transformation $x_q = x_p + \alpha$, $y_q = y_p + \beta$, where α, β are constants, and they depend on the slope and width of face f.

Each point (x, y) in the unfolded plane lying on the other side of s with respect to e_2 is moved to the point $(x + \alpha, y + \beta)$.

The slope of the line containing (p, q) is referred to as *merging direction* of face f, and is denoted as $\phi(f)$. Theorem 6.4 indicates the following result.

Corollary 6.4.1 If the slope θ of a line segment ℓ in face f is such that (i) $\theta < \phi(f)$ then ℓ is strictly monotone descent, (ii) $\theta = \phi(f)$ then all points in ℓ have same z-coordinate, and (iii) $\theta > \phi(f)$ then ℓ is strictly monotone ascent.

Let $\pi_{md}(s, t)$ be the shortest monotone descent path from $s \in f_0$ to $t \in f_m$ passing through a sequence of parallel edges $\{e_1, e_2, \ldots, e_{m-1}\}$. Along this path there exist some faces $\{f_{j_1}, f_{j_2}, \ldots, f_{j_k}\}$ such that all the points on the path $\pi_{md}(s, t)$ in face f_{j_i} have same z-coordinate ξ_{j_i}; the portions of the path in all other faces are strictly monotone descent. Now, we have the following theorem.

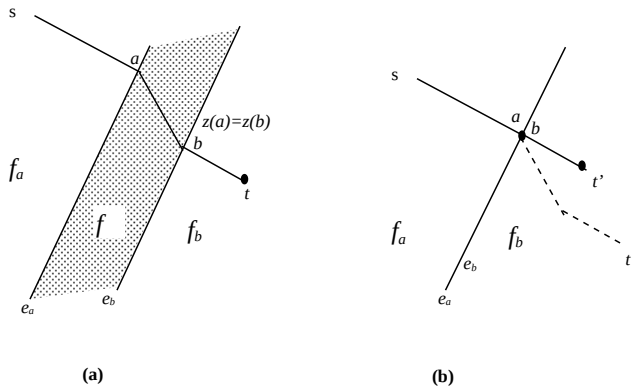

Figure 6.14: Demonstration of translation event where face f is merged to the line e_a

Theorem 6.5 *If the translations $T(f_{j_1}), T(f_{j_2}), \ldots, T(f_{j_k})$ are applied (in any order) on the unfolded plane of faces $f_0, f_1, \ldots, f_m$ then the shortest monotone descent path $\pi_{md}(s, t)$ will become a straight line segment from s to t in the transformed plane.*

Proof : Let us first assume that $k = 1$, i.e., $\pi_{md}(s, t)$ passes through a face f with all points having the same z-coordinate. Let f_a and f_b be its preceding and succeeding faces

with separating edges e_a and e_b respectively. We also assume that $\pi_{md}(s,t)$ consists of three consecutive line segments $[s,a]$, $[a,b]$, $[b,t]$ lying in f_a, f and f_b respectively. Note that, all the points on $[a,b]$ have same z-coordinate. If we apply $T(f)$, the points b and t will be mapped to a and t' (see Figure 6.14). Now, in the transformed plane, the shortest path from s to t' is the straight line segment $[s,t']$. We argue that $[s,t']$ will pass through a. On the contrary, assume that $[s,t']$ intersect e_a at a', and a' is the image of $b' \in e_b$ under $T(f)$, $b' \neq b$. Thus, $d(s,a') + d(a',t') < d(s,a) + d(a,t')$. Now, applying reverse transformation, $d(s,a') + d(b',t) < d(s,a) + d(b,t)$. From Lemma 6.9, $d(s,a') + d(a',b') + d(b',t) < d(s,a) + d(a,b) + d(b,t)$. This leads to a contradiction.

Let there exist several faces on the path $\pi_{md}(s,t)$ such that all the points of $\pi_{md}(s,t)$ in that face have same z-coordinate. If we apply transformation T on one face at a time, the above result holds. The order of choosing the face for applying transformation T is not important due to the following argument: (i) a point p on the unfolded plane will be affected due to the same set of transformation irrespective of in which order they are applied, and (ii) the effects of all the transformations affecting on a point are additive. □

Lemma 6.11 *If the shortest monotone descent path $\pi_{md}(s,t)$ passes through a sequence of parallel edges, then all the line segments of $\pi_{md}(s,t)$, which are strictly monotone descent, are parallel in the unfolded plane of all the faces.*

Proof : Follows from Theorem 6.5. □

Theorem 6.6 *If the line segments of shortest monotone descent path $\pi_{md}(s,t)$ in faces $f_{1^*}, f_{2^*}, \ldots, f_{k^*}$ are strictly monotone descent then their slopes are equal. The slope of the portions of $\pi_{md}(s,t)$ in all other faces are equal to the merging angle of the corresponding faces.*

Algorithm

Step 1: We compute the planar unfolding of the faces $f_1, f_2, \ldots, f_m$ on the face f_0 that contains s. We assume that the entire terrain is in first quadrant, and all the edges in $\mathcal{T}$ are parallel to the y-axis.

Step 2: We compute the merging angles of all the faces $f_i, i = 1, 2, \ldots, m$, and store them in an array Φ in ascending order. Each element contains its face-id.

Step 3: Let θ be the slope of the line joining s and t in the unfolded plane. We sequentially inspect the element of the array Φ from its first element onwards and go on marking the faces that need to be merged. Here the optimum path passes through the marked face at equal altitude. We initialize k by 1.

Step 4: Repeat the following procedure until $\Phi[k] > \theta$

Merging phase: (* Here $\Phi[k] < \theta$, and the translation event take place as follows *)

Let $\Phi[k]$ correspond to a face f. We transform the entire terrain by merging the two boundaries of face f, i.e., compute the destination point t under the translation. The face f is marked. We update k by $k + 1$ and update θ by the joining s with the new position of t.

Step 5: The value θ, after the execution of Step 4, corresponds to the slope of the path segments which are strictly monotone descent along $\pi_{md}(s, t)$. We start from the point s at face f_0, and consider each face f_i, $i = 1, 2, \ldots, m$ in order. If face f_i is not marked, $\pi_{md}(s, t)$ moves in that face along a line segment of slope θ; otherwise, $\pi_{md}(s, t)$ moves along a line segment of slope $\Phi[i]$.

Step 6: Finally, report the optimum path $\pi_{md}(s, t)$.

6.4.2 Correctness and complexity analysis of the algorithm

Lemma 6.12 *Our algorithm correctly computes the shortest monotone descent path between two query points s and t through a sequence of m faces of a polyhedral terrain bounded by parallel edges in $O(m\log m)$ time.*

Proof : We prove the correctness of the algorithm by contradiction. The path obtained by our algorithm is $\pi(s,t)$. It passes through the faces at equal altitude for which the merging angles are $\{\Phi[1],\ldots,\Phi[k]\}(\subseteq\Phi)$, and follows strictly monotone descent (with angle $=\theta$) in the faces having merging angles $\{\Phi_{k+1},\Phi_{k+2},\ldots,\Phi_m\}$, where $\Phi_i<\theta$ for $i=1,\ldots,k$, and $\Phi_i>\theta$ for $i=k+1,\ldots,m$.

Let the optimum path $\pi'(s,t)$ passes through the faces at equal altitude for which the merging angles are $\{\Phi[1],\ldots,\Phi[k']\}(\subseteq\Phi)$, and follows strictly monotone descent (with angle $=\theta'$) in the faces having merging angles $\{\Phi_{k'+1},\Phi_{k'+2},\ldots,\Phi_m\}$, where $\Phi_i<\theta'$ for $i=1,\ldots,k'$, and $\Phi_i>\theta'$ for $i=k'+1,\ldots,m$.

If $k=k'$, then by Theorem 6.5 and 6.6 we conclude that $\theta=\theta'$, and the path obtained by our algorithm is optimum. Below we argue that k cannot be different than k'.

Let us assume that $k<k'$, or in other words, $\theta<\theta'$ (since Φ is created in ascending order of merging angles). Thus, our algorithm chooses few more faces than the optimum solution where the path goes through the same height. Now, the three cases stated below are exhaustive.

As $\theta<\theta'$, $\pi'(s,t)$ diverges upward from $\pi(s,t)$ in all the faces where both the paths follow strictly monotone descent property.

In some faces $\pi'(s,t)$ goes through equal height but $\pi(s,t)$ follows monotone descent property. There also $\pi'(s,t)$ diverges upward from $\pi(s,t)$.

In those faces where $\pi'(s,t)$ and $\pi(s,t)$ go through equal height, they remain parallel.

Since $\pi(s,t)$ has reached t, $\pi'(s,t)$ will reach somewhere above t in face f_m. The similar argument proves that $k \not\succ k'$.

Given a sequence of m faces of a polyhedral terrain bounded by parallel lines, and two query points s and t, Steps 1 and 2 of the algorithm compute the merging angles and sort them in $O(m\log m)$ time. Step 3 needs $O(1)$ time. Each iteration of Step 4 needs $O(1)$ time, and we may need $O(n)$ such iterations for reporting the shortest monotone descent path from s and t. Thus the time complexity result follows. □

Below we present an improvement of our algorithm suggested by an anonymous examiner of this thesis.

Note that, the slope of the line segments of shortest monotone descent path $\pi_{md}(s,t)$ in the faces where they are strictly monotone descent, are all equal (see Lemma 6.11. Let θ be the slope of those line segments. If we can predict the slope θ then we can detect the faces that are being merged using translation events in merging face. But, since θ is not known a priori, we choose different values of θ from the array of merging angles Φ (unsorted) in a systematic manner. Initially, we compute the planar unfolding of all the faces, and mark s and t in the unfolded plane. We copy the array Φ in a temporary array Θ; each element of Θ also points to its corresponding face. We execute the following steps to justify the correctness of our choice. A *flag* bit is used during the execution, and is initialized with "0".

Step 1: Compute θ' = median of the elements in Θ.

Step 2: Split Θ in two subsets, namely Θ_1 = the members in Θ whose values are less than θ', and Θ_2 = the members in Θ whose values are greater than θ'.

Step 3: Let θ_1= maximum element in Θ_1 and θ_2 = minimum element in Θ_2.

Step 4: Execute the translation events for the members in Θ_1 as follows:

- If *flag* = 0, then apply the translation event of all the faces in Θ_1.
- If *flag* = 1, then undo the translation event of all the faces in Θ_2. The reason is

that, the translation event applied on these faces in the previous iteration, is not required in this iteration.

• Update the location of t in the changed environment.

Step 5: Join s and t by a straight line (in the unfolded plane). Let the angle of this line is $\widehat{\theta}$. Here one of the following three situations may take place: (i) $\widehat{\theta} \in [\theta_1, \theta_2]$, (ii) $\widehat{\theta} < \theta_1$, and (iii) $\widehat{\theta} > \theta_2$.

In Case (i) $\widehat{\theta} = \theta$. In Case (ii), $\theta < \widehat{\theta}$; so, we assign $\Theta = \Theta_1$, and repeat the same process with $flag = 0$. In Case (iii), $\theta > \widehat{\theta}$; so we assign $\Theta = \Theta_2$, and repeat the same process with $flag = 1$.

Theorem 6.7 *The shortest monotone descent path between two query points s and t through a sequence of m faces of a polyhedral terrain bounded by parallel edges can be computed in $O(m)$ time.*

Proof : At each inductive step, the computation of median among the elements in Θ needs $O(|\Theta|)$ time [51]. The translation events for the faces in Θ_1 or the undo of the translation events for the faces in Θ_2 also needs $O(|\Theta|)$ time. This computes the updated position of t also. Since in each iterative step, the size of Θ is reduced by half of its previous step, the time complexity of the entire algorithm is $O(m)$. □

6.5 Conclusion

We have proposed polynomial time algorithms for finding the shortest monotone descent path from a point s to a point t in a polyhedral terrains in two special cases where (i) t is a point in convex $DFR(s)$, and (ii) the path from s to t passes through a set of faces bounded by parallel edges. The general problem is still unsolved. Even the shortest monotone descent path through a given edge sequence is difficult to compute.

Chapter 7

Conclusion

In this thesis, we studied various facility location optimization problems which have potential applications in several practical problems in wireless communication, operations research, guard placement, drainage network design, irrigation canal layout, and many such other problems. Our concentration was mainly focused on the geometric domain, and the objective was to design simple, efficient and implementable algorithms for those practical problems. In the first chapter we made a brief survey of the related problems and highlighted the challenging open problems mentioned in those papers. We could solve few of those problems. We introduced some new problems in the area of geometric facility location, and discussed their importance in practical world. Our contribution in this thesis is mentioned below.

In Chapter 2, we studied the problem of locating the position of a single base station on the boundary of a convex region in the context of wireless communication. The problem is important in the sense that sometime some region may be forbidden for installing the tower of the base-station for wireless communication. But inside this region, mobile communication is always necessary. So, our objective was to place the base station at the boundary of that region such that by assigning minimum range to this base station we can cover the entire region. We considered a simplified version where the

region under consideration is convex. Our proposed algorithm for this problem runs in $O(n)$ time. An obvious generalization is, instead of one, try to install two or more base stations having equal range. Here also, the objective is to cover the entire region with minimum range of the base stations. We could not solve this problem in general. But, if we have to place two base stations, and the center of these two base stations are restricted to appear on a specified edge of the polygonal region, the problem can be solved in $O(n)$ time. As a future work, we either have to prove the hardness result for the general problem, or to design a polynomial time algorithm.

In the next chapter, we solved the query version of the above problem. Here, the convex polygon is given; we have to preprocess it such that, we can identify the location of one base station on the query line segment that can cover the entire polygon with minimum range. The preprocessing time and space complexities of our proposed algorithm are $O(n \log n)$ and $O(n)$ respectively, and the worst case time required for the query answering is $O(\log^2 n)$. An interesting problem is to design an efficient data structure and algorithm that can reduce the query time to $O(\log n)$.

Then we considered the problem of finding the location of a guard on the boundary of a convex polygon P that maximizes the external area that can be covered under L-*visibility*, where the length L and the polygon P are supplied as an input. A restricted version of this problem was already solved in linear time by Gewali et al. [72], where the length of L is less than half of the perimeter of P. We addressed the same problem where L is greater than half of the perimeter of P, and proposed a simple algorithm that runs in linear time.

In the last two chapters, we switched our attention in designing the algorithms for the following two variations of the shortest path problems on the surface of a polyhedron.

In chapter 5, an efficient and implementable algorithm for computing the approximate shortest path between a pair of points on the surface of a weighted polyhedron was proposed. Several algorithms on this problem are already available; but their running time depend on the geometric parameter of the input polyhedron. We tried to remove

one among these geometric parameters from the time complexity. But unfortunately for an arbitrary polyhedron, the approximation factor of our proposed algorithm is dependent on that geometric parameter. But we showed that, in a restricted case, where each triangular face is non-obtuse and the perpendicular distance of each side of from its opposite vertex is less than the length of that side, our algorithm achieves a solution whose length is at most $2 \times opt + \epsilon nW$, where opt is the length of the shortest path. Thus both the approximation factor and the time complexity of our algorithm is free from the geometric parameter θ (where θ is the minimum angle incident to the vertices of all the triangular faces in P) in the restricted case as mentioned above.

In Chapter 6, we concentrated on the polyhedral terrain, and studied the problem of designing the shortest monotone descent path from a given source point s. This problem is very important in the context of irrigation and drainage layout. The general problem is very hard to solve, and was mentioned as an open problem in [26]. We solved the problem in two special cases when (i) the terrain is a convex one, and (ii) when the desired path is searched through a sequence of faces where the two mutually opposite boundaries of each face are parallel. For the variation (i), the given convex terrain can be preprocessed in $O(n^2 \log n)$ time using $O(n^2)$ space with respect to a given source point s such that for any query point t we could report the shortest distance in $O(\log n)$ time. The path itself can be reported in $O(\log n + k)$ time where k is the number of faces through which the shortest path passes. For the variation (ii), we could propose an $O(n \log n)$ time algorithm for finding the shortest path between a pair of points through a given sequence of faces which are not necessarily in convex position, but are bounded by parallel edges. The general problem is still unsolved.

Bibliography

[1] P. K. Agarwal, S. Har-Peled and M. Karia, *Computing approximate shortest paths on convex polytopes*, Algorithmica, vol. 33, pp. 227-242, 2002.

[2] P. K. Agarwal and J. Matousek, *Dynamic half-space range reporting and its applications*, Algorithmica, vol. 13, pp. 325-345, 1995.

[3] P. K. Agarwal and M. Sharir, *Efficient algorithms for geometric optimization*, ACM Computing Surveys, vol. 30, pp. 412-458, 1998.

[4] P.K. Agarwal, M. Sharir and E. Welzl, *The discrete 2-center problem*, Discrete and Computational Geometry, vol. 20, pp. 287-305, 1998.

[5] A. Aggarwal, L. J. Guibas, J. Saxe and P. Shor, *A linear time algorithm for computing the Voronoi diagram of a convex polygon*, Discrete and Computational Geometry, vol. 4, pp. 591-604, 1989.

[6] A. Aggarwal, M. M. Klawe, S. Moran, P. W. Shor, and R. Wilber, *Geometric applications of a matrixsearching algorithm*, Algorithmica, vol. 2, pp. 195-208, 1987.

[7] A. Aggarwal and D. Kravets, *A linear time algorithm for finding all furthest neighbors in a convex polygon*, Information Processing Letters, vol. 31, pp. 17-20, 1989.

[8] D. Aingworth, C. Chekuri, P. Indyk and R. Motwani, *Fast estimation of diameter and shortest paths (without matrix multiplication)*, SIAM Journal on Computing, vol. 28, pp. 1167-1181, 1999.

[9] L. Aleksandrov, M. Lanthier, A. Maheshwari and J. R. Sack, *An ϵ approximation algorithm for weighted shortest path queries on polyhedral surfaces*, Proc. of the European Workshop on Computational Geometry, 1998.

[10] L. Aleksandrov, M. Lanthier, A. Maheshwari and J. R. Sack, *An ϵ approximation algorithm for weighted shortest paths on polyhedral surfaces*, Proc. of the Scandinavian Workshop on Algorithmic Theory, LNCS 1432, pp. 11-22, 1998.

[11] L. Aleksandrov, A. Maheshwari and J. R. Sack, *Approximation algorithms for geometric shortest path problems*, Proc. of the Symposium on Theory of Computing, pp. 286-295, 2000.

[12] L. Aleksandrov, A. Maheshwari and J. R. Sack, *An improved approximation algorithms for computing geometric shortest paths* , Proc. of the Symposium on Foundations of Computing Theory, LNCS 2751, pp. 246-257, 2003.

[13] L. Aleksandrov, A. Maheshwari and J. R. Sack, *Determining approximate shortest paths on weighted polyhedral surfaces*, Journal of the ACM, vol. 52, pp. 25-53, 2005.

[14] O. Alp, Z. Drezner and E. Erkut, *An efficient genetic algorithm for the p-median problem*, Annals of Operations Research, vol. 122, pp. 21-42, 2003.

[15] I. Althofer, G. Das, D. Dobkin, D. Joseph and J. Soares, *On sparse spanners of weighted graphs*, Discrete and Computational Geometry , vol. 9, pp. 81-100, 1993.

[16] B. Aronov, A. R. Davis, T. K. Dey, S. P. Pal and D.C. Prasad, *Visibility with reflection*, Proc. of the Eleventh Annual Symposium on Computational Geometry, pp. 316-325, 1995.

[17] B. Aronov, M. V. Kreveld, R. V. Oostrum and K. Varadarajan, *Facility location on terrains*, Proc. of the International Symposium on Algorithm and Computation, LNCS 1533, pp. 19-28, 1998.

[18] S. Arora, P. Raghavan and S. Rao, *Approximation schemes for the Euclidean k-medians and related problems*, Proc. of the 30th ACM Symposium on Theory of Computing, pp. 106-113, 1998.

[19] T. Asano, *Efficient algorithms for finding the visibility polygons for a polygonal region with holes*, Transactions of IECE of Japan, E-68, pp. 557-559, 1985.

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

[21] R. Bar-Yehuda, A. Efrat and A. Itai, *A simple algorithm for maintaining the center of a planar point-set*, Proc. of the Canadian Conference on Computational Geometry, pp. 252-257, 1993.

[22] B. Ben-Moshe, *Geometric Facility Location Optimization*, PhD. thesis, Department of Computer Science, Ben-Gurion University of the Negev Beer-Sheva, Israel, 2004.

[23] B. Ben-Moshe, B. K. Bhattacharya and Q. Shi, *An optimal algorithm for the continuous/discrete weighted 2-center problem in trees*, Proc. of the 7th Latin American Theoretical Informatics Symposium, LNCS 3887, pp. 166-177, 2006.

[24] B. Ben-Moshe, Matthew J. Katz and Michael Segal, *Obnoxious facility location: complete service with minimal harm*, International Journal of Computational Geometry and Applications, vol. 10, pp. 581-592, 2000.

[25] M. de Berg, P. Bose, K. Dobrint, M. van Kreveld, M. Overmars, M. de Groot, T. Roos, J. Snoeyink, and S. Yu, *The complexity of rivers in triangulated terrains*, Proc. of the 8th Canadian Conference on Computational Geometry, pp. 325-330, 1996.

[26] M. de Berg and M. van Kreveld, *Trekking in the Alps without freezing or getting tired*, Algorithmica, vol. 18, pp. 306-323, 1997.

[27] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry - Algorithms and Applications*, Springer-Verlag, 1997.

[28] S. Bespamyatnikh and D. Kirkpatrick, *Rectilinear 2-center problems*, Proc. of the 11th Canadian Conference on Computational Geometry, pp. 68-71, 1999.

[29] S. Bespametnikh, K. Kedem, M. Segal and A. Tamir, *Optimal facility location under various distance functions*, International Journal of Computational Geometry and Applications, vol. 10, pp. 523-534, 2000.

[30] B. K. Bhattacharya, G. Das, A. Mukhopadhyay and G. Narasimhan, *Optimally computing a shortest weakly visible line segment inside a simple polygon*, Computational Geometry: Theory and Applications, vol. 23, pp. 1-29, 2002.

[31] B. K. Bhattacharya and A. Mukhopadhyay, *Computing in linear time a chord from which a simple polygon is weakly internally visible*, Proc. of the 6th International Symposium on Algorithms and Computation, pp. 22-31, 1995.

[32] B. K. Bhattacharya, A. Mukhopadhya and G. T. Toussaint, *Computing a shortest weakly externally visible line segment for a simple polygon*, International Journal of Computational Geometry and Applications, vol. 9, pp. 1-29, 1999.

[33] B. K. Bhattacharya and G. T. Toussaint, *On geometric algorithms that use the furthest-point Voronoi diagram*, in G. T. Toussaint, Editor, Computational Geometry, North-Holland, Amsterdam, Netherlands, pp. 43-61, 1985.

[34] J. -D. Boissonnat, J. Czyzowicz, O. Devillers and M. Yvinec, *Circular separability of polygon*, Proc. of the Annual ACM-SIAM symposium on Discrete algorithms, pp. 273 - 281 1995.

[35] P. Bose and G. Toussaint, *Computing the constrained Euclidean, geodesic and link center of a simple polygon with applications*, Proc. of the Pacific Graphics International, pp. 102-112, 1996.

[36] P. Bose and Q. Wang, *Facility location constrained to a polygonal domain*, Proc. of the Latin American Theoretical Informatics Symposium, LNCS 2286, pp. 153-164, 2002.

[37] J. Canny and J. Reif, *New lower bound techniques for robot motion planning problems*, Proc. of the IEEE Symposium on Foundations of Computer Science, pp. 49-60, 1987.

[38] T.M. Chan, *More planar two-center algorithms*, Computational Geometry: Theory and Applications, vol. 13, pp. 189-198, 1999.

[39] M. Charikar and S. Guha, *Improved combinatorial algorithms for the facility location and k-median problems*, SIAM Journal on Computing, vol. 34, pp. 803-824, 2005.

[40] M. Charikar, S. Guha, E. Tardos and D. B. Shmoys, *A constant-factor approximation algorithm for the k-median problem*, Journal of Computer and System Sciences, vol. 65, pp. 129-149, 2002.

[41] B. Chazelle, *A theorem on polygon cutting with applications*, Proc. of the IEEE Symposium on Foundations of Computer Science, pp. 339-349, 1982.

[42] B. Chazelle, *Efficient polygon triangulation*, Proc. of the IEEE Symposium on Foundations of Computer Science, pp. 220-230, 1990.

[43] B. Chazelle, *Triangulating a simple polygon in linear time*, Discrete Computational Geometry, vol. 6, pp. 485-524, 1991.

[44] B. Chazelle, *Application challenges to computational geometry: CG impact task force report*, Technical Report TR-521-96, Princeton University, 1996.

[45] B. Chazelle, D. Liu and A. Magen, *Sublinear geometric algorithms*, SIAM Journal on Computing, vol. 35, pp. 627-646, 2005.

[46] J. Chen and Y. Han, *Shortest paths on a polyhedron*, International Journal on Computational Geometry and Applications, vol. 6, pp. 127-144, 1996.

[47] Y. J. Chiang, F.P. Preparata and R. Tamassia, *A unified approach to dynamic point location, ray shooting, and shortest paths in planar maps*, SIAM Journal on Computing, vol. 25, pp. 207-233, 1996.

[48] V. Chvatal, *A combinatorial theorem in plane geometry*, Journal of Combinatorial Theory Ser. B, vol. 18, pp. 39-41, 1975.

[49] D. Coppersmith, *Rectangular matrix multiplication revisited*, Journal of Complexity, vol. 13, pp. 42-49, 1997.

[50] E. Cohen and U. Zwick, *All-pairs small-stretch paths*, Journal of Algorithms, vol. 38, pp. 335-353, 2001.

[51] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, MIT Press, 1990.

[52] O. Daescu, N. Mi, C. Shin and A. Wolff, *Furthest-point queries with geometric and combinatorial constraints*, Computational Geometry: Theory and Applications, vol. 33, pp. 174-185, 2006.

[53] G. Das, P. Heffernan and G. Narasimhan, *Finding all weakly- visible chords of a polygon in linear time*, Nordic Journal of Computing, vol. 1, pp. 433-457, 1994.

[54] E. W. Dijkstra, *A note on two problems in connection with graphs*, Numerical Mathematics, vol. 1, pp. 267-271, 1959.

[55] D. Dor, S. Halperin and U. Zwick, *All pairs almost shortest paths*, SIAM Journal on Computing, vol. 29, pp. 1740-1759, 2000.

[56] Z. Drezner, *On a modified one-center model*, Management Science, vol. 27, pp. 848-851, 1983.

[57] Z. Drezner, *Conditional p-center problems*, Transportation Science, vol. 23, pp. 51-53, 1989.

[58] Z. Drezner, *Facility Location: A Survey of Applications and Methods*, Springer Verlag, 1995.

[59] Z. Drezner, A. Mehrez and G. O. Wesolowsky, *The facility location problem with limited distances*, Transportation Science, vol. 25, 183-187, 1991.

[60] H. ElGindy and D. Avis, *A linear time algorithm for computing the visibility polygon from a point*, Journal of Algorithms, vol. 2, pp. 186-197, 1981.

[61] H. ElGindy and M. T. Goodrich, *Parallel algorithms for shortest path problems in polygons*, Algorithmica, vol.8, pp. 461-486, 1992.

[62] M. L. Elkin, *Computing almost shortest paths*, Technical Report MCS01-03, Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel, 2001.

[63] J. Elzinga and D. W. Hearn, *Geometrical solutions to some minimax location problems*, Transportation Science, vol. 6, pp. 379-394, 1972.

[64] D. Eppstein, *Faster construction of planar two-centers*, Proc. of the 8th ACM-SIAM Symposium on Discrete Algorithms, pp. 131-138, 1997.

[65] S. P. Fekete, J. S. B. Mitchell and K. Weinbrecht, *On the continuous Weber and k-median problems*, Proc. of the 16th Annual ACM Symposium on Computational Geometry, pp. 70-79, 2000.

[66] M. L. Fredman, *New bounds on the complexity of the shortest path problem*, SIAM Journal on Computing, vol. 5, pp. 49-60, 1976.

[67] M. L. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, Journal of the ACM, vol. 34, pp. 596-615, 1987.

[68] M. L. Fredman and D. E. Willard, *Surpassing the information- theoretic bound with the fusion trees*, Journal of Computer and System Sciences, vol. 47, pp. 424-436, 1993.

[69] M. L. Fredman and D. E. Willard, *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, Journal of Computer and System Sciences, vol. 48, pp. 533-551, 1994.

[70] H. N. Gabow, *Scaling algorithms for network problems*, Journal of Computer and System Sciences, vol. 31, pp. 148-168, 1985.

[71] H. N. Gabow and R. E. Tarjan, *Scaling algorithms for network problems*, SIAM Journal on Computing, vol. 18, pp. 1013-1036, 1989.

[72] L. Gewali, R. Venkatasubramanian and D. Glasser, *Algorithms for computing grazing area*, manuscript, http://citeseer.ist.psu.edu/485516.html.

[73] S. K. Ghosh, *Visibility Algorithms in the Plane*, Cambridge University Press, 2007.

[74] S. K. Ghosh, A. Maheshwari, S. P. Pal, S. Saluja and C. E. Veni Madhaban, *Computing the shortest path tree in a weak visibility polygon*, Proc. of the 11th Symposium on Foundation of Software Technology and Theoretical Computer Science, LNCS 560, pp. 369-389, 1991.

[75] S. K. Ghosh and D. M. Mount, *An output-sensitive algorithm for computing visibility graphs*, SIAM Journal on Computing, vol. 20, pp. 88-910, 1991.

[76] A. Glozman, K. Kedem and G. Shpitalnik, *On some geometric selection and optimization problems via sorted matrices*, Computational Geometry, vol. 11, pp. 17-28, 1998.

[77] A. V. Goldberg, *Scaling algorithms for the shortest paths problem*, SIAM Journal on Computing, vol. 24, pp. 494-504, 1995.

[78] A. Goldwasser, *A survey of linear programming in randomized subexponential time*, ACM-SIGACT News, vol. 26, pp. 96-104, 1995,

[79] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, NY, 1980.

[80] M. T. Goodrich, S. Shauck and S. Guha, *Parallel methods for visibility and shortest path problems in simple polygons*, Algorithmica, vol. 8, pp. 461-486, 1992.

[81] M. T. Goodrich, S. Shauck and S. Guha, *An addendum to parallel methods for visibility and shortest path problems in simple polygons*, Algorithmica, vol. 9, pp. 515-516, 1993.

[82] M. T. Goodrich and R. Tamassia, *Dynamic ray shooting and shortest paths via balanced geodesic triangulations*, Journal of Algorithms, vol. 23, pp. 51-73, 1997.

[83] C. Gray and W. Evans, *Optimistic shortest paths on uncertain terrains*, Proc. of the 16th Canadian Conference on Computational Geometry, pp. 68-71, 2004.

[84] L. J. Guibas and J. Hershberger, *Optimal shortest path queries in a simple polygon*, Journal of Computer and System Sciences, vol. 39, pp. 126-152, 1989.

[85] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir and R. E. Tarjan, *Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons*, Algorithmica, vol. 2, pp. 209-233, 1987.

[86] U. I. Gupta, D. T. Lee and J. Y. T. Leung, *Efficient algorithms for interval graphs and circular arc graphs*, Networks, vol. 12, pp. 459-467, 1982.

[87] T. Hagerup, *Improved shortest paths on the word RAM*, Proc. of the 27th International Colloquium on Automata, Languages and Programming, LNCS 1853, pp. 61-73, 2000.

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

[89] H. W. Hamacher and S. Nickel, *Classification of location models*, Location Science, vol. 6, pp. 229-242, 1998.

[90] J. Hershberger, *Minimizing the sum of diameters efficiently*, Computational Geometry: Theory and Applications, vol. 2, pp. 111-118, 1992.

[91] J. Hershberger, *A faster algorithm for the two-center decision problem*, Information Processing Letters, vol. 47, pp. 23-29, 1993.

[92] J. Hershberger, *Optimal parallel algorithms for triangulated simple polygon*, International Journal of Computational Geometry and Applications, vol. 5, pp. 145-170, 1995.

[93] D. S. Hochbaum and D. Shmoys, *A best possible heuristic for the k- center problem*, Mathematics of Operations Research, vol. 10, pp. 180-184, 1985.

[94] F. Hurtado, V. Sacristan and G. Toussaint, *Facility location problems with constraints*, Studies in Locational Analysis, vol. 15, pp. 17-35, 2000.

[95] R. Z. Hwang, R. C. Chang and R. C. T. Lee, *The slab dividing approach to solve the Euclidean p-center problem*, Algorithmica, vol. 9, pp. 1-22, 1993.

[96] R. Z. Hwang, R. C. Chang and R. C. T. Lee, *The searching over separators strategy To solve some NP-hard problems in subexponential time*, Algorithmica, vol. 9, pp. 398-423, 1993.

[97] C. Icking and R. Klein, *The two guards problem*, International Journal of Computational Geometry and Applications, vol. 2, pp. 275-285, 1992.

[98] J. W. Jaromczyk and M. Kowaluk, *An efficient algorithm for the Euclidean two-center problem*, Proc. of the 10th Annual ACM Symposium on Computational Geometry, pp. 303-311, 1994.

[99] D. B. Johnson, *Efficient algorithms for shortest paths in sparse graphs*, Journal of the ACM, vol. 24, pp. 1-14, 1977.

[100] S. Kapoor, *Efficient computation of geodesic shortest paths*, Proc. of the Symposium on Theory of Computing, pp. 770-779, 1999.

[101] M. J. Katz, K. Kedem and M. Segal, *Discrete rectilinear 2-center problems*, Computational Geometry : Theory and Applications, vol. 15, pp. 203-214, 2000.

[102] M. J. Katz, K. Kedem and M. Segal, *Improved algorithms for placing undesirable facilities*, Computers and Operations Research, vol. 29, pp. 1859-1872, 2002.

[103] S. K. Kim and C. S. Shin, *Efficient algorithms for two-center problems for a convex polygon*, Proc. of the 6th International Conference on Computing and Combinatorics, LNCS 1858, pp. 299-309, 2000.

[104] M. Korupolu, C. Plaxton and R. Rajaraman, *Analysis of a local search heuristic for facility location problems*, Journal of Algorithms, vol. 37, pp. 146-188, 2000.

[105] M. van Kreveld, *On quality paths in polyhedral terrains*, Proc. of the International Workshop on Advanced Research in Geographic Information Systems, LNCS 884, pp. 113-122, 1994.

[106] D. G. Kirkpatrick, *Optimal search in planar subdivisions*, SIAM Journal on Computing, vol. 12, pp. 28-35, 1983.

[107] M. Lanthier, *Shortest Path Problems on Polyhedral Surfaces*, PhD. thesis, Ottawa-Carleton Institute for Computer Science, Carleton University, Canada, 1999.

[108] M. Lanthier, A. Maheswari and J. R. Sack, *Shortest anisotropic paths in terrains*, Proc. of the International Colloquium on Automata, Languages and Programming, pp. 524-533, 1999.

[109] M. Lanthier, A. Maheswari and J. R. Sack, *Approximating weighted shortest paths on polyhedral surfaces*, Algorithmica, vol. 30, pp. 527-562, 2001.

[110] D. T. Lee, *Furthest neighbor Voronoi diagrams and applications*, Report 80-11-FC-04, Dept. Elect. Engrg. Comput. Sci., Northwestern Univ., Evanston, IL, 1980.

[111] D. T. Lee and F. P. Preparata, *Euclidean shortest paths in the presence of rectilinear barriers*, Networks, vol. 14, pp. 393-410, 1984.

[112] S. Lin and B. W. Kernighan, *An effective heuristic algorithm for the traveling salesman problem*, Operations Research, vol. 21, pp. 498-516, 1973.

[113] J. Lin and J. S. Vitter, *Approximation algorithms for geometric median problems*, Information Processing Letters, vol. 44, pp. 245-249, 1992.

[114] A. Marchetti-Spaccamela, *The p-center problem in the plane is NP-complete*, Proc. of the 19th Allerton Conference on Communication, Control and Computing, pp. 31-40, 1981.

[115] C. Mata and J. S. B. Mitchell, *A new algorithm for computing shortest paths in weighted planar subdivisions*, Proc. of the 13th ACM Symposium on Computation Geometry, pp. 264-273, 1997.

[116] N. Megiddo, *Linear-time algorithms for linear programming in R^3 and related problems*, SIAM Journal on Computing, vol. 12, pp. 759-776, 1983.

[117] N. Megiddo, *The weighted Euclidean 1-center problem*, Mathematics of Operation Research, vol. 8, pp. 498-504, 1983.

[118] N. Megiddo, *On the complexity of some geometric problems in unbounded dimension*, Journal of Symbolic Computation, vol. 10, pp. 327-334, 1990.

[119] K. Mehlhorn and Z. Galil, *Monotone switching circuits and Boolean matrix product*, Computing, vol. 16, pp. 99-111, 1976.

[120] P. B. Mirchandani and R. L. Francis, *Discrete Location Theory*, John Wiley and Sons, 1998.

[121] J. S. B. Mitchell, D. M. Mount and C. H. Papadimitrou, *The discrete geodesic problem*, SIAM Journal on Computing, vol. 16, pp. 647-668, 1987.

[122] J. S. B. Mitchell and C. H. Papadimitrou, *The weighted region problem: finding shortest paths through a weighted planar subdivision*, Journal of the Association for Computing Machinery, vol. 38, pp. 18-73, 1991.

[123] J. S. B. Mitchell and M. Sharir, *New results on shortest paths in three dimensions*, Proc. of the 20th symposium on Computational geometry, pp. 124-133, 2004.

[124] S. Ntafos, *Watchman routes under limited visibility*, Computational Geometry: Theory and Applications, vol. 1, pp. 149-170, 1992.

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

[126] C. H. Papadimitrou, *An algorithm for shortest path motion in three dimension*, Information Processing Letters, vol. 20, pp. 259-263, 1985.

[127] M. S. Patreson, *Complexity of monotone networks for boolean matrix product*, Theoretical Computer Science, vol. 1, pp. 13-20, 1975.

[128] J. Plesnik, *A heuristic for the p-center problem in graphs*, Discrete Applied Mathematics, vol. 17, pp. 263-268, 1987.

[129] F. Preparata, *Minimum spanning circle*, Technical report, Univ. Illinois, Urbana, IL, in Steps into Computational Geometry, 1977.

[130] F. P. Preparata, *A new approach to planar point location*, SIAM Journal of Computing, vol. 10, pp. 473-482, 1981.

[131] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction, second edition*, Springer-Verlag, 1990.

[132] J. H. Reif and Z. Sun, *An efficient approximation algorithm for weighted region shortest path problem*, Proc. of the Workshop on Algorithmic Foundations of Robotics, pp. 191-203, 2000.

[133] J. H. Reif and Z. Sun, *Adaptive and compact discretization for weighted region optimal path finding*, Proc. of the Foundations of Computing Theory, pp. 258-270, 2003.

[134] N.C. Rowe and R.S. Ross, *Optimal grid-free path planning across arbitrary contoured terrain with anisotropic friction and gravity effects*, IEEE Transactions on Robotics and Automation, vol. 6, pp. 540-553, 1990.

[135] J. R. Sack and J. Urrutia, *Handbook of Computational Geometry*, Elsevier Science, 2000.

[136] Y. Schreiber and M. Sharir, *An optimal-time algorithm for shortest paths on a convex polytope in three dimensions*, Proc. of the 22nd Annual ACM Symposium on Computational Geometry, pp. 30-39, 2006.

[137] R. Seidel, *On the all-pair-shortest-path problem in unweighted undirected graphs*, Journal of Computer and System Sciences, vol. 51, pp. 400-403, 1995.

[138] M. I. Shamos, *Computational Geometry*, PhD. thesis, Department of Computer Science, Yale University, New Haven, CT , 1978.

[139] M. I. Shamos and D. Hoey, *Closest-point problem*, Proc. of the 16th Annual IEEE Symposium on Foundations Computer Science, pp. 151-162, 1975.

[140] M. Sharir, *A near-linear algorithm for the planar 2-center problem*, Discrete Computational Geometry, vol. 18, pp. 125-134, 1997.

[141] M. Sharir and A. Schorr, *On shortest paths in polyhedral space*, SIAM Journal on Computing, vol. 15, pp. 193-215, 1986.

[142] T. Shermer, *Recent results in art galleries*, Proceedings of the IEEE, vol. 80, pp. 1384-1399, 1992.

[143] C. Shin, J. Kim, S. K. Kim and K. Chwa, *Two-center problems for a convex polygon*, Proc. of the 6th Annual European Symposium on Algorithms, pp. 199-210, 1998.

[144] A. Shoshan and U. Zwick, *All pair shortest paths in undirected graphs with integer weights*, Proc. of the 40th IEEE Symposium on Foundations of Computer Science, pp. 605-614, 1995.

[145] V. Strassen, *Gaussian elimination is not optimal*, Numerische Mathematik, vol. 13, pp. 354-356, 1969.

[146] C. Swamy and D. B. Shmoys, *Fault-tolerant facility location*, Proc. of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 725-736, 2003.

[147] J. J. Sylvester, *A question in the geometry of situation*, Quarterly Journal of Mathematics, pp. 1-79, 1857.

[148] T. Takaoka, *A new upper bound on the complexity of the all pairs shortest path problem*, Information Processing Letters, vol. 43, pp. 195-199, 1992.

[149] T. Takaoka, *Subcubic cost algorithms for the all pairs shortest path problem*, Algorithmica, vol. 20, pp. 309-318, 1998.

[150] M. Thorup, *Undirected single-source shortest paths with integer weight in linear time*, Journal of the ACM, vol. 46, pp. 362-394, 1999.

[151] M. Thorup, *Floats, integers, and single source shortest paths*, Journal of Algorithms, vol. 35, pp. 189-201, 2000.

[152] M. Thorup, *On RAM priority queues*, SIAM Journal on Computing, vol. 30, pp. 86-109, 2000.

[153] G. T. Toussaint and D. Avis, *On a convex hull algorithm for polygons and its application to triangulation problem*, Pattern Recognition, vol. 15, pp. 23-29, 1982.

[154] K. R. Varadarajan and P. K. Agarwal, *Approximating shortest paths on a non-convex polyhedron*, SIAM Journal on Computing, vol. 30, pp. 1321-1340, 2000.

[155] A. Weber, *er den Standort der Industrien*, Tuebingen, 1909.

[156] G. Wesolowsky, *The Weber problem: history and perspectives*, Location Science, vol. 1, pp. 5-23, 1993.

[157] M. Ziegelmann, *Constrained shortest paths and related problems*, PhD. thesis, Universitat des Saarlandes, (Max-Plank Institut fur Informatik), 2001.

[158] U. Zwick, *All pairs shortest paths in weighted undirected graphs- exact and almost exact algorithms*, Proc. of the 39th IEEE Symposium on Foundations of Computer Science, pp. 310-319, 1998.

[159] U. Zwick, *All pairs lightest shortest paths*, Proc. of the 31th Annual ACM Symposium on Theory of Computing, pp. 61-69, 1999.

[160] U. Zwick, *Exact and approximate distances in graphs - a survey*, Proc. of the 9th European Symposium on Algorithms, pp. 33-48, 2001. 1999.

Publications by the author on some of which the content of the thesis is based

[T1] S. Roy, S. Das and S. C. Nandy, *Shortest monotone descent path problem in polyhedral terrain*, Computational Geometry : Theory and Applications, vol. 37, pp. 115-133, 2007.

[T2] S. Roy, A. Karmakar, S. Das and S. C. Nandy, *Constrained minimum enclosing circle with center on a query line segment*, Proc. of the International Symposium on Mathematical Foundations of Computer Science, LNCS 4162, pp. 765-776, 2006.

[T3] D. Bardhan, S. Roy and S. Das, *Optimal guard placement problem under L-visibility*, Proc. of the International Conference Computational Science and Its Applications, LNCS 3980, pp. 10-19, 2006.

[T4] S. Roy, D. Bardhan and S. Das , *Efficient algorithm for placing base stations by avoiding forbidden zone*, Proc. of the 2nd International Conference on Distributed Computing and Internet Technology, LNCS 3816, pp. 105- 116, 2005.

[T5] S. Roy, S. Bhattacharjee, S. Das and S. C. Nandy, *A Fast algorithm for point labeling problem*, Proc. of the 17th Canadian Conference on Computational Geometry, pp. 152-155, 2005.

[T6] S. Roy, S. Das and S. C. Nandy, *Shortest monotone descent path problem in polyhedral terrain*, Proc. of the 22nd Annual Symposium on Theoretical Aspects of Computer Science, LNCS 3404, pp. 281-292, 2005.

[T7] S. Roy, S. Das and S. C. Nandy, *A practical algorithm for approximating shortest weighted path between a pair of points on polyhedral surface*, Proc. of International Conference Computational Science and Its Applications, LNCS 3045, pp. 42-52, 2004.

[T8] S. Roy, P. P. Goswami, S. Das and S. C. Nandy, *Optimal algorithm for a special point-labeling problem*, Information Processing Letters, vol. 89, pp. 91-98, 2004.

[T9] S. Roy, P. P. Goswami, S . Das and S. C. Nandy, *Optimal algorithm for a special point-labeling problem*, Proc. of the 8th Scandinavian Workshop on Algorithm Theory, LNCS 2368, pp. 110-120, 2002.

[T10] D. Bardhan, S. Roy and S. Das, *Optimal guard placement problem under L-visibility*, invited in International Journal of Computational Geometry and Applications, Special issue of selected papers from the 4th International Conference on Computational Science and Its Applications.

[T11] S. Roy, A. Karmakar, S. Das and S. C. Nandy, *Constrained minimum enclosing circle with center on a query line segment*, communicated to Algorithmica.

[T12] S. Roy, D. Bardhan and S. Das, *Base station placement on boundary of a convex polygon*, communicated to Journal of Parallel and Distributed Computing.