RECTILINEAR SHORTEST PATHS AMONG OBSTACLES IN THE PLANE

by

Pinaki Mitra

B.E, Jadavpur University, Calcutta, 1987

M.E, Indian Institute of Science, Bangalore, 1989

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

© Pinaki Mitra 1995
SIMON FRASER UNIVERSITY
March 1995

All rights reserved. This work may not be
reproduced in whole or in part, by photocopy
or other means, without the permission of the author.

APPROVAL

Name: Pinaki Mitra

Degree: Doctor of Philosophy

Title of thesis: Rectilinear Shortest Paths Among Obstacles in the Plane

Examining Committee: Prof. David Fracchia
Chair

Prof. Binay K. Bhattacharya
Senior Supervisor

Prof. Arthur Liestman
Supervisor

Prof. Tom Shermer
Supervisor

Prof. Pavol Hell
SFU External Examiner

Prof. Joseph S. B. Mitchell
External Examiner

Date Approved: October 3, 1994

SIMON FRASER UNIVERSITY

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

Rectilinear Shortest Paths Among Obstacles in the Plane.

Author: ______________________________

(signature)

Pinaki Mitra

(name)

March 9, 1995

(date)

Abstract

The shortest-path problem has been studied in various settings in Computational Geometry literatures. This includes the shortest-path problem inside the simple polygon and the shortest-path problem avoiding a set of polygonal obstacles. Further variations of these problems are possible for different types of polygons or polygonal obstacles.

In this thesis we will study the rectilinear shortest path problem avoiding a set of isothetic rectangles and vertical line segment obstacles. We will present efficient preprocessing algorithms to answer the shortest path between two arbitrary query points. We also demonstrate an approximation algorithm with less preprocessing and query time to report an approximate shortest path between two arbitrary query points. Then we present an efficient parallel algorithm to preprocess the set of rectangles to answer the shortest-path query between two arbitrary points using a single processor. We also present an efficient parallel algorithm to answer the single shot shortest-distance between a source and a destination point specified during the input. Lastly we present an efficient parallel algorithm to preprocess a set of vertical line segments to answer the approximate shortest-path query between two arbitrary points.

Acknowledgements

I would like to thank Prof. Binay K. Bhattacharya for his help and encouragement to carry out the research. I would like to thank Prof. Hossam ElGindy for introducing me to this area. A part of this thesis is a joint work carried out with him in [17]. I would like to thank Prof. Godfried Toussaint from whom I learned a lot of issues on Computational Geometry. I am grateful to Prof. Tom Shermer for providing several suggestions which has improved the presentation as well as some results in the thesis. I would like to thank Prof. Pavol Hell and Prof. Arthur Liestman for their comments on the thesis. Help provided by Prof. Ramesh Krishnamurti is also acknowledged. I would also like to acknowledge Prof. Lou Hafer with whom I had carried out some research though they are not related to my thesis.

Last but not the least I would like to thank Roman Bacik, Dayaram Gaur and Graham Finlayson with whom I had spent several hours discussing on various technical materials.

Dedicated to :-

My grandmother Smt. Tushar Kana Basu Mallick

&

My mother Smt. Bani Mitra

Contents

Abstract iv

Acknowledgements v

1 Introduction 1

1.1 Definitions and Notations 3

2 Review 6

2.1 Sequential Shortest-Path Algorithms 7

2.2 Approximations in Shortest-Path Problems 9

2.3 Parallel Shortest-Path Algorithms 9

2.4 Other Areas of Shortest-Path Problems 10

3 Shortest-Path Query 13

3.1 Preliminaries 13

3.2 Carrier Graphs 14

3.3 Shortest Path Queries Between Two Corner Points 26

3.3.1 Sequential Preprocessing and Query Algorithms 26

3.3.2 Constructing a Shortest Path 31

3.4 Removal of Corner Point Restriction 33

4 Approximate Shortest-Path Algorithms 37

4.1 A Brief Review of Some Existing Results 38

4.2 Query Between Two Arbitrary Points 40

4.2.1 Query Between Two Arbitrary Corner Points 40
4.2.2 Removal of the Corner Point Restriction 52
4.3 Vertical Line Segment Obstacles 58

5 Parallel Shortest Path Algorithms **65**
5.1 Parallel Preprocessing and Sequential Query algorithms 67
5.1.1 Removal of Corner Point Restriction 69
5.2 An Efficient Parallel Single-Source Shortest-Distance Algorithm . . . 70
5.2.1 Constructing the Graph 71
5.2.2 Searching the Graph 75
5.3 Parallel Preprocessing Algorithm for Vertical Line Segments 75

6 Conclusion and Open Problems **78**

List of Figures

1.1 The Eight Staircase Chains and Their Plane Partition 5

3.1 Construction of the Carrier Graph G_{+x} : Step I - Right Shadows . . . 16
3.2 Construction of the Carrier Graph G_{+x} : Step II - Upper and Lower Shadows 17
3.3 Construction of the Carrier Graph G_{+x} : Step III - Orientation 18
3.4 An undirected carrier graph UG_{+x} 19
3.5 A directed carrier graph G_{+x} 20
3.6 An Illustration for Lemma 3.4 22
3.7 Illustration for Lemma 3.6 24
3.8 Illustration for Lemma 3.9 30
3.9 Corner Points in $\mathcal{R}$ Obtained by Horizontal Ray Shooting from s & t 34
3.10 Corner Points in $\mathcal{R}$ Obtained by Vertical Ray Shooting from s & t . . 35

4.1 Sparse Visibility Graph Construction-I 41
4.2 Sparse Visibility Graph Construction-II 42
4.3 Case I - t_i Is Between t_k and t_l 45
4.4 Case II - t_i is above t_k 47
4.5 A Case in which the Constant of the Approximation For Rectangular Obstacles is Tight 48
4.6 Illustration of a st-graph and its $<_L$ and $<_R$ order 53
4.7 Illustrating the Ray-Shooting Procedure 56
4.8 A Case in which the Constant of the Approximation For Vertical Line Segments is Tight 62

5.1 Construction of the Graph G : Step I - Right Shadows in R_{+x} 72
5.2 Construction of the Graph G : Step III - Upper and Lower Shadows in R_{+x} 73

Chapter 1

Introduction

We study the rectilinear shortest-path problem in the presence of $\mathcal{R}$, a set of n isothetic disjoint rectangles and $\mathcal{B}$, a set of n vertical line segments. This problem has applications in motion planning problems. Besides that it also has applications in VLSI layout design where routing is done only along some fixed orientations, usually along horizontal and vertical directions. Our goal here is the following :

- *The design of an efficient algorithm to preprocess the set $\mathcal{R}$ to answer a shortest-distance/shortest-path query between two arbitrary points efficiently. More specifically we want to spend subquadratic preprocessing time to answer the shortest-distance/shortest-path query in sublinear time.*

- *The design of an efficient approximation algorithm to preprocess the set $\mathcal{R}/\mathcal{B}$ to answer an approximate shortest-distance/shortest-path query between two arbitrary points efficiently. More specifically we want to spend subquadratic preprocessing time to answer the query in polylogarithmic time.*

- *The design of an efficient parallel algorithm to preprocess $\mathcal{R}$ to answer a shortest-distance/shortest-path query between two arbitrary query points using a single processor.*

- *The design of an efficient parallel algorithm to answer a single-shot shortest-distance/shortest-path problem between the source and the destination in the*

presence of $\mathcal{R}$. More formally our aim here is to design an **NC** *algorithm with subquadratic processor-time-product.*

- *The design of an* **NC** *algorithm with subquadratic processor-time-product to preprocess the set $\mathcal{B}$ to answer an approximate shortest-distance/shortest-path query between two arbitrary query points using a single processor.*

A naive approach to these problems would be to build up a grid graph by taking the union of horizontal and vertical trapezoidations and placing a vertex at every intersection between horizontal and vertical edges of these two trapezoidations. This graph maintains the shortest-path information between any pair of corner points. But this graph has $O(n^2)$ vertices as well as $O(n^2)$ edges in the worst case and the complexity of all our subsequent computations would be affected by the size of this graph. So our target is to build up a sparse graph that maintains all the geometric shortest-path information.

In this thesis our goal is to design efficient algorithms. The overall organization of the thesis is as follows.

In **Chapter 2** we first make a brief survey of the existing literature of the shortest-path algorithms. Then we survey some results on spanners related to approximate shortest-path problems. Lastly we survey some results on the parallel shortest-path algorithms and some other variations of the shortest-path problem.

In **Chapter 3** we solve the shortest-path query between two arbitrary corner points of $\mathcal{R}$. For this problem we can use the **Sparse Visibility Graph** of Clarkson et al [8]. Given $\mathcal{R}$ we can maintain the shortest-path information between any two corner points in this graph. But this graph has $O(n \log n)$ edges and $O(n \log n)$ vertices. So this would also affect our subsequent shortest-path computations.

Our idea is to construct three directed acyclic planar graphs called carrier graphs. These three graphs contain sufficient information to support the shortest-path queries. We efficiently preprocess all three carrier graphs to answer the shortest-distance as well as shortest-path queries between two arbitrary points. The technique used is completely graph-theoretic for searching the shortest path and utilizes the algorithm of planar separators due to Lipton and Tarjan [34] in each of those carrier graphs.

In **Chapter 4** first we present an approximation algorithm for the shortest-path query problem. The goal here is to exhibit a tradeoff with the results obtained in **Chapter 3**. In this chapter we reduce both the preprocessing and the query time at the expense of the optimality of the reported path. With this approach we are able to report a path whose length is at most three times the optimal distance when the two query points are corner points of $\mathcal{R}$. If the two query points are arbitrary, a modified approximation algorithm returns a path whose length is at most three times the optimal distance. The approach here uses the staircase separator of Atallah and Chen [2] and the Voronoi diagram computation on the sparse visibility graph of Clarkson et al [8]. Lastly we show how to improve the preprocessing time and space for the case of vertical line segment obstacles.

In **Chapter 5** we present a parallel preprocessing algorithm on the CREW PRAM model to answer the shortest-path query between two arbitrary points using a single processor. Also in that chapter we present an **NC** algorithm for the single-source shortest-path problem with subquadratic processor-time product. Lastly we present an efficient parallel preprocessing algorithm to answer an approximate shortest-path query between two arbitrary points in the presence of vertical line segment obstacles.

Lastly, in **Chapter 6** we summarize our contributions and pose some problems for future research.

1.1 Definitions and Notations

Definition 1.1 *Given a set $\mathcal{R}$ of disjoint orthogonal rectangles and a point s, the* upper shadow *of s is defined as the point where a vertically upward ray emanating from s first intersects a rectangle in $\mathcal{R}$. Lower, right and left shadows are defined in a similar fashion.*

Definition 1.2 *Let $\mathcal{C}$ denote the set of all corner points of $\mathcal{R}/\mathcal{B}$.*

Definition 1.3 *Given a set $\mathcal{R}$ of disjoint orthogonal rectangles and a point s, the* descending $+x$ preferred orthogonal *chain, denoted by C_1, is the path from s to* infinity *constructed by starting at s and moving* right *until the right shadow of s is reached, and then moving* downward *along the rectangle boundary until we can resume the motion to the right. The remaining seven different ways for constructing chains are defined in a similar fashion, and are assigned increasing labels in the counterclockwise order around s. An illustration of the eight chains is shown in figure 1.1.*

The eight chains partition the plane into regions R_i, where $i = 1..8$, such that R_i is bounded by the chains C_i and C_{i+1}. All indices are modulo eight.

Definition 1.4 *For any point p, p_x and p_y denote its x and y coordinates, respectively.*

Definition 1.5 *Let $L_1(p,q)$ denote L_1 or the rectilinear distance between p and q. In other words $L_1(p,q) = |p_x - q_x| + |p_y - q_y|$.*

Definition 1.6 *Let $L_2(p,q)$ denote L_2 or the Euclidean distance between p and q. In other words $L_2(p,q) = \sqrt{(p_x - p_y)^2 + (q_x - q_y)^2}$.*

Definition 1.7 *A polygon $\mathcal{P}$ is rectilinear and rectilinearly convex if all its edges are either horizontal or vertical and the intersection of a vertical or horizontal line with $\mathcal{P}$ is a single line segment.*

Definition 1.8 *Let $\mathcal{Z}$ denote a rectilinear and rectilinearly convex polygon around the set of obstacles $\mathcal{R}/\mathcal{B}$.*

Definition 1.9 *$sd(s,t)$ will denote the shortest distance between s and t in presence of $\mathcal{R}$. Similarly $sp(s,t)$ will denote the shortest path between s and t in presence of $\mathcal{R}$.*

Definition 1.10 *$asd(s,t)$ will denote the approximate shortest distance between s and t in presence of $\mathcal{R}$.*

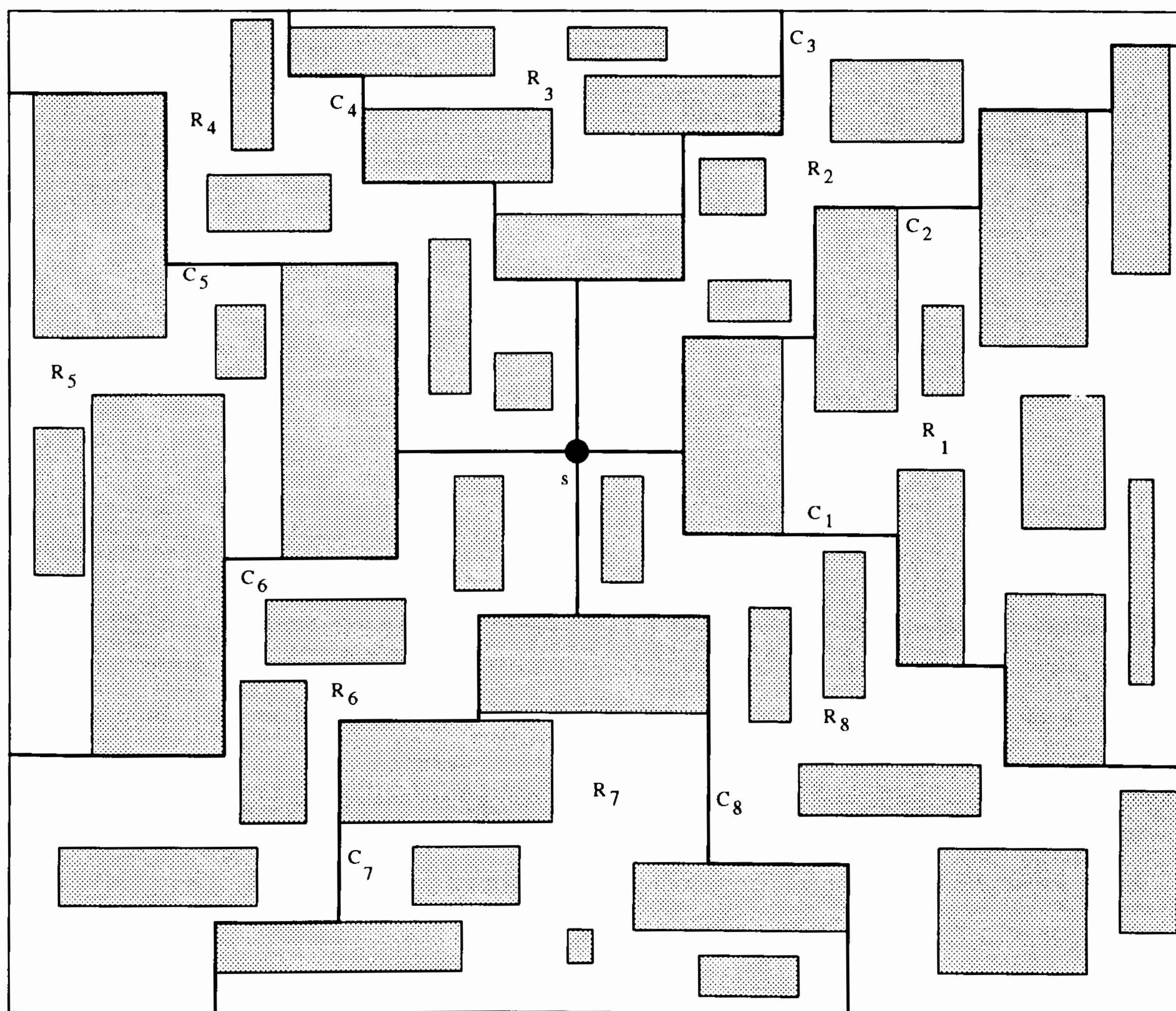


Figure 1.1: The Eight Staircase Chains and Their Plane Partition

Chapter 2

Review

Shortest-path problems received considerable attention in Computational Geometry literature. These problems are defined in the following way. We have to start from a source and move to a destination through a path of the shortest length in a specified metric. We can broadly classify the shortest-path problems in the following two categories :

- The first category of problems concerns the shortest-path problem inside simple polygons with no holes. Here the source, the destination and the path are constrained to lie inside the simple polygon with no holes. The problem here is to determine the shortest path that satisfies the preceding constraint and connects the source and the destination.
- The second category of problems concerns the shortest path between the source and the destination inside a simple polygon with holes. The part of the plane devoid of those holes is usually referred to as free-space. In motion planning problems the shortest-path computation in the free-space is required.

In this thesis we will concentrate more on the second category of problems. The overall organization of this chapter is as follows. In **section 2.1** we will survey sequential shortest-path algorithms. In **section 2.2** parallel shortest-path algorithms will be reviewed. In **section 2.3** we will briefly describe other areas in the field of shortest-path motion planning.

2.1 Sequential Shortest-Path Algorithms

The shortest-path problem inside a simple polygon was first studied by Lee and Preparata. [32]. A linear-time algorithm to compute the shortest-path tree inside a triangulated simple polygon from a vertex specified in the input was proposed by Guibas et al [25]. In a subsequent result Guibas and Hershberger [24] relaxed the restriction of the source being specified in the input. They proposed a solution that reports the shortest distance between two arbitrary query points in $O(\log n)$ time using $O(n)$ preprocessing where n is the number of vertices of the simple polygon.

Similar types of works have been done for the shortest-path problem in the free-space. In the Euclidean metric the computation of the shortest path uses the visibility graph of the free-space. Given a set of polygonal obstacles, where the total number of polygonal vertices is n, the visibility graph of the free-space can be computed by an output-sensitive algorithm of Ghosh and Mount [19] in $O(n \log n + m)$ time, where m is the number of edges in the visibility graph. After computing the visibility graph we can use Dijkstra's algorithm [14] to compute the required shortest path. In a weighted graph $G = (V, E)$, where all weights are positive, Dijkstra's algorithm with a heap can be implemented in $O(|E| \log |V|)$ time and with a more complex data structure of Fredman and Tarjan [18] in $O(|E| + |V| \log |V|)$ time. So the shortest path in the free-space using this method can be computed in $O(n \log n + m)$ time. But m, the number of edges in the visibility graph can be as large as $O(n^2)$. Hence this method doesn't give us a subquadratic time algorithm for the shortest-path computation. So for the shortest-path computation we cannot afford to compute the entire visibility graph because the shortest path will not use most of its edges. In fact whether a $o(n^2)$ time algorithm exists to compute the shortest path between any two points in the free-space in Euclidean metric remained an open problem for a long time. Only recently Mitchell [40] came up with a subquadratic time algorithm for the shortest-path problem in Euclidean metric. The algorithm computes the L_2 shortest distance between two points in the presence of polygonal obstacles in $O(n^{3/2+\epsilon})$ time. This algorithm was subsequently improved by Hershberger and Suri [27]. From a fixed source s, specified during the input, their algorithm computes the shortest-path map

in $O(n \log^2 n)$ time and $O(n \log n)$ space to answer the shortest-distance query in $O(\log n)$ time.

But the Euclidean shortest-path problem in the presence of only vertical line segments is more well behaved. Here in this setup the shortest path has the monotonicity property. In other words the shortest path between any two points in the presence of vertical line segments is always monotone in the horizontal direction. This property was exploited in the result of Lee and Preparata [32]. They proposed a solution to compute a shortest path between two points in optimal $O(n \log n)$ time using plane sweep where n is the total number of vertical line segments. Their algorithm also builds the shortest-path-map from a source in $O(n \log n)$ time to answer the shortest-path query to the source in $O(\log n + k)$ time where k is the number of segments in the reported path.

This work was subsequently generalized by de Rezende, Lee and Wu [12] to the case of n isothetic disjoint rectangles but in rectilinear or L_1 metric. They presented an optimal $O(n \log n)$ time preprocessing algorithm to answer the shortest-distance query from the query point to a source s which is specified as a input before the preprocessing step. The query time of their algorithm is $O(\log n)$ for the shortest-distance query and $O(\log n + k)$ for the shortest-path query where k is the size of the reported path.

This work was further generalized by Clarkson, Kapoor and Vaidya [8] for arbitrary polygonal obstacles. They solved the rectilinear shortest-path problem from a given source in L_1 metric. They build up a structure called the **Sparse Visibility Graph** whose total number of edges as well as total number of vertices is $O(n \log n)$ where n is the total number of polygonal vertices. This is much smaller in size than the entire visibility graph and its construction will be given in more detail in **section 4.1**. Subsequently Dijkstra's algorithm on this graph is used to compute a shortest-path tree from the source. So the total time required to compute the shortest distance using their method is $O(n \log^2 n)$. Independently Mitchell [36] also obtained the same bound for this problem.

2.2 Approximations in Shortest-Path Problems

The problem of approximate shortest-paths has been studied in the context of spanners. Usually a spanner $G' = (V, E')$ of a graph $G = (V, E)$ is a subgraph of G such that for any two vertices $v_1, v_2 \in V$ the shortest distance between v_1 and v_2 in G' is at most a constant times the shortest distance between them in G. In computational geometry literature, the spanner for a set of points was studied first. Chew [10] introduced the notion of graphs that approximate the complete Euclidean graph. He showed that the shortest-distance between two points in L_1 metric Delaunay triangulation is at most $\sqrt{10}$ times the L_2 distance between them. Subsequently Dobkin, Friedman and Supowit [13] proved that the shortest path in a L_2 metric Delaunay triangulation is at most $(1+\sqrt{5})\pi/2$ times the L_2 distance between those two points. This constant was subsequently improved to 2.42 by Keil and Gutwin [30].

Clarkson [7] gave an approximation algorithm to compute a path whose length $(1+\epsilon)$ times the optimal L_2 distance. The algorithm requires $O(n/\epsilon + n\log n)$ time after building up a data structure of size $O(n/\epsilon)$ in $O((n/\epsilon)\log n)$ time.

2.3 Parallel Shortest-Path Algorithms

Now we will take a brief look at parallel computational geometry with the focus on the shortest-path problem. In the design of parallel algorithms the following two principles are very crucial.

- *The design of* **NC** *algorithms, i.e., we should use a polynomial number of processors and polylogarithmic time.*
- *The minimization of the processor-time product. In the ideal case this product should be equal to the sequential lower bound of the problem.*

In the design of a parallel algorithm for the single-source shortest-path problem the difficult part is to bring down the processor time product. The sequential single-source shortest-path algorithms in graphs use dynamic programming and are hard

to parallelize efficiently. To design efficient parallel algorithms divide-and-conquer is the usual paradigm. The main problem that is encountered while parallelizing the single-source shortest-path problem is the transitive closure bottleneck. In other words, the processor-time product in a **NC** algorithm for the single-source shortest-path algorithm tends towards the sequential complexity for the all-pairs shortest-path problem.

An efficient parallel algorithm for the shortest-path problem inside a n-vertex simple polygon was first proposed by ElGindy and Goodrich [16]. Their algorithm can compute the Euclidean shortest path between a source and a destination inside a simple polygon using $O(n)$ processors in $O(\log n)$ time. Also it can compute the shortest-path tree from a vertex of a simple polygon in $O(\log^2 n)$ time using $O(n)$ processors. Both algorithms assume the *CREW PRAM* model of computation. The approach is based on centroid decomposition of the tree that corresponds to the dual of the triangulation. Subsequently Hershberger [26] proposed a parallel algorithm to compute the shortest-path tree inside a simple polygon using $O(n/\log n)$ processors and $O(\log n)$ time.

Parallel algorithms for the shortest path in free space had been studied only very recently. Atallah and Chen [2] proposed an algorithm to report the rectilinear shortest distance between two arbitrary query points in the presence of n isothetic and disjoint rectangular obstacles. Their algorithm reports the shortest-distance query in $O(\log^2 n)$ time with a single processor after preprocessing the input with $O(n^2)$ processors in $O(\log^2 n)$ time in the *CREW PRAM* model. The approach is based on recursive splitting of the set of rectangles using staircase separators and the use of efficient parallel algorithms for monotone matrices. Recently Atallah and Chen [3] improved the processor complexity in the preprocessing algorithm to $O(n^2/\log n)$.

2.4 Other Areas of Shortest-Path Problems

There have been several other works on the shortest-path problem. Here we will briefly mention a few of them.

- **Shortest Path in Higher Dimensions :** In the L_2 metric, the shortest-path problem between two points avoiding a set of polyhedral obstacles had been proved to be NP-hard by Canny and Reif [9]. The difficulty in computing the Euclidean shortest-path in higher dimension arises due to the fact that the break-points of the shortest path can be an interior point of an edge unlike two dimensions. Papadimitriou [42] proposed a fully polynomial time approximation scheme to compute an approximate shortest-path between two points avoiding a set of polyhedral obstacles in three-dimensions.

  In contrast to the L_2 metric the shortest-path problem is polynomially solvable in the rectilinear metric in three-dimensions. Clarkson et al [8] proposed an algorithm to compute a L_1 shortest-path between two points avoiding a set of n non-intersecting three-dimensional rectilinear obstacles, in $O(n^2 \log^3 n)$ time.

- **Weighted Region Problems :** In the usual shortest-path problem we have to compute a path avoiding a set of polygonal obstacles. This can be conceived in the following way. We assign costs per unit distance to travel within a region. So in the usual shortest-path problems we can assign unit cost to the free space and infinite costs to obstacles. But in practice an obstacle cost may not be so large. So we can assign finite costs greater than one to those obstacles. Our goal is to move from the source to the destination along a path with the least cost. This problem was studied by Mitchell and Papadimitriou [39].

- **Minimum Link Paths :** Another measure of path length is the number of links in the path. The link distance between two points in the plane is the number of links in an obstacle free polygonal path that joins the two points with the minimum number of links. Suri [47] solved the link distance problem inside a triangulated simple polygons in linear time. Mitchell et al., [38] solved the minimum link path problem among obstacles in the plane.

- **Bicriteria Shortest Paths :** A minimum-link path may be far from the optimal with respect to the length. Similarly a shortest path may have lot more links than the minimum length path. Thus in many situations it is desirable

to find a path with the minimum length and has few links. This problem and other variations of the "bicriteria" path problem has been investigated in Arkin et al., [1].

- **On-line Shortest-Path Algorithm :** Many times in motion planning problems a complete description of all the obstacles are not known in advance. The robot comes to know about each obstacle when it comes within its line of sight. The goal here is to move from the source to the destination along a nearly optimal path. There is a negative result in this area which states that if obstacles are not required to have a bounded ratio between their lengths and widths then there is no heuristic that can produce a path that is at most a constant times the optimal. This result is due to Papadimitriou and Yannakakis [44].

For a more detailed survey on these problems please refer to Mitchell [37] or Papadimitriou [43].

Chapter 3

Shortest-Path Query

3.1 Preliminaries

Given a set $\mathcal{R}$ of n barriers, the shortest-path query problem **SPQ** asks for a preprocessing of $\mathcal{R}$ such that a description of the shortest path between two points (*origin* and *destination*) can be reported efficiently. Based on the type of barriers in $\mathcal{R}$, the metric, the dimension, and the type of queries, a variety of **SPQ** problems may be defined.

In case the barriers form the boundary of a simple polygon, [25] describe a data structure, that can be constructed in linear time once the polygon has been triangulated, that supports shortest Euclidean length queries between two arbitrary points in logarithmic time. The path itself can be retrieved in additional time proportional to its number of turns. The result is based on the idea of constructing a hierarchy of nested subpolygons over an underlying triangulation, whose dual is known to have a tree form, such that any shortest path crosses a small number of subpolygons.

In case the environment consists of axes parallel or isothetic boxes as the barriers and a known origin for all subsequent shortest-path queries, algorithms for various settings have been presented in [8], [12], [36]. For isothetic rectangular obstacles, de Rezende et al [12] presented an $O(n \log n)$ algorithm to compute the shortest-path map from a source point specified during the input. Their approach uses the monotonicity of the shortest path in at least one of the two axes directions.

In this chapter we present efficient sequential algorithms for the **SPQ** problem where the barriers in $\mathcal{R}$ are disjoint planar rectangles whose sides are parallel to the coordinate axes, and subsequent queries ask for the shortest L_1 path between two arbitrary points that avoids the barriers in $\mathcal{R}$. The segments forming such path are also restricted to be parallel to the coordinate axes. The heart of our contribution is utilizing the geometric property of monotonicity of the shortest path between every pair of co-planar points, in this setting, to construct three planar graphs, called carrier graphs, which either contain the shortest path between every two corners of the rectangles in $\mathcal{R}$ or can be used to guide the search for such path. Each graph can then be searched using purely graph theoretic techniques. As will be demonstrated through the chapter, the construction of these graphs can be performed efficiently.

In the following section we describe the carrier graphs, which contains the shortest-path information, and study their properties. In **section 3.2** we present an algorithm for constructing these graphs to answer queries for length of the shortest L_1 path between any two corner points of rectangles in $\mathcal{R}$. In **section 3.3** we show how to preprocess these carrier graphs to answer the shortest-distance or the shortest-path query. The first algorithm described in **section 3.3.1** performs $O(n^2)$ time preprocessing to answer each subsequent query between two corner points in $O(1)$ time. The second algorithm described in **section 3.3.1** performs $O(n^{1.5})$ preprocessing to answer each subsequent query between two corner points in $O(\sqrt{n})$ time. The retrieval of the shortest path can be performed in time proportional to the number of its segments. The details of this step had been described in **section 3.3.2. Section 3.4** is devoted to the details of removing the restriction that query points are corner points of rectangles in $\mathcal{R}$.

3.2 Carrier Graphs

In this section we introduce a class of planar directed-acyclic graphs, called *carrier graphs*, and demonstrate that they contain sufficient information to support shortest-path queries when both origin and destination are part of the query.

We now introduce three needed carrier graphs, denoted by G_{+x}, G_{+y} and G_{-y}.

Each graph is constructed to be a directed acyclic graph which contains paths that are monotonic with respect to a specified direction. Those graphs exploit the knowledge that each shortest path in this setting is monotone in at least one of the x or y directions [12]. Only G_{+x} will be described, since the other two graphs are defined in a similar fashion.

Let $\mathcal{T}$ be the set of line segments obtained by connecting each corner point on the right side of a rectangle in $\mathcal{R}$ to its right shadow with respect to $\mathcal{R}$. We first define the undirected graph $UG_{+x} = (V, E)$ as follows (please refer to figure 3.1 & figure 3.2 for illustrations) :

- V consists of the corner points of rectangles in $\mathcal{R}$, and the right shadows of the corner points on the right side of the rectangles in $\mathcal{R}$ with respect to $\mathcal{R}$, and upper and lower shadows of the corner points on the left sides of the rectangles in $\mathcal{R}$ with respect to $\mathcal{R} \cup \mathcal{T}$.
- Two vertices $a, b \in V$ are connected by an edge $e = (a, b) \in E$ if and only if
  - a and b are consecutive points on the side of a rectangle in $\mathcal{R}$ or on an element in $\mathcal{T}$, or
  - a is the upper or lower shadow of a corner point with respect to $\mathcal{R} \cup \mathcal{T}$ and b is the corresponding corner point in $\mathcal{R}$.

  In addition, a weight $w(e)$ equal to the distance between corresponding points is assigned to each edge.

We complete the definition of the carrier graph G_{+x} by performing the following orientation and transformation :

1. For each element in E that corresponds to a horizontal edge we assign a direction from the endpoint having lower $x-$coordinate to the endpoint having larger $x-$coordinate.

2. For each element in $(u, v) \in E$ that corresponds to a vertical edge, we split each vertex incident on E into 2 vertices (u into u' and u''; v into v' and v''). The vertex with lower $x-$coordinate than v and adjacent to it if any is joined to v', and

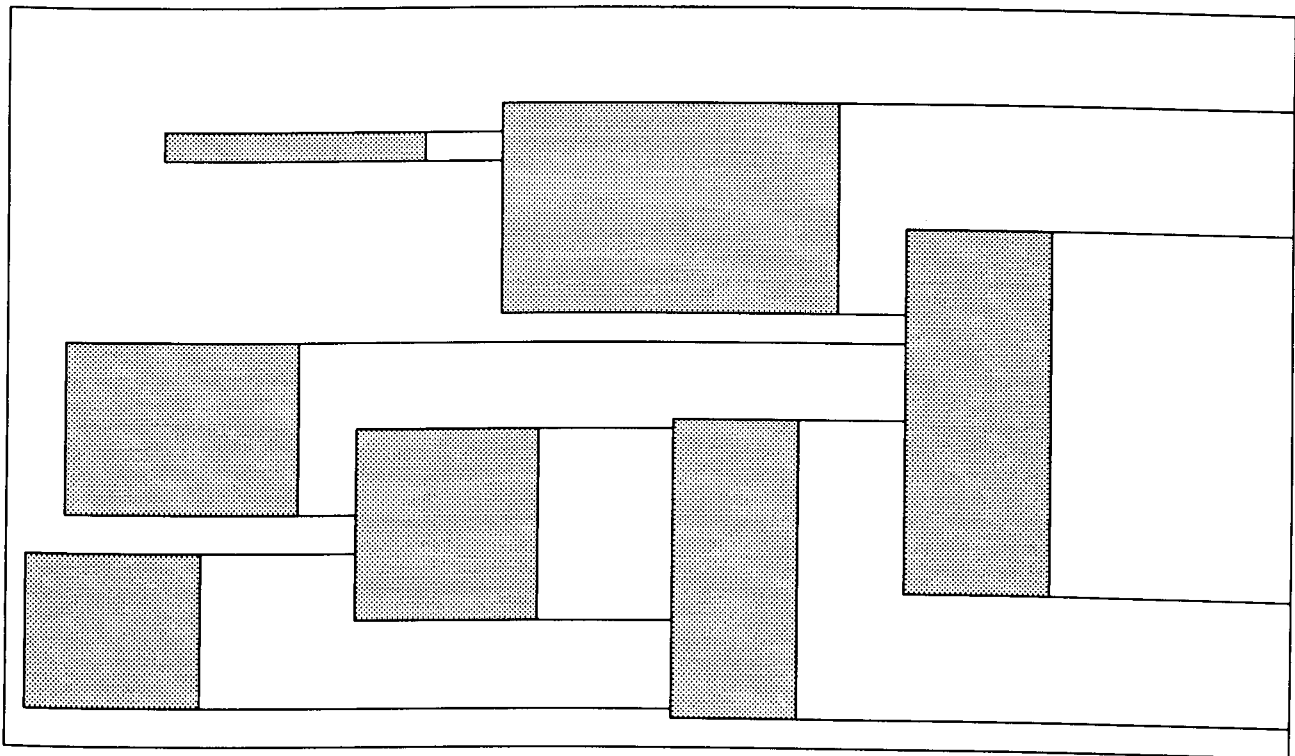


Figure 3.1: Construction of the Carrier Graph G_{+x} : Step I - Right Shadows

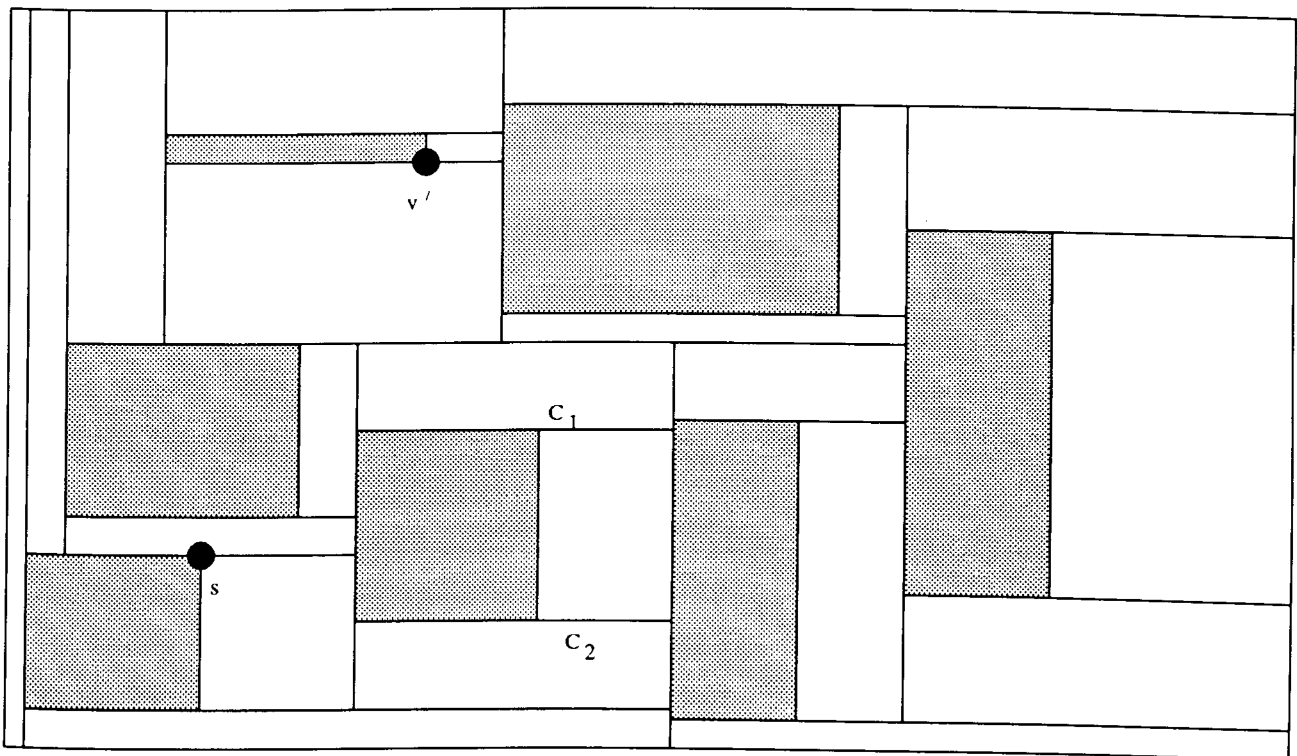


Figure 3.2: Construction of the Carrier Graph G_{+x} : Step II - Upper and Lower Shadows

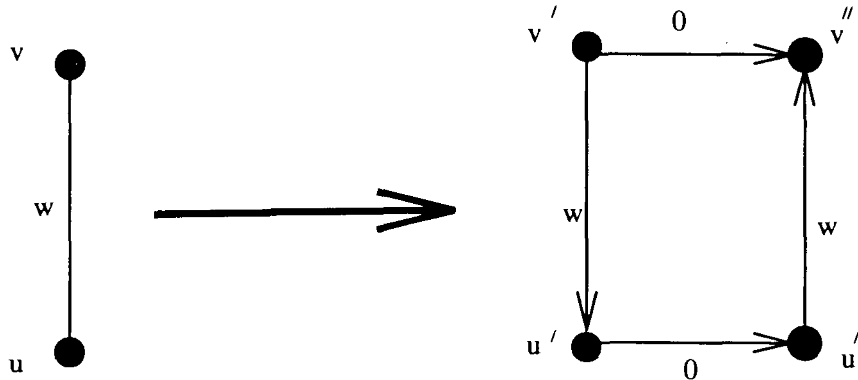


Figure 3.3: Construction of the Carrier Graph G_{+x} : Step III - Orientation

the vertex with higher $x-$coordinate than v and adjacent to it if any is joined to v''. v' is joined to v'' with a directed edge of zero weight. The vertex u is handled in a similar fashion. v' is joined to u' with a directed edge of weight w, and u'' is joined to v'' with a directed edge of weight w. w is the weight of the edge (u, v). An illustration of this transformation is given in figure 3.3. Also please refer to illustrations figure 3.4 and figure 3.5 for an undirected carrier graph UG_{+x} and the directed carrier graph G_{+x} after the transformation.

We now demonstrate that the defined carrier graphs are sufficient to support shortest-path queries through the following properties.

Lemma 3.1 [12] *Given a set $\mathcal{R}$ of disjoint orthogonal rectangles and a point s, for each point in R_1 (R_3, R_5, R_7) defined with respect to s, there exists a shortest path to s that is monotone in the $+x$ direction ($+y$, $-x$, $-y$ direction respectively). For each point in R_2, R_4, R_6, and R_8 there exists a shortest path to s that is monotone in both the x and y-directions.*

Lemma 3.2 *For each vertex $s \in V$ that corresponds to a corner point of a rectangle in $\mathcal{R}$, the vertices in V that correspond to points left of the chains C_3 and C_8 emanating from s are* **not** *reachable from s in G_{+x}.*

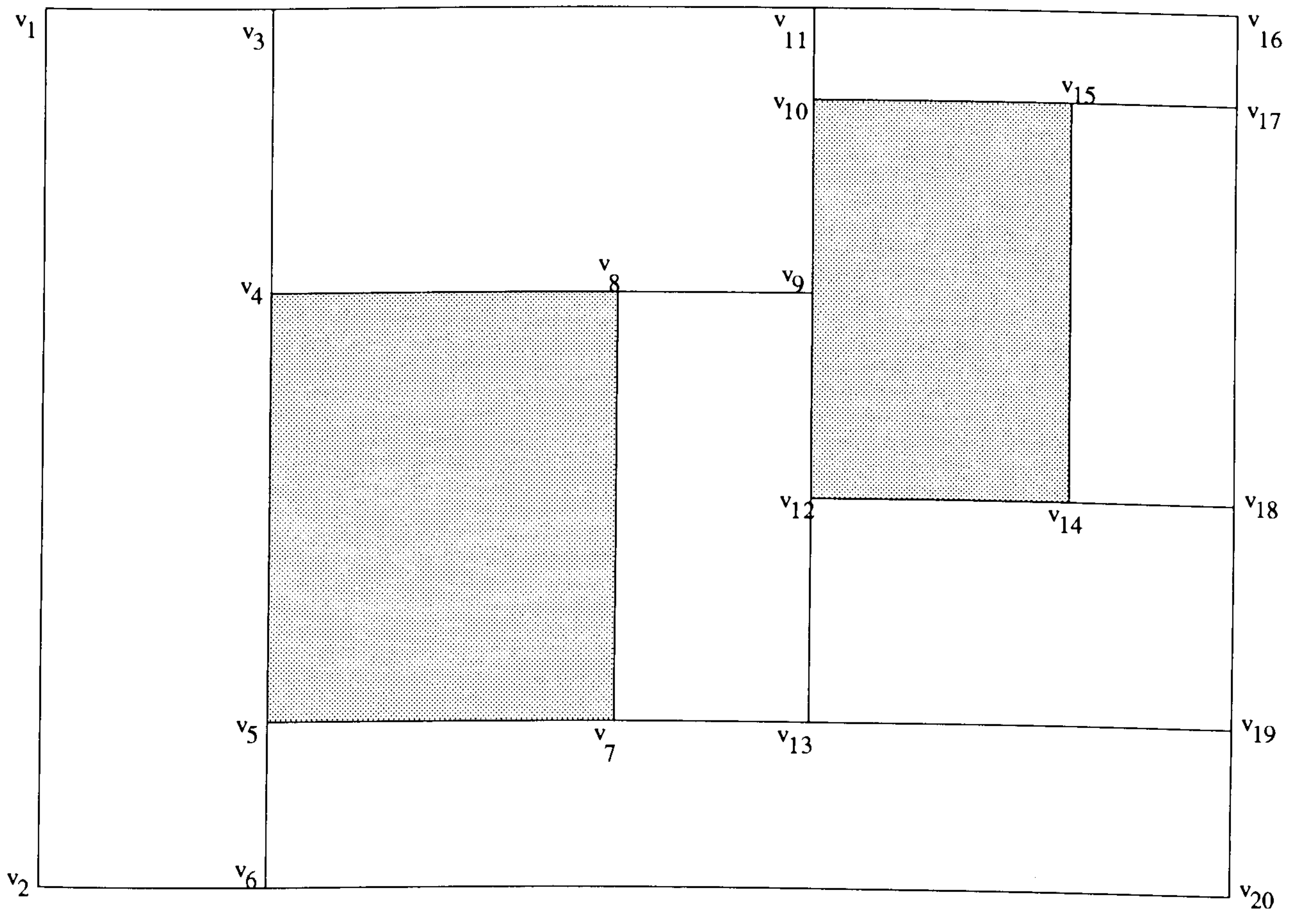


Figure 3.4: An undirected carrier graph UG_{+x}

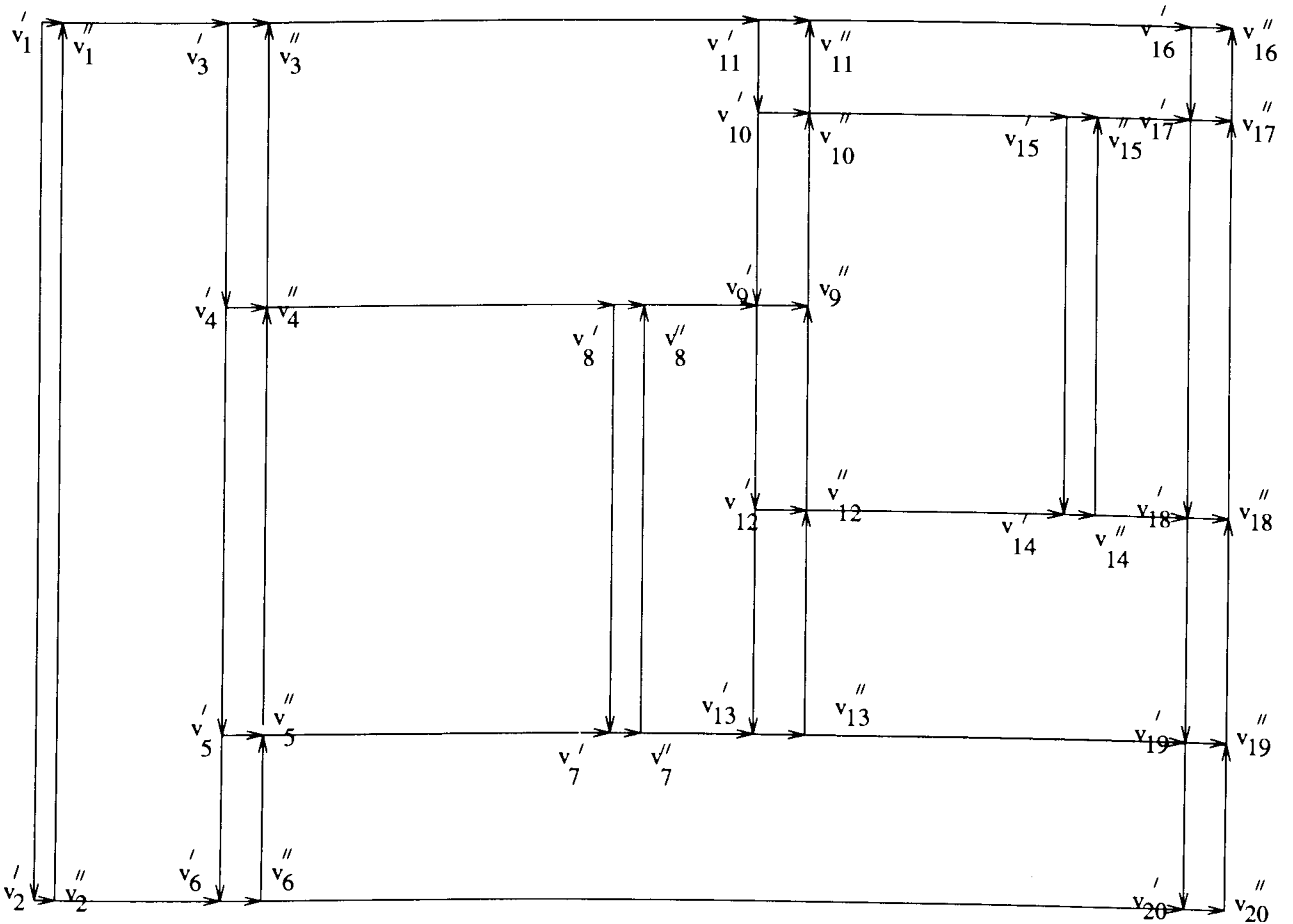


Figure 3.5: A directed carrier graph G_{+x}

Proof : de Rezende et al [12] proved that a vertex on the left of the chain C_3 and C_8, emanating from s, cannot be reached by a $+x$ monotone path starting from s. The orientation of the edges in G_{+x} enforces that every path emanating from s is monotone in $+x$ direction. Therefore, all vertices on the left of two chains C_3 and C_8 are not reachable from s in G_{+x}. □

Lemma 3.3 *For each vertex $s \in V$ that corresponds to corner point of a rectangle in $\mathcal{R}$, the paths corresponding to two chains C_1 and C_2 lie in G_{+x}.*

Proof : From the definition of the graph UG_{+x}, it is easy to see that the segments forming chains C_1 and C_2 are edges in the undirected graph. In the chain C_1, a vertical segment that corresponds to an edge $e = (u, v) \in UG_{+x}$ is mapped into the sequence of two edges $((v_1, u_1); (u_1, u_2))$ in G_{+x}. Similarly in the chain C_2, a vertical segment which corresponds to an edge $e = (u, v) \in UG_{+x}$ is mapped into the sequence of two edges $((u_1, u_2); (u_2, v_2))$ in G_{+x}. In both cases we can see that a path corresponding to the chain is maintained and its cost remains unaltered. □

Lemma 3.4 [12] *For each vertex $s \in V$, let V_l denote the set of vertices belonging to the left vertical sides of the subset of $\mathcal{R}$ in the region $R_{+x} = R_8 \cup R_1 \cup R_2$. For any point $u \in R_{+x}$ there is a shortest path from s to u which reaches u via either of the two vertices $u_a, u_b \in V_l$, where u_a is the vertex having the lowest ordinate among all vertices in V_l which are visible from u having lower x-coordinate and higher y-coordinate compared to u. Similarly u_b is the vertex having the highest ordinate among all vertices of V_l which are visible from u having lower x-coordinate and lower y-coordinate compared to u. Please refer to figure 3.6 for an illustration for the lemma.*

Lemma 3.5 *For each vertex $s \in V$, the shortest paths to all the vertices, that lie in the region R_1 of s, lie in G_{+x}.*

Proof : We will use an inductive argument to prove the lemma by ranking the corner points in R_1, excluding those in C_1 and C_2, according to their $x-$coordinates. Vertices on the two chains are assigned the rank of zero. Our assumption in the

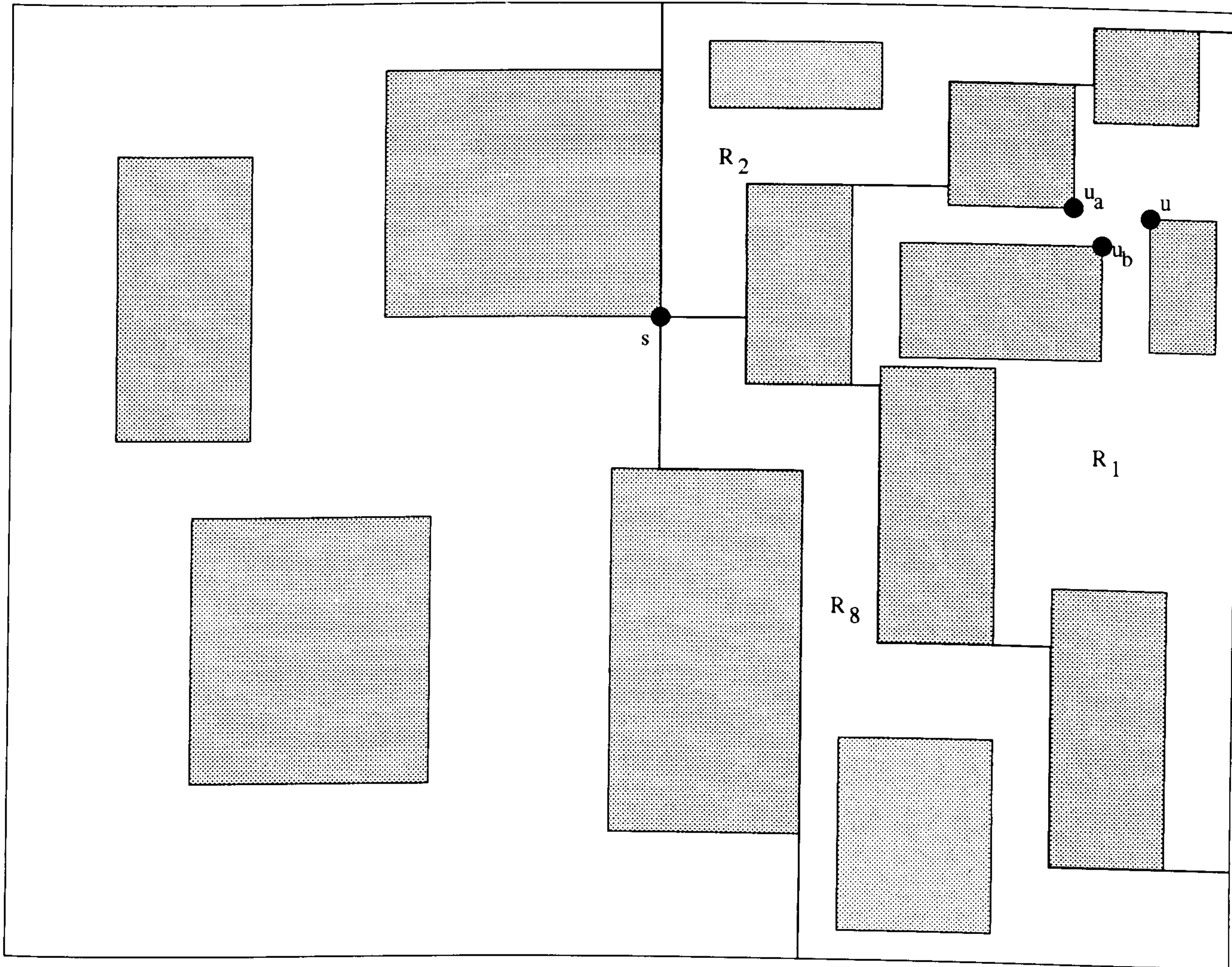


Figure 3.6: An Illustration for Lemma 3.4

induction is that the shortest path from s to every vertex in the region R_1 having rank less than m is maintained in G_{+x}. We have to show that this holds for vertices with mth ranked vertex.

The basis trivially holds for all vertices on chains C_1 and C_2. Now let u be the vertex with mth rank. It follows from the fact that the two chains C_1 and C_2 emanating from s are maintained in G_{+x}, as shown in **Lemma 3.3**, that no horizontal edge starting from a vertex on the left of either C_1 or C_2 can cross the region R_1 of s. Therefore, each vertex $u \in V$ in the region R_1 is connected to each of the two vertices u_a and u_b in R_1, described in **Lemma 3.4**, with a path whose length equals to the L_1 distance from u. From our inductive hypothesis, the shortest-paths from s to both u_a and u_b are maintained in G_{+x}. Also u is joined to both u_a and u_b by shortest paths. Thus the lemma follows. □

Lemma 3.6 *If a vertex $t \in R_2 \cup R_8$ corresponding to a vertex $s \in V$ is reachable from s in G_{+x}, then the shortest path from s to t is also maintained in G_{+x}. (Note that some vertices in R_2 and R_8 may be unreachable from s as demonstrated by v' in Figure 3.7.)*

Proof : By contradiction. Let us assume that $t \in R_2$, the case of $t \in R_8$ is handled similarly, is reachable from s but the shortest path from s to t is not maintained in G_{+x}. Let u be the corner point of the shortest path from s to t in G_{+x} where the first reversal in the $y-$direction occurred, and let w be the preceding corner point on the path. Therefore the path from s to w denoted by P, must be monotone in both $x-$ and $y-$ directions (refer to figure 3.7 for illustrations).

We now construct a path, denoted by P_1, from u to s along the reverse direction of the edges of G_{+x} as follows :

- Starting from u we move down a vertical edge in G_{+x} until a horizontal edge is reached; we then move in the reverse direction until we can resume the motion along vertical edges. Repeat the above procedure until we intersect either C_2 (which is implicitly maintained in G_{+x}), or P or both at s. Let z be the first such intersection point.

Figure 3.7: Illustration for Lemma 3.6

The chain P_1 is clearly monotone in both the $x-$ and $y-$ directions. Therefore, the path from s to u obtained by concatenating the portion of C_2 or P from s to z and the part of P_1 from z to u, whose segments are edges in G_{+x}, is monotone in both directions and is thus shorter than the path from s to u via w. This is a contradiction, and thus the lemma follows. □

Lemma 3.7 *If s and t are two vertices in V, under the assumption that t lies in the first quadrant with respect to s (i.e., $s_x < t_x$ and $s_y < t_y$), and t is not reachable in both G_{+x} and G_{+y} from s, then the graphs G_{+x} and G_{+y} contain sufficient information to construct a path from s to t with the shortest length.*

Proof : In this case we can easily conclude, by **Lemmas 3.2 & 3.5**, that t lies in region R_2 of s. Therefore, any shortest path between s and t is monotone in both x and y directions and has a length of $|t_x - s_x| + |t_y - s_y|$.

In addition, it follows from the facts that the chain C_2 is maintained in G_{+x} and t is not reachable from s in G_{+x} that a path, denoted by P, emanating from t along the reverse direction of the edges in G_{+x} as follows :

- Starting from t we move down a vertical edge in G_{+x} until a horizontal edge is reached, we then move in its reverse direction until we can resume the motion along vertical edges. Repeat the above procedure until infinity is reached.

This path P does not intersect C_2. Therefore, it must intersect the chain C_3, which is implicitly maintained in G_{+y}, if two graphs were to be overlaid. Let z be the first such intersection point. The chain obtained by concatenating the portion of C_3 from s to z and the part of P from z to t is monotone in both $x-$ and $y-$ directions, and is thus a path with the shortest length from s to t. □

We now state the main result of this section as follows

Theorem 3.1 *The carrier graphs G_{+x}, G_{+y} and G_{-y} contain sufficient information to support shortest-path queries when both* origin *and* destination *are corner points of the rectangles in $\mathcal{R}$.*

3.3 Shortest Path Queries Between Two Corner Points

In this section we will present algorithms for preprocessing a set of barriers $\mathcal{R}$ such that subsequent shortest distance queries between two arbitrary corner points can be answered in sub-linear time. Our approach is to compute the carrier graphs and some additional information to support such queries. We will then describe a simple method for constructing a path with minimum length in time proportional to the number of its segments.

3.3.1 Sequential Preprocessing and Query Algorithms

A straightforward idea is to compute and store the transitive closure of G_{+x}, G_{+y}, and G_{-y}. Since each of the graphs is planar, directed and acyclic, use the following result to compute the all-pairs shortest-paths for each graph in $O(n^2)$ time.

Lemma 3.8 [28] *The single source shortest path problem on a directed acyclic graph $G = (V, E)$ can be solved in $O(|V| + |E|)$ time.*

A subsequent query of two arbitrary corner vertices u and v (without loss of generality we assume v lies in the first quadrant of u) can now be preprocessed in $O(1)$ time as follows :

- If $dist(u, v) = min\{d_{G_{+x}}(u, v), d_{G_{+y}}(u, v)\} \neq \infty$ then $sd(u, v) = dist(u, v)$
- Otherwise $sd(u, v) = |v_x - u_x| + |v_y - u_y|$

where $d_G(u, v)$ denotes for the shortest distance between vertices u and v in G. Therefore we have established the following result :

Theorem 3.2 *The orthogonal shortest distance between two arbitrary corner points can be determined in $O(1)$ time with $O(n^2)$ preprocessing and storage.*

Proof : The correctness of the above procedure follows directly from properties of the carrier graphs proved in **Theorem 3.1** and correctness of the algorithm of **Lemma 3.8**. Each of the carrier graphs is constructed by applying the well known plane-sweep paradigm twice, a procedure that requires $O(n \log n)$ time. However, this step is dominated by the time required to compute and store the transitive closure for each graph. The time complexity of the search procedure of $O(1)$ is clearly correct, and thus the theorem follows. ◻

A more efficient preprocessing procedure makes use of the planar-separator theorem [34] which is as follows.

Theorem 3.3 *The n vertices of a planar graph G can be partitioned into three sets S, V_1 and V_2, such that no edge joins a vertex from V_1 with a vertex from V_2, neither V_1 nor V_2 contains more than $2n/3$ vertices, and S contains no more than $2\sqrt{2}\sqrt{n}$ vertices.*

This preprocessing algorithm consists of building a shortest-path-search **SPS** tree for each of the carrier graphs. At each node, we store vertices of a $O(\sqrt{n})$ separator S of the corresponding graph together with the shortest distance from each vertex in S to all vertices of the graph. A detailed description of this procedure for G_{+x} is as follows :

Build SPS(G_{+x})-Tree

- **S-1.** if $|V| \geq c$, where c is a constant, then
  - (a) compute a separator S of UG_{+x}
  - (b) compute and store the shortest distance from all the vertices in G_{+x} to each vertex in S.
  - (c) identify the connected components of G_{+x} - S and for each vertex store the label of its component
  - (d) for each connected component, build *SPS-tree* in a recursive fashion
- **S-2.** otherwise, compute the transitive closure of G_{+x}.

This preprocessing allows us to search for the shortest distance between two vertices when SP-QUERY(s,t,root) is invoked, where root is the pointer to the root node of **SPS** tree.

SP-QUERY(s,t : vertex; n : pointer to a node in **SPS** tree)

- **Case I.** if the current node is a leaf in the *SPS-tree*, then we retrieve the distance between s and t from the transitive closure stored at this node.

- **Case II.** if the current node is an internal node in the *SPS-tree* and s and t belong to different connected components along its descendents, then

$$dist(s,t) = \min_{v_i \in S}\{d(s,v_i) + d(v_i,t)\}$$

  We can also use the same formula if either s or t or both belong to S.

- **Case III.** If the current node is an internal node in the *SPS-tree* and s and t belong to the same connected component among its descendents, then we search the *SPS-tree* for the shortest distance between s and t along a path whose vertices solely lie in the identified component, denoted by $d'(s,t)$. The shortest distance is now computed as

$$dist(s,t) = \min\{\min_{v_i \in S}\{d(s,v_i) + d(v_i,t)\}, d'(s,t)\}$$

Lemma 3.9 *The algorithm* **SP-QUERY** *correctly computes the shortest distance between vertices s and t.*

Proof : To prove the correctness of **SP-QUERY** we first explain how the shortest path between any two vertices in G can behave. For this we refer to figure 3.8. For the ease of explanation we assume that the removal of the planar separator S from UG_{+x} disconnects it into two disjoint components. A similar argument will hold in the case there are multiple components. Let us denote the set of vertices in these two components by V_1 and V_2.

The first case is when the vertex s belongs to V_1 and the vertex t belongs to V_2, as shown in figure 3.8(a). Let us consider the two parts of the shortest path from s to t, i.e., the path from s to s_k, denoted by $P_1(s, s_k)$ and the the path from s_k to t, denoted by $P_2(s_k, t)$. Now $P_1(s, s_k)$ must be the shortest path from s to s_k. Similarly the path $P_2(s_k, t)$ will be the shortest path from s_k to t. Both $P_1(s, s_k)$ and $P_2(s_k, t)$ are determined in **S-1(b)**, where we compute the shortest-path tree from each separator vertex. Hence in this case the shortest distance between s and t will be correctly identified in step **step II** of **SP-QUERY**. Similar argument holds if either of s or t belongs to S.

If both vertices s and t belong to V_1 (or V_2) then we have to check for one more candidate path $P_3(s, t)$ which completely lies in V_1 (or V_2), as shown in figure 3.8(b), and **step III** of **SP-QUERY** in that case correctly identifies the shortest distance. □

Lemma 3.10 *An orthogonal shortest-distance query between two vertices in G_{+x} can be processed in $O(\sqrt{n})$ time with $O(n\sqrt{n})$ preprocessing time and storage.*

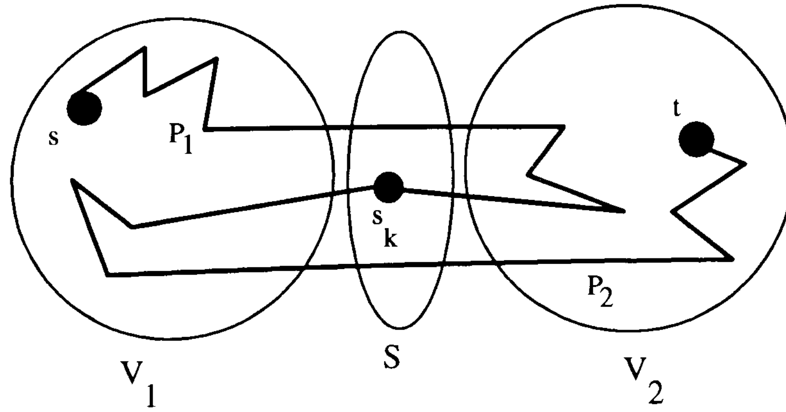
Proof: Computing the planar-separator requires $O(n)$ time [34], and identifying connected components can also be performed in $O(n)$ time. However, computing the shortest distances from all vertices in the graph to separator vertices requires constructing the shortest path tree from the $O(\sqrt{n})$ separator vertices. Using the algorithm of **Lemma 3.8**, since the graph is planar, directed and acyclic, leads to a total $O(n\sqrt{n})$ running time. Therefore, the total time complexity of the algorithm **Build** can be expressed as

$T(n) = \sum_{i=1}^{k} T(\alpha_i n) + c * n\sqrt{n}$, where $\sum_{i=1}^{k} \alpha_i \leq 1$, $0 < \alpha_i \leq 2/3$, $c \geq 0$ is a constant.

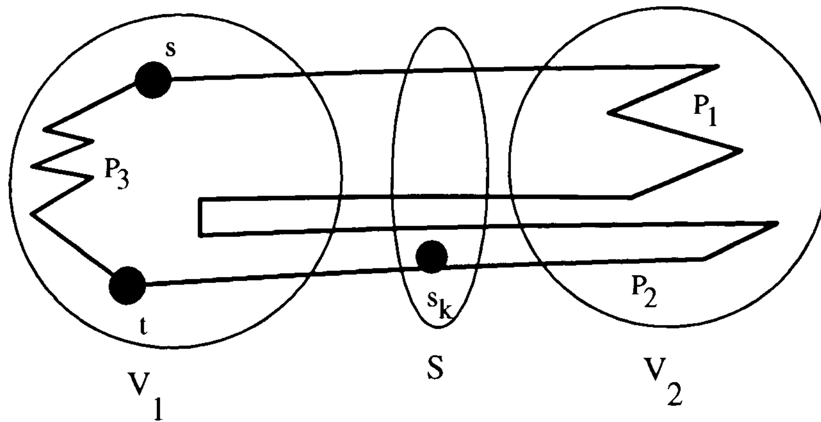
The solution of the recurrence is $T(n) \in O(n\sqrt{n})$.

The time required to process each subsequent query, in the worst case, can be clearly expressed by the following recurrence

$$Q(n) \leq Q(2/3n) + c * \sqrt{n},$$ where c is a positive constant.



(a)



(b)

Figure 3.8: Illustration for Lemma 3.9

whose solution leads to $Q(n) \in O(\sqrt{n})$. Thus the lemma follows. □

After performing the same preprocessing on the remaining two carrier graphs G_{+y} and G_{-y}, the shortest distance query between two corner points u and v (assume, without loss of generality that v lies in the first quadrant of u) of $\mathcal{R}$ can be reported as follows :

- if $dist(u,v) = min\{d_{G_{+x}}(u,v), d_{G_{+y}}(u,v)\} \neq \infty$ then $sd(u,v) = dist(u,v)$
- otherwise, $sd(u,v) = |v_x - u_x| + |v_y - u_y|$

which leads us to the main result of this subsection.

Theorem 3.4 *An orthogonal shortest-distance query between two arbitrary corner points amidst the set of n rectangular barriers $\mathcal{R}$ can be processed in $O(\sqrt{n})$ time with $O(n\sqrt{n})$ preprocessing time and storage.*

3.3.2 Constructing a Shortest Path

When an actual shortest path needs to be constructed, two different situations occur. We handle each situation separately.

- **Case 1** Shortest path is maintained in a single carrier graph

  In the preprocessing approach of **Theorem 3.2** we maintain an additional matrix of size $O(n^2)$, for each carrier graph. The entry (i,j) in each matrix contains the location of the vertex which precedes the vertex v_j in the shortest path from v_i to v_j. That is, if such a path exists.

  In the preprocessing approach of **Lemma 3.10** we maintain for each vertex v_i, in each of the carrier graphs, the location of the vertices which precede v_i along the shortest paths from the separator vertices from which v_i is reachable.

  The above information can clearly be generated and stored within the same time and storage bounds of **Theorem 3.2** and **Lemma 3.10**.

- **Case 2** Destination is not reachable from the origin in any of the carrier graphs

  In this situation we know, by **Lemma 3.5**, that the destination point lies in R_2 or R_8 of s. We describe the case of R_2. The case of R_8 is handled similarly.

  To build the shortest path from s to t we need to compute the intersection point z of C_3 emanating from s and a path P emanating from t along the reverse direction of the edges of G_{+x} as follows :

  - Starting from t we move in the reverse direction of a vertical edge in G_{+x} until a horizontal edge is reached, we then move in its reverse direction until we can resume the motion along vertical edges.

  The existence of such an intersection was proved in **Lemma 3.7**.

  Let c and p denote the last vertex of the currently constructed chains C_3 and P respectively. We construct each of these two staircases one edge at a time in an alternating fashion according to the following conditions :

  - (a) If $c_x < p_x$ and $c_y < p_y$, then we continue the construction of both C_3 and P one edge at a time.
  - (b) If $c_x < p_x$ and $c_y > p_y$, then we start tracing back along C_3 while adding new edges to P until their intersection point is reached.
  - (c) If $c_x > p_x$ and $c_y < p_y$, then we start tracing back along P while adding new edges to C_3 until their intersection point is reached.

The total time spent to determine the path is bounded by the number of segments in the path in each of the two cases. Therefore we can conclude that

Theorem 3.5 *An orthogonal shortest path between two corner points can be determined sequentially in*

(i) $O(k)$ time with $O(n^2)$ preprocessing, or

(ii) $O(\sqrt{n}+k)$ time with $O(n\sqrt{n})$ preprocessing,

where k is the number of segments in the constructed shortest path.

3.4 Removal of Corner Point Restriction

In this section we describe the additional preprocessing required to remove the restriction that the origin and destination be corner points of the rectangular barriers. Informally our approach is based on observing that each of the origin and destination points can be associated with a pair of vertices in each of the carrier graphs (as shown in figure 3.9 & figure 3.10). The shortest paths between the identified vertices in each graph are sufficient to support the shortest path computation from the origin to destination. There is a maximum of *four* such paths in each graph. Therefore, at most *eight* **SPQ** problems need to be solved.

Assume, without loss of generality, that t lies in the first quadrant with respect to s and they don't belong to the same rectangle. We shoot a ray towards the right from s and let u_1u_2 be the edge of the rectangle which it hits first. In a similar way we shoot a ray towards the left from t and v_1v_2 be the edge of the rectangle which it hits first. In a similar way we obtain two more edges u_3u_4 and v_3v_4 by shooting two rays upwards and downwards from s and t respectively.

All the above computation can be efficiently performed during query time once we preprocess the horizontal and the vertical trapezoidation by constructing the hierarchical representation, introduced by Kirkpatrick [31]. For each subsequent query we perform point-location on the hierarchies to locate the rectangles in the subdivision that contain the point s and t. The query can be carried out in $O(\log n)$ time after $O(n)$ time preprocessing.

The shortest distance between s and t can now be computed as follows

- **Step I.** Compute the minimum among four following expressions : $L_1(s, u_i) + sd(u_i, v_j) + L_1(v_j, t)$, where $i, j \in \{1, 2\}$. Here $sd(a, b)$ denotes the shortest distance between a and b in G_{+x}.

  If the minimum value is defined then the shortest distance between s and t is found. Otherwise, we proceed to the next step.

- **Step II.** Compute the minimum among four following expressions : $L_1(s, u_i) + sd(u_i, v_j) + L_1(v_j, t)$ where $i, j \in \{3, 4\}$. Here $sd(a, b)$ denotes the shortest

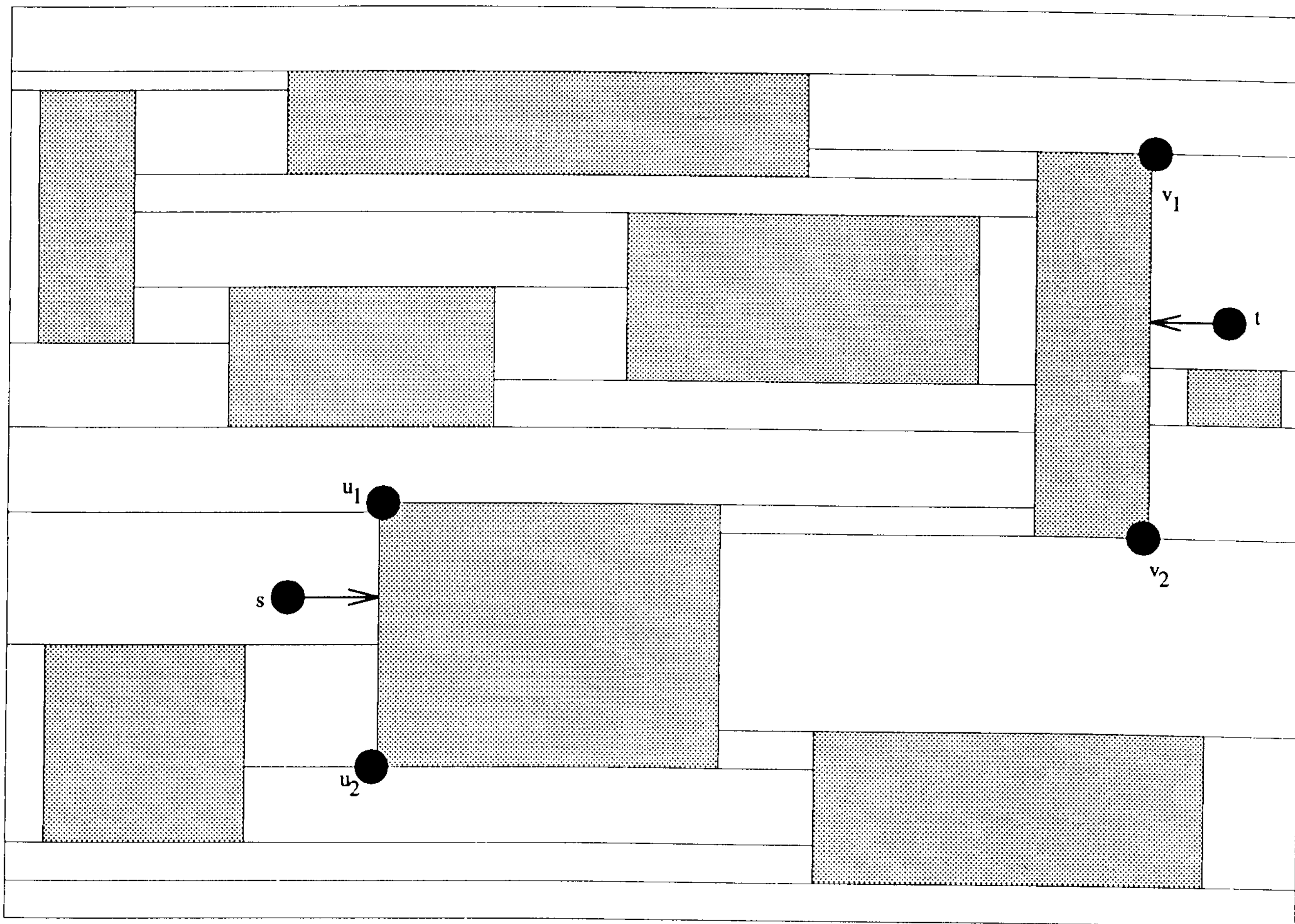


Figure 3.9: Corner Points in $\mathcal{R}$ Obtained by Horizontal Ray Shooting from s & t

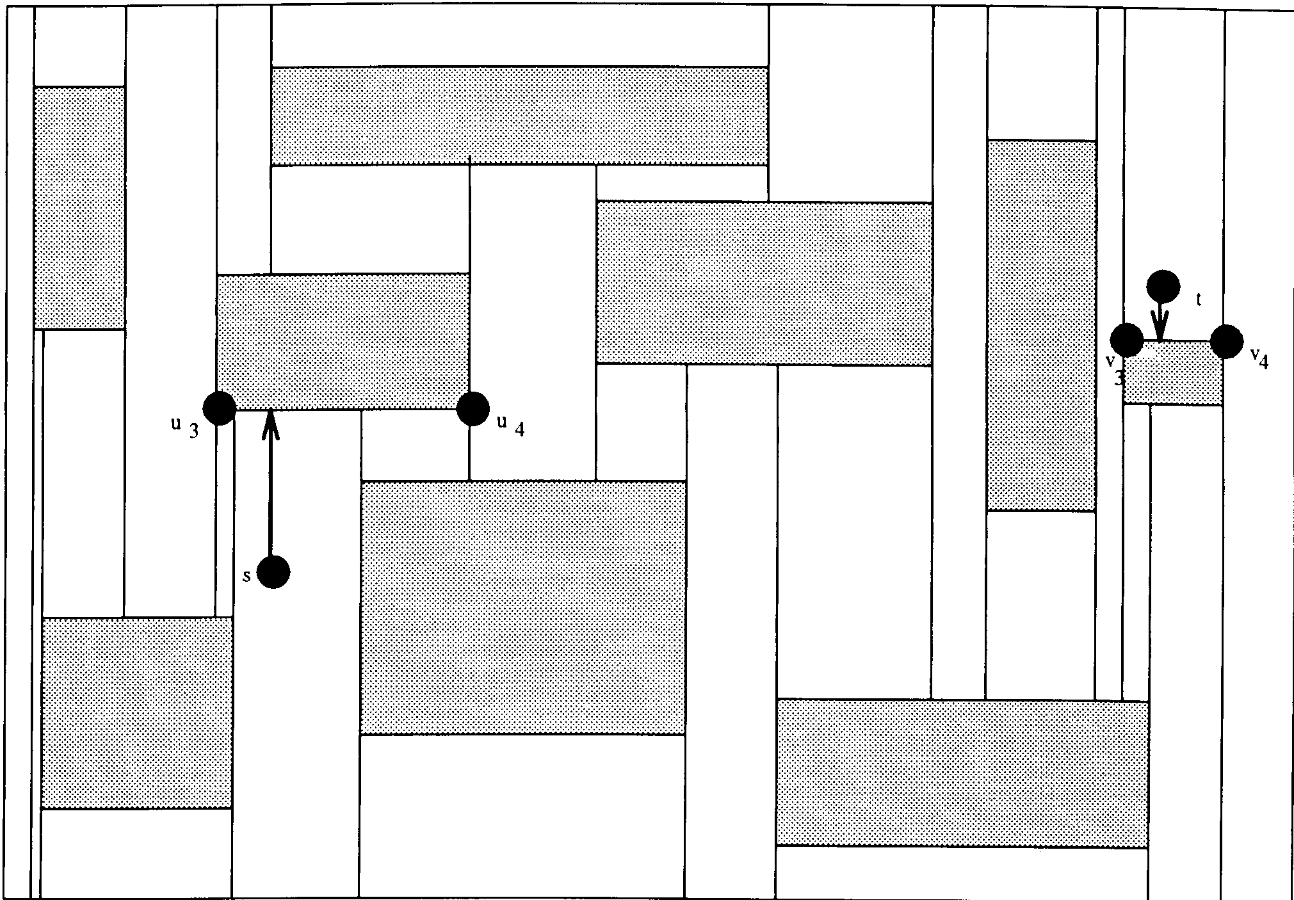


Figure 3.10: Corner Points in $\mathcal{R}$ Obtained by Vertical Ray Shooting from s & t

distance between a and b in G_{+y}.

If the minimum value is defined then the shortest distance between s and t is found. Otherwise, we proceed to the next step.

- **Step III.** The shortest distance between s and t equals $|s_x - t_x| + |s_y - t_y|$.

Lemma 3.11 *The above algorithm correctly determines the shortest distance between arbitrary origin and destination points.*

Proof : The assumption that t lies in the first quadrant of s leads to the fact that t lies in either R_1, R_2 or R_3 of s. If t lies in R_1 of s then the shortest paths from u_1 to v_1, from u_1 to v_2, from u_2 to v_1, and from u_2 to v_2 are maintained in G_{+x}. Also from the **Lemma 3.4** [12] we know that the shortest-path between between s and t goes through at least one corner point in $\{u_1, u_2\}$ and at least one corner point in $\{v_1, v_2\}$. Therefore the shortest distance will be computed at the termination of **step I**. If t lies in R_3 of s then the shortest paths from u_3 to v_3, from u_3 to v_4, from u_4 to v_3, and from u_4 to v_4 are maintained in G_{+y}. Therefore shortest distance will be computed at the termination of **step II**. Otherwise, if t belongs to R_2 of s and the shortest distance is determined by **step I, step II or step III**. The correctness of this claim follows from **Lemma 3.6.** ◫

The above mentioned hierarchy can be constructed within the same time and storage bounds of the preprocessing algorithms, of the previous section, to support $O(\log n)$ point location queries. Therefore, we can conclude that

Theorem 3.6 *The orthogonal shortest path between two arbitrary points can be determined by a single processor in*

(i) $O(\log n + k)$ *time with* $O(n^2)$ *preprocessing, or*

(ii) $O(\sqrt{n} + k)$ *time with* $O(n\sqrt{n})$ *preprocessing,*

where k is the number of segments in the constructed shortest path.

Chapter 4

Approximate Shortest-Path Algorithms

In this chapter we consider the problem of approximate rectilinear shortest-path query between two arbitrary points in the presence of n isothetic and disjoint rectangular obstacles. We present an algorithm that reports a path whose length is at most three times the optimal path length between two arbitrary points. Our algorithm takes $O(n \log^3 n)$ preprocessing time, $O(n \log^2 n)$ space and $O(\log n)$ query time for the distance problem. The actual path can be reported in $O(\log n + k)$ where k is the number of segments in the reported path. Thus here we exhibit a tradeoff in terms of time complexity and optimality with the the result in the previous chapter where an exact solution of this query problem is given at the expense of $O(n\sqrt{n})$ preprocessing and $O(\sqrt{n}+k)$ query time or using $O(n^2)$ preprocessing and $O(\log n + k)$ query time.

Many times in motion planning problems instead of obtaining an optimal path, researchers have computed nearly optimal paths that can be obtained by spending much less time. This issue was addressed by Clarkson in [7]. In this chapter we present an algorithm to solve the query problem for the approximate shortest path between two arbitrary points avoiding the rectangles in $\mathcal{R}$. The algorithm reports a path whose length is at most three times the optimal path length between two arbitrary corner points and at most three times the optimal path length between two

arbitrary points. For this we spend $O(n \log^3 n)$ preprocessing to answer the approximate shortest-distance query in $O(\log n)$ time. The actual path can be reported in $O(\log n + k)$ time where k is the size or the total number of segments in the reported path. Our approach for solving this problem uses the staircase separator of Atallah and Chen [2] which can produce a balanced splitting of the set of rectangles in R, and Voronoi diagram computation on the sparse visibility graph, introduced in Clarkson et al., [8], to solve the rectilinear shortest-path problem in the presence of obstacles.

Our overall organization of the chapter is as follows. In **section 4.1** we review existing results on the rectilinear shortest-path problem, which we will be using later. In **section 4.2.1** we present an algorithm to answer the approximate shortest-distance query between two arbitrary corner points of $\mathcal{R}$ in $O(\log n)$ time which uses $O(n \log^3 n)$ preprocessing and $O(n \log^2 n)$ space. Subsequently in **section 4.2.2** we relax the corner point restriction. Finally in **section 4.3** we consider the case of vertical line segment obstacles and in that section we present a preprocessing algorithm on the set of n vertical line segments in $O(n \log^2 n)$ time and $O(n \log n)$ space to answer the approximate shortest-distance query between two arbitrary query points in $O(\log n)$ time.

4.1 A Brief Review of Some Existing Results

For the sake of completeness in this section we will briefly introduce the following two structures from existing literature.

- **Staircase Separator** : The notion of the staircase separator was introduced in [2]. The staircase separator S of $\mathcal{R}$ is an orthogonal chain avoiding the set of obstacles in $\mathcal{R}$. S has $O(n)$ segments and is monotone along both x and y directions. S splits $\mathcal{R}$ to two subsets R' and R'' in such a way that both $|R'| \leq 7n/8$ and $|R''| \leq 7n/8$. Also S can be constructed in $O(n \log n)$ time. The existence of S greatly facilitates the design of divide-and-conquer type algorithms for problems on rectangles, since we can obtain a balanced decomposition.

- **Sparse Visibility Graphs :** This graph was introduced in [8]. Given $\mathcal{R}$ we can maintain the shortest-path information between any two corner points $h_i, h_j \in \mathcal{H}$ in this graph. This graph $G = (V, E)$ is defined as follows :

  - **I.** Each corner point of $\mathcal{R}$ is joined to all four shadow points. Please refer to figure 4.1 for the illustration. In the illustration p_m and p_n are two shadow points of v_k. Let us denote the set of all shadow points by $\mathcal{P}$. There is an edge between each point and its shadow points. Let us denote this set of edges by E_1. For each obstacle edge $e_i = (v_i, v_j)$ if projected shadows are $p_l, p_{l+1}, \ldots, p_m \in \mathcal{P}$ in order then following edges $(v_i, p_l), (p_l, p_{l+1}), \ldots, (p_m, v_j)$ are included in E. Here we note that if there is no shadow point on an edge e_i then we include that edge in the graph. Let us denote the set of all edges of this type by E_2.

  - **II.** We take a vertical line which passes through a point which has the median value x_m with respect to x coordinates of corner points. We join an edge between each point v_i to the point $v_i' = (x_m, v_{i_y})$ if it is visible from v_i. Let $v_l', \ldots, v_m'$ be those Steiner points in sorted order along y directions. Then we join every two adjacent points in that order by an edge if they are mutually visible. Please refer to figure 4.2 for this construction. We apply this construction recursively for all points with the value of x coordinate less than x_m and separately for all points with x coordinate values greater than x_m. The same construction is repeated by horizontal splittings. Let E_3 denote the set of edges added by this recursive construction both in vertical and horizontal directions. Also let $\mathcal{Q}$ denote the set of Steiner points introduced by this construction. Here $|\mathcal{Q}| \in O(n \log n)$.

So in $G = (V, E)$ we have

- $V = \mathcal{H} \cup \mathcal{P} \cup \mathcal{Q}$.
- $E = E_1 \cup E_2 \cup E_3$.

Each edge of the graph is associated with an weight that is the L_1 distance between its two endpoints. This graph can be computed in $O(n \log^2 n)$ time; the algorithm is described in [8]. Between any pair of corner points v_i, v_j the shortest path will be maintained in G, i.e., a subgraph of G.

Here we would like to note that our carrier graph has fewer vertices and edges than the sparse visibility graph of [8]. The shortest path between every pair of h_i and h_j is always maintained in the sparse visibility graph, but that is not so in case of carrier graphs. For this reason we have to also use the sparse visibility graph along with the carrier graphs in the case of rectangles. But for the case of vertical line segments we only need the carrier graphs.

4.2 Query Between Two Arbitrary Points

Let s and t denote the two query points. Without loss of generality we will assume that t lies in the first quadrant of s, i.e., in other words $s_x < t_x$ and $s_y < t_y$. For the first part of our discussion we will concentrate on the approximate shortest-path query between two arbitrary corner points of $\mathcal{R}$. Subsequently we will relax this restriction. This is done by reducing the problem of query between two arbitrary points to a query constrained between corner vertices of rectangles. The exact procedure is described in more detail in the later part of the section. Let $\mathcal{Z}$ be a bounding rectilinear and rectilinearly convex polygon around $\mathcal{R}$ whose size is $O(|R|)$.

4.2.1 Query Between Two Arbitrary Corner Points

Now let R' and R'' denote the two subsets of $\mathcal{R}$ after being split by the staircase separator S. Let v_i and v_j be two arbitrary corner points belonging to R' and R'' respectively. Let $\mathcal{Z}$ denote the rectilinear and rectilinearly convex enclosing polygon of the set $\mathcal{R}$ with $O(n)$ vertices. The following property which holds for the shortest path between v_i and v_j have been used in [2].

Lemma 4.1 *There exists a shortest path between v_i and v_j whose intersection with*

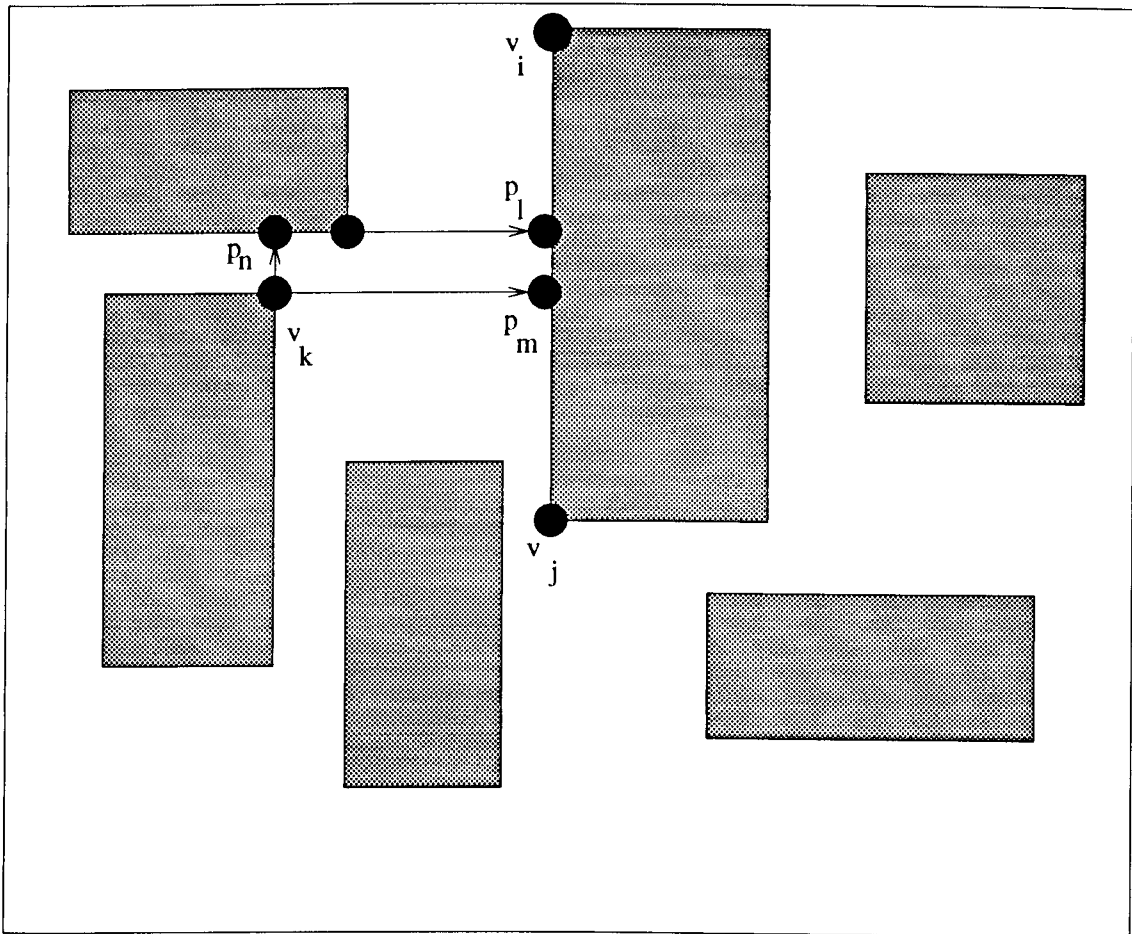


Figure 4.1: Sparse Visibility Graph Construction-I

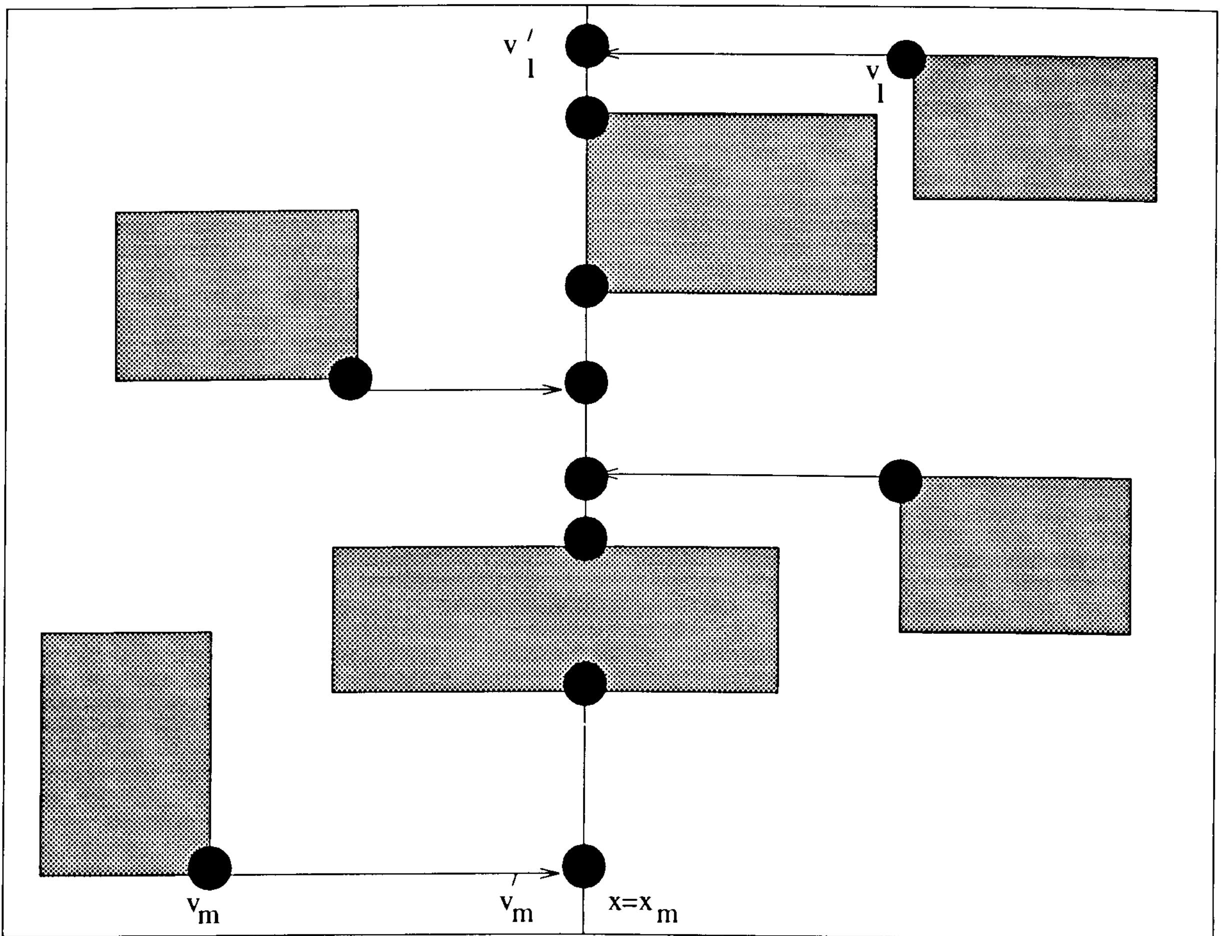


Figure 4.2: Sparse Visibility Graph Construction-II

the staircase S must contain at least one of the horizontal or vertical projection of the corner points of $R' \cup R''$ on S or the two intersection points of S with $\mathcal{Z}$.

Preprocessing Phase of the Algorithm

The preprocessing algorithm consists of building a search tree whose every node maintains the approximate shortest-path information between corner points of $\mathcal{R}$. We will name this tree the approximate shortest path search tree, or **ASPST**. Now we give a high-level description of the preprocessing phase of our algorithm.

Algorithm Preprocess :

Comments : The results of the computation of the first three steps are stored at the current node of **ASPST**. Both subtrees, built up recursively in the last step, are maintained as children of the current node of **ASPST**.

- **Step I.** Compute the staircase separator S, which gives a balanced decomposition of $\mathcal{R}$ into two subsets R' and R'' as described in [2].

- **Step II.** Project each corner point of R' and R'', both horizontally and vertically onto S, if the line joining the projected point and the corresponding corner point doesn't intersect any other obstacle or the enclosing polygon. Let $\mathcal{T} = \{t_1, t_2, \ldots, t_p\}$ denote the set of all such projected points along with the two intersection points of S with the enclosing polygon. This set can be computed by the trapezoidation algorithm using the plane sweep as described in [45].

- **Step III.** Compute the Geodesic Voronoi Diagram of $\mathcal{T}$ on $\mathcal{C}$, the set of corner points in $\mathcal{R}$. More formally we partition $\mathcal{C}$ into subsets $V_1, V_2, \ldots, V_p$ such that $\bigcup_{i=1}^{p} V_i = \mathcal{C}$ and for each corner point $v_j \in V_i$, t_i is geodesically nearer to it compared to $t_j \neq t_i$. The detailed algorithm to compute this efficiently will be given later in this section.

- **Step IV.** Recursively build up **ASPST** on both R' and R''. □

Now let us consider two arbitrary corner points of $\mathcal{R}$, say v_i and v_j, which lie on two opposite sides of the staircase S. Let t_k and t_l belonging to $\mathcal{T}$ be the geodesically nearest point of v_i and v_j respectively. Now we consider the following path :

$Q = sp(v_i, t_k) \cup sp(t_k, t_l) \cup sp(t_l, v_j)$

Here we note that $sp(t_k, t_l)$ is taken along the staircase S. Then the following property holds on the path Q.

Theorem 4.1 *The length of the path Q between v_i and v_j is at most three times the length of the shortest path $sp(v_i, v_j)$.*

Proof : Please refer to figure 4.3 & figure 4.4 for illustrations. From **Lemma 4.1** we know that there always exists a shortest path whose intersection with $\mathcal{T}$ is nonempty. Let $sp(v_i, v_j)$ be the shortest path having this property. Let $t_i \in \mathcal{T}$ be one of the points in $sp(v_i, v_j) \cap \mathcal{T}$. There are a few cases to be considered here. In one case t_k is above t_l and in another case t_k is below t_l. But since the same argument holds in both cases we will elaborate on the first case only. Again three possible cases can arise when the first condition holds.

- **Case I.** t_i is beteween t_k and t_l.
- **Case II.** t_i is above t_k.
- **Case III.** t_i is below t_l.

Since **Case III** and **Case II** are symmetrical we will argue for the first two cases only.

- **Case I:** Here let us consider the ratio between $sd(v_i, t_k) + sd(t_k, t_i)$ to $sd(v_i, t_i)$. To design the worst case situation we have to make $sd(t_k, t_i)$ as large as possible. Now we claim that $sd(t_k, t_i) \leq 2 \times sd(v_i, t_i)$. This follows from subsequent arguments.

  From the triangle inequality of L_1 metric we have :-

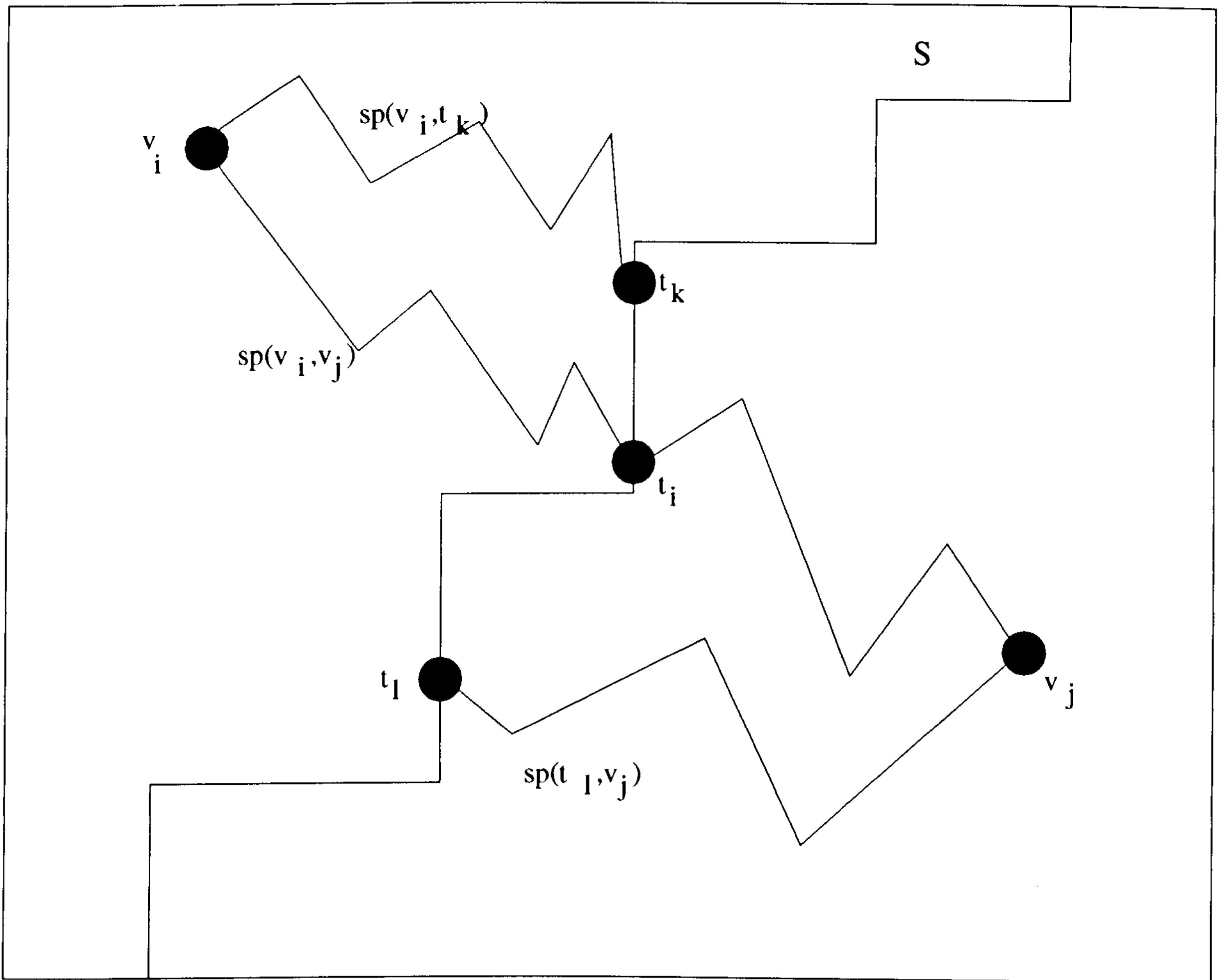


Figure 4.3: Case I - t_i Is Between t_k and t_l

$sd(t_k, t_i) = L_1(t_k, t_i) \leq L_1(v_i, t_k) + L_1(v_i, t_i)$
$\leq sd(v_i, t_k) + sd(v_i, t_i) \leq sd(v_i, t_i) + sd(v_i, t_i)$

The last inequality holds because t_k is the geodesic Voronoi neighbour of v_i in the set $\mathcal{T}$ and therefore $sd(v_i, t_k) \leq sd(v_i, t_i)$.

Thus we have $sd(t_k, t_i) \leq 2 \times sd(v_i, t_i)$.

Therefore the total distance traversed is equal to :-

$$sd(v_i, t_k) + sd(t_k, t_i) \leq sd(v_i, t_i) + 2 \times sd(v_i, t_i) = 3 \times sd(v_i, t_i) \tag{4.1}$$

Following exactly similar argument we obtain

$$sd(v_j, t_l) + sd(t_l, t_i) \leq 3 \times sd(v_j, t_i) \tag{4.2}$$

Therefore adding (1) and (2) we obtain :-

$sd(v_i, t_k) + sd(t_k, t_l) + sd(t_l, v_j) \leq 3 \times (sd(v_i, t_i) + sd(t_i, v_j)) = 3 \times sd(v_i, v_j)$

- **Case II:** Following the same argument of Case I we claim that

$sd(v_j, t_l) + sd(t_l, t_i) \leq 3 \times sd(v_j, t_i)$

$$\Rightarrow sd(v_j, t_l) + sd(t_l, t_k) \leq 3 \times sd(v_j, t_i) \tag{4.3}$$

since $sd(t_l, t_k) \leq sd(t_l, t_i)$.

Again t_k is the geodesic Voronoi neighbour of v_i. Therefore

$$sd(v_i, t_k) \leq sd(v_i, t_i) \tag{4.4}$$

Combining (3) and (4) we have

$sd(v_i, t_k) + sd(t_l, t_k) + sd(v_j, t_l) \leq sd(v_i, t_i) + 3 \times sd(v_j, t_i) \leq 3 \times (sd(v_j, t_i) + sd(t_i, v_i)) = 3 \times sd(v_i, v_j)$

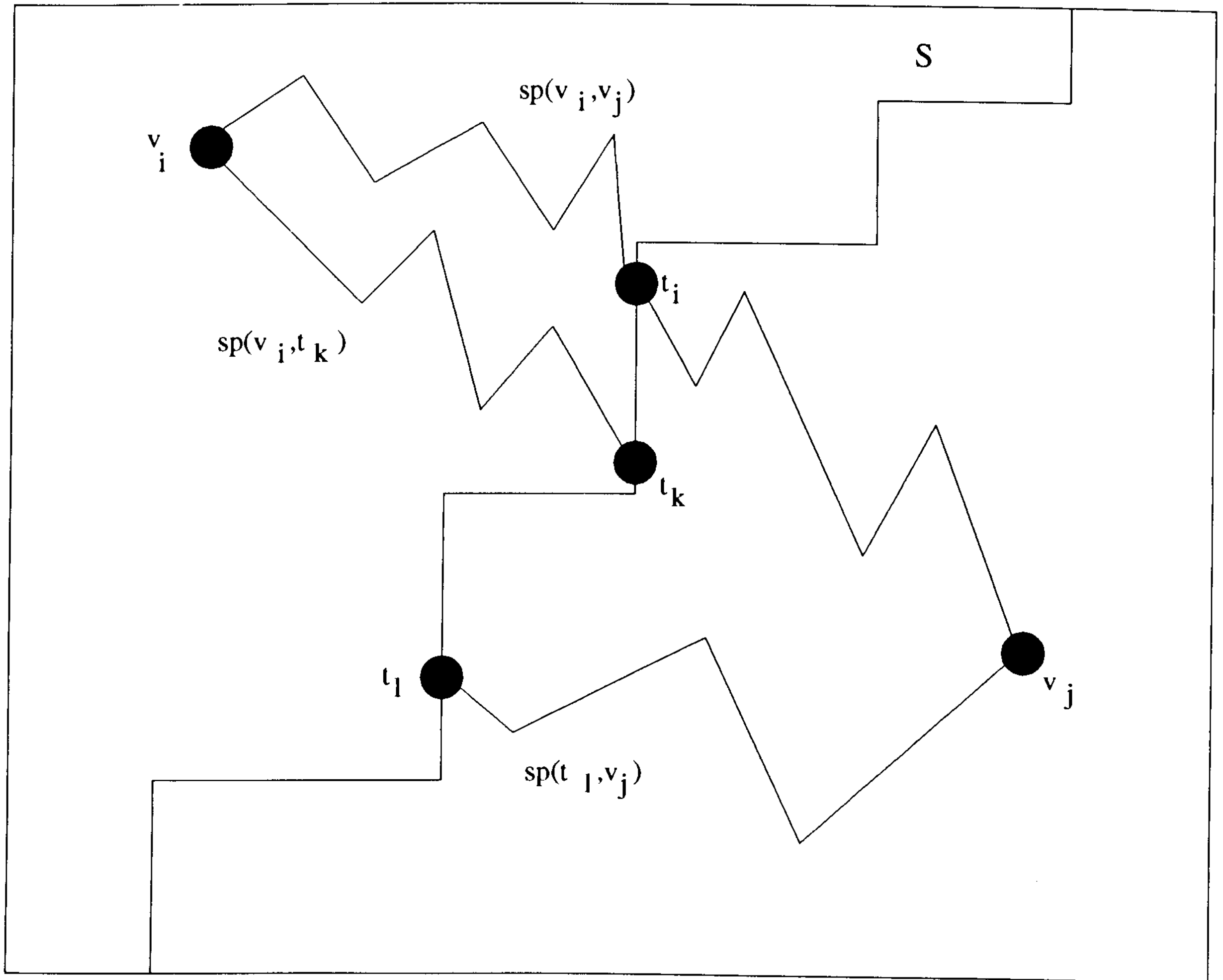


Figure 4.4: Case II - t_i is above t_k

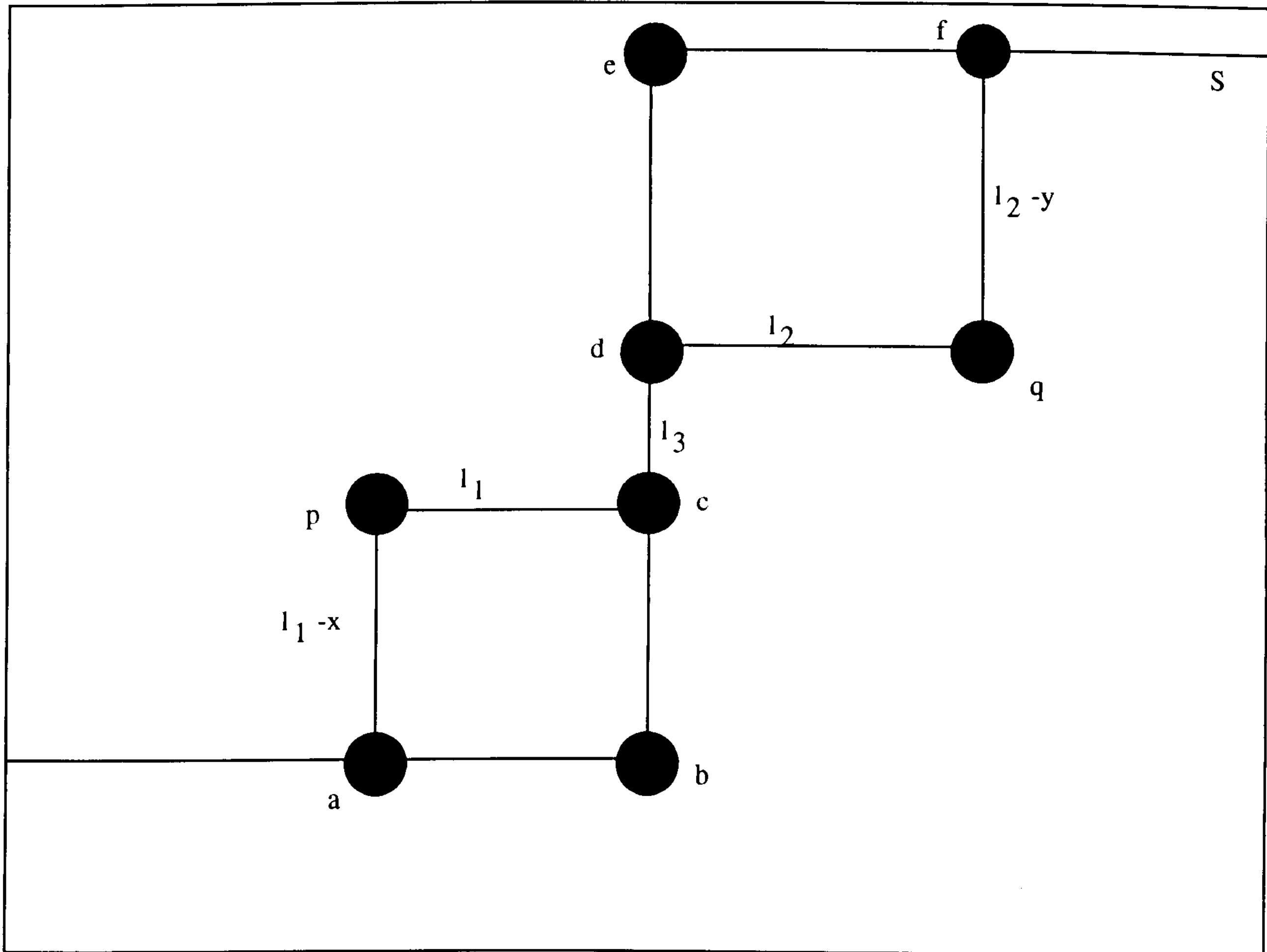


Figure 4.5: A Case in which the Constant of the Approximation For Rectangular Obstacles is Tight

□

So from the above theorem we can see that Q is a near optimal path between v_i and v_j. To show that the ratio of approximation is tight, i.e., the ratio of the reported path to the optimal path can be three in some cases, we consider the illustration in figure 4.5. Let p and q denote two corner points of rectangles in $\mathcal{R}$ and we assume the shortest path between them is monotone in both directions. Let S denote the staircase separator. Let a and c denote the vertical and horizontal projections of p on S. Similarly let d and f denote the horizontal and vertical projections of q on S. The shortest distance between p and q is $l_1 + l_3 + l_2$. But a being nearer to p than c and f being nearer to q than d the length of the reported path $pabcdefq$ is $(l_1 - x) + l_1 + (l_1 - x) + l_3 + (l_2 - y) + l_2 + (l_2 - y)$ which is $3l_1 + 3l_2 + l_3 - 2x - 2y$. So in the case when l_1 and l_2 is much larger than l_3, x and y the ratio of the reported path to the optimal path can go as close to three as possible. This example shows that the constant of our approximation using this technique is tight.

Now we will describe how to compute the **Step III** of the algorithm **Preprocess** efficiently. For that purpose we will make use of the result in [35] which computes the Voronoi partition of a subset of vertices in a graph. More formally given a weighted graph $G = (V, E)$ and $V' \subset V$ where $|V'| = k$, we want to partition V into subsets $V_1, V_2, \ldots, V_k$ such that $\bigcup_{i=1}^{k} V_i = V$ and $v_i \in V'$ is nearer, to all vertices of V_i in G, compared to other $v_j \in V'$, $j \neq i$. In all cases we will assume edge weights to be non-negative. Here we briefly describe the method of [35] to compute this partition.

Lemma 4.2 *Given a weighted graph $G = (V, E)$ and $V' \subset V$ we can compute the above mentioned Voronoi partition of V in $O(|E| \log |V|)$ time.*

The technique used here is to add a new vertex s and to join it to each vertex $v_i \in V'$ with edges of weight 0. Now we run the single-source shortest-path algorithm from s in this transformed graph. Now in the shortest-path tree of s, all vertices hanging in the subtree rooted at $v_i \in V'$ will be put into the set V_i. From the construction it directly follows that any vertex in V_i is nearer to v_i than any other $v_j \in V'$ where $j \neq i$. The desired complexity can be achieved using the heap implementation of [14] for the single source shortest-path algorithm.

Lemma 4.3 *The* **step III** *of the algorithm* **Preprocess** *can be computed in* $O(n \log^2 n)$ *time at the first level of recursion.*

Proof : The Voronoi diagram of $\mathcal{T}$ on $\mathcal{H}$ can be computed in the following way :

- **Step I.** Compute the sparse visibility graph G_1 for $\mathcal{T} \cup R'$ and G_2 for $\mathcal{T} \cup R''$.
- **Step II.** Apply the algorithm of [35] described in **Lemma 4.2** to both G_1 and G_2 respectively with all vertices of $\mathcal{T}$ as the set V'.

The computation of **step I** requires $O(n \log^2 n)$ time [8]. Since $|V| = O(n \log n)$ and $|E| = O(n \log n)$ for the sparse visibility graph G **step II** can be carried out in $O(n \log^2 n)$ time from **Lemma 4.2.** □

Theorem 4.2 *The algorithm* **Preprocess** *requires* $O(n \log^3 n)$ *time and* $O(n \log^2 n)$ *space.*

Proof : The time required to compute the staircase separator in **step I** is $O(n \log n)$ [2]. To compute $\mathcal{T}$ using trapezoidation will take $O(n \log n)$ time. Therefore using **Lemma 4.3** we obtain the following recurrence relation for the overall time complexity $T(n)$, for the algorithm **Preprocess** as :

$T(n) \leq T(\alpha n) + T((1 - \alpha)n) + c * n \log^2 n$, $1/8 \leq \alpha \leq 7/8$ and c is a positive const.

Therefore, $T(n) \in O(n \log^3 n)$. At each level of recursion to store the complete approximate shortest-path information the space required is $O(n \log n)$. Therefore the total space required is $O(n \log^2 n)$. □

Algorithm to answer the query

In this section we will describe how to answer the approximate shortest-distance query between two arbitrary corner points. Let v_i and v_j be the two query points. We use the following procedure to answer the approximate shortest-distance query between v_i and v_j.

Algorithm Query :

Input : v_i, v_j; **Output :** $asd(v_i, v_j)$;

- **Step I.** Using binary search over the staircase S stored in root of **ASPST**, determine if v_i and v_j lie on the opposite or the same side of S or on S.

- **Step II.** If v_i and v_j lie on opposite sides of S or if at least one of v_i or v_j lie on S compute the geodesic Voronoi neighbours of v_i and v_j from the **ASPST** built during the preprocessing step. Let t_k and t_l respectively be the geodesic Voronoi neighbours of v_i and v_j. We report $sd(v_i, t_k) + sd(t_k, t_l) + sd(t_l, v_j)$ as the approximate shortest distance between v_i and v_j, i.e., $asd(v_i, v_j)$ and **stop**. Otherwise if v_i and v_j lie on the same side of S then go to **step III**.

- **Step III.** Go to the corresponding child node in the **ASPST** tree, depending on which side of S v_i and v_j are, and repeat from **step I**.

Lemma 4.4 *Algorithm* **Query** *reports a distance which is at most three times the optimal path length between v_i and v_j in $O(\log^2 n)$ time.*

Proof : The proof of the fact that the distance reported by the algorithm **Query** is at most three times the optimal path length directly follows from **Theorem 4.1**.

To carry out a binary search in **step I** requires $O(\log n)$ time. **Step II** can be computed in $O(1)$ time. Therefore the worst case running time $Q(n)$ of the algorithm **Query** follows from the following recurrence :-

$$Q(n) \leq Q(7n/8) + c * \log n, \text{ where } c \text{ is a positive constant.}$$

Therefore $Q(n) \in \log^2 n$. □

Thus we have established the following theorem.

Theorem 4.3 *Between two arbitrary corner points an approximate orthogonal shortest distance through a path avoiding $\mathcal{R}$ whose length is at most three times the optimal length can be obtained in $O(\log^2 n)$ query time using $O(n \log^3 n)$ preprocessing and $O(n \log^2 n)$ space.*

To improve the query time we have to speed up the search for the staircase separator in **ASPST** such that v_i and v_j lie on the opposite sides. Our approach here is a bottom up search instead of a top-down search as described in algorithm **Query**. To do this we consider all staircase separators collectively and perform planar point location preprocessing using the algorithm of [15]. With this preprocessing we can identify the two regions corresponding to the two leaf nodes of **ASPST** in which v_i and v_j lies in $O(\log n)$ time. Subsequently we identify the lowest common ancestor of those two leaf nodes in **ASPST**. Since the height of **ASPST** is $O(\log n)$ this step will take $O(\log n)$ time. Let t_k and t_l be the geodesic Voronoi neighbours of v_i and v_j on the staircase separator stored in the lowest common ancestor. Then $sd(v_i, t_k) + sd(t_k, t_l) + sd(t_l, v_j)$ is reported as $asd(v_i, v_j)$. Thus we have established the following theorem.

Theorem 4.4 *Between two arbitrary corner points an approximate orthogonal shortest distance through a path avoiding $\mathcal{R}$ whose length is at most three times the optimal length can be obtained in $O(\log n)$ query time using $O(n \log^3 n)$ preprocessing and $O(n \log^2 n)$ space.*

4.2.2 Removal of the Corner Point Restriction

In this section we will relax the restriction of the two query points being two corner points in the set $\mathcal{H}$. Let those two query points be s and t respectively.

Here we face a problem to answer the reachability query between two arbitrary corner points efficiently with less preprocessing. By reachability we mean if there is a path between two corner points in the carrier graphs. This statement would be more clear once we will look into the details of the algorithm **General-Query** and its correctness proof in **Lemma 4.6**. We will use the carrier graphs G_{+x}, G_{+y}, G_{-y} for this problem. The goal here is to use these carrier graphs to answer the reachability query without computing the transitive closure on them. For this we will make use of some properties that hold on any planar st-graph, G. Any planar st-graph is a planar directed acyclic graph with exactly one source s and one sink t and when embedded in the plane both s and t should lie on the boundary of the external face. The following

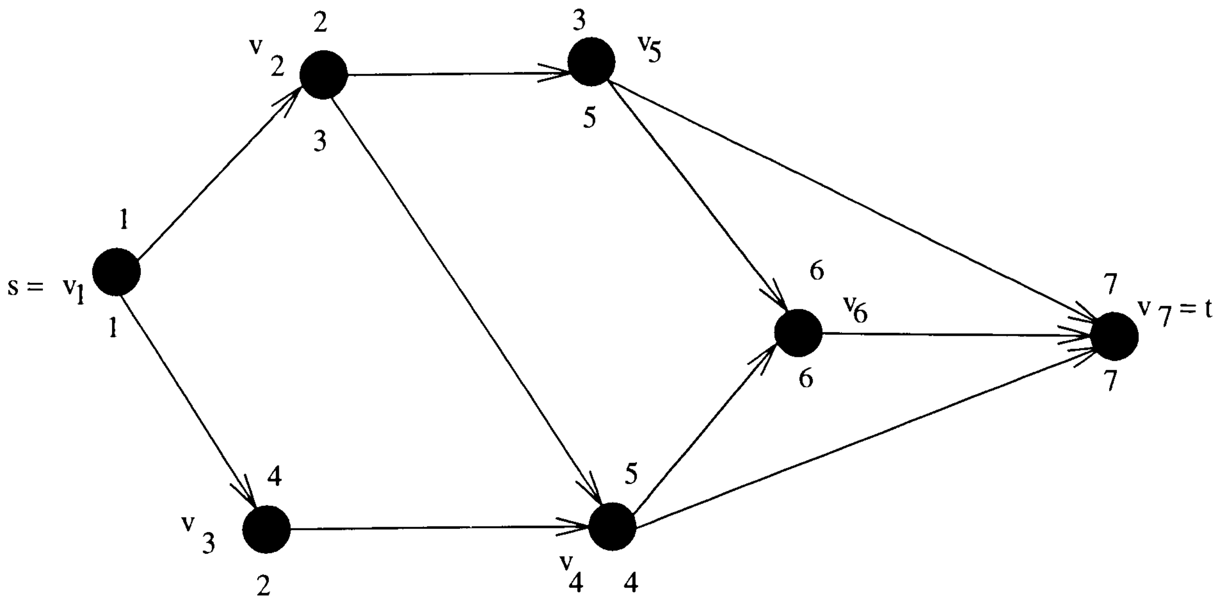


Figure 4.6: Illustration of a st-graph and its $<_L$ and $<_R$ order

result will be particularly useful in our algorithm.

Lemma 4.5 [29], [46] *Let G be a planar st-graph with n vertices. There exist two total orders on the vertices of G, denoted $<_L$ and $<_R$, such that there is a directed path from u to v if and only if $u <_L v$ and $u <_R v$. Furthermore, the orders $<_L$ and $<_R$ can be computed in $O(n)$ time.*

These two orderings are illustrated in figure 4.6. Both of them are topological ordering on the planar DAG G along with the ordering of vertices done from left to right (or right to left) order of the planar embedding. In figure 4.6 we see that the labels associated with vertices v_2 and v_6 are (2,3) and (6,6) respectively. Thus according to **Lemma 4.5** there should be path from v_2 to v_6 which is indeed the situation there. Also we see that the v_3 and v_5 have labels (4,2), (3,5) respectively. Thus according to **Lemma 4.5** they are not reachable from one another and this also holds in our example.

We can easily verify that all our G_{+x}, G_{+y} and G_{-y} are planar st-graphs. So we preprocess each of them according to **Lemma 4.5** in $O(n)$ time using the algorithm of [29] to answer the reachability problem between two arbitrary query vertices in $O(1)$ time. Here we note that since we assume $s_x < t_x$ and $s_y < t_y$, we will use only G_{+x} and G_{+y}.

Using the algorithm of [31] or [15] we preprocess the planar subdivisions formed by the horizontal and vertical trapezoidations of $\mathcal{R}$ for ray shooting queries along horizontal and vertical directions respectively.

All these preprocessing can be done within the same space and time bound of the preprocessing step of **Theorem 4.4**.

We use the following steps to determine the approximate shortest distance between s and t.

Algorithm General-Query :

Comments : Please refer to figure 4.7 for the illustration.

- **Step I.** Shoot a horizontal ray towards right from s. Let u_1 and u_2 be two vertices on the edge of the rectangle hit by the ray. From t we shoot another

horizontal ray towards left and let v_1 and v_2 be two vertices on the edge of the rectangle hit by the ray. If v_j, $j \in \{1,2\}$ is not reachable from u_i, $i \in \{1,2\}$ in G_{+x} then go to **step II** else compute $asd(s,t)$ as

$asd_x(s,t) = min\{L_1(s,u_i) + asd(u_i,v_j) + L_1(v_j,t)\}$ where $i \in \{1,2\}$ and $j \in \{1,2\}$.

In the above expression $asd(u_i,v_j)$, where $i \in \{1,2\}$ and $j \in \{1,2\}$, is computed using the algorithm described in the previous section. **Stop**.

- **Step II.** Shoot a vertical ray up from s. Let u_3 and u_4 be two vertices on the edge of the rectangle hit by the ray. Shoot another vertical ray downwards from t. Let v_3 and v_4 denote the corresponding corner points. If v_j, $j \in \{3,4\}$ is not reachable from u_i, $i \in \{3,4\}$ in G_{+y} then go to **step III** else compute $asd(s,t)$ as

  $asd_y(s,t) = min\{L_1(s,u_i) + asd(u_i,v_j) + L_1(v_j,t)\}$ where $i \in \{3,4\}$ and $j \in \{3,4\}$.

  In the above expression $asd(u_i,v_j)$, where $i \in \{3,4\}$ and $j \in \{3,4\}$, is computed using the algorithm described in the previous section. **Stop.**

- **Step III.** Compute $asd(s,t) = |t_x - s_x| + |t_y - s_y|$.

Lemma 4.6 *The algorithm* **General-Query** *correctly computes the approximate shortest distance between s and t, which is at most three times the optimal distance, and the time complexity of the algorithm is* $O(\log n)$.

Proof : Since t lies in the first quadrant of s the shortest path between them can be of three types from the result of [12] :

- **Case I.** The shortest path is monotone only in $+x$ direction.
- **Case II.** The shortest path is monotone only in $+y$ direction.

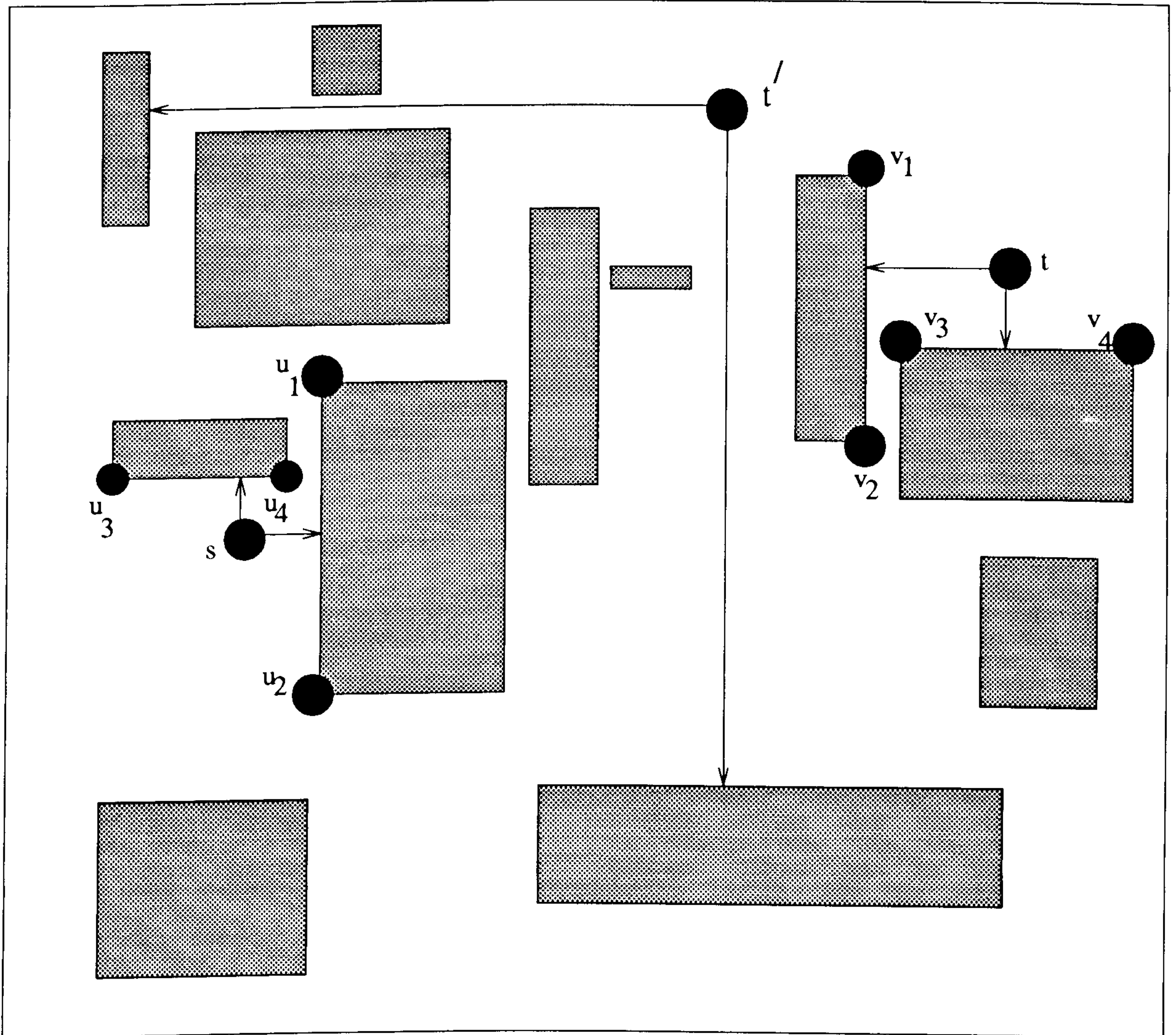


Figure 4.7: Illustrating the Ray-Shooting Procedure

- **Case III.** The shortest path is monotone in both $+x$ and $+y$ directions.

In **Case I** from **Lemma 3.4** [12] we know that the shortest path will start from s and will either go through u_1 or u_2 and will enter t through either v_1 or v_2. Also in this situation v_j, $j \in \{1,2\}$ is always reachable from u_i, $i \in \{1,2\}$ in G_{+x} from **Lemma 3.5**. **Step I** of the algorithm **General-Query** exhaustively computes all four types of paths. Therefore the minimum of all four types of paths computed in **Step I** must be at most three times the shortest path between s and t in **Case I**. Also if **Case II** holds then v_j, $j \in \{1,2\}$ will be unreachable from u_i, $i \in \{1,2\}$ in G_{+x} and the algorithm will advance to **Step II**.

Similarly if **Case II** holds the shortest path will start from s and will pass through either of u_3 or u_4 and will enter t through either v_3 or v_4. Also in this case v_j, $j \in \{3,4\}$ will be reachable from u_i, $i \in \{3,4\}$ in G_{+y}. Therefore the minimum of all four shortest paths computed in **Step II** will be atmost three times the shortest path between s and t in this case.

If **Case III** holds the shortest path between s and t will be computed in **Step I** or **Step II** respectively in case v_j, $j \in \{1,2\}$ is reachable from u_i, $i \in \{1,2\}$ in G_{+x} or v_j, $j \in \{3,4\}$ is reachable from u_i, $i \in \{3,4\}$ in G_{+y} and the correctness in that case follows from **Lemma 3.6** and **Lemma 3.2**. Otherwise if the reachability condition is not satisfied in either of the two cases as illustrated in figure 4.7 between s and t', then we know **Case III** definitely holds and we get the exact shortest distance between s and t in **Step III**.

The time required by the algorithm **General-Query** is $O(\log n)$ for the point location query and $O(1)$ for the reachability query between corner points, which follows from **Lemma 4.5**. Also $O(\log n)$ time for each approximate shortest distance computation between corner points in **Step I** and **Step II** from **Theorem 4.4**. Therefore the overall time complexity of the algorithm **General-Query** is $O(\log n)$. □

Here we note that the actual path between s and t can be reported in $O(\log n + k)$ time where k is the number of segments in the reported path. This follows from the fact that if at least one among $asd_x(s,t)$ or $asd_y(s,t)$ is not equal to infinity then the actual path can be reported from the approximate shortest-path information

maintained in the preprocessing phase in **ASPST**. Otherwise if both $asd_x(s,t) = asd_y(s,t) = +\infty$ then we already know that the shortest path between between s and t is monotone in both $+x$ and $+y$ directions. In that case the exact shortest path can be reported by growing two monotone staircases, one C_3 type staircase from s and another C_6 type staircase from t. The existence of intersection between these two types of paths follows from [12] because the shortest path between s and t is monotone in both $+x$ and $+y$ directions. Also these two staircase paths are readily available in the sparse visibility graphs or carrier graphs computed in the preprocessing phase.

Thus we have established the following theorem.

Theorem 4.5 *Between two arbitrary query points an approximate orthogonal shortest distance through a path whose length is at most three times the optimal length can be obtained in $O(\log n)$ query time using $O(n \log^3 n)$ preprocessing and $O(n \log^2 n)$ space. The actual path can be reported in $O(\log n + k)$ time where k is the size of the output path.*

4.3 Vertical Line Segment Obstacles

In this section we will show how to improve on the preprocessing time in the special case of n vertical line segment obstacles, $\mathcal{B}$. Now we will describe the modified preprocessing algorithm for $\mathcal{B}$. Again our goal here is to precompute the approximate shortest-path search tree or **ASPST**. The algorithm is as follows.

Algorithm Preprocess :

Comments : To avoid repeated use of median finding algorithm we can sort all line segments in $\mathcal{B}$ by their $x-$coordinates. The results of the computations of the first three steps are stored at the current node of **ASPST**.

- **Step I.** Find out the vertical line S with the median value of $x-$coordinate among the line segments in $\mathcal{B}$. Let B_1 denote the set of line segments in $\mathcal{B}$ to the left of S and B_2 denote the set of line segments in $\mathcal{B}$ to the right of S.

- **Step II.** Project each cornerpoint of the line segments in $\mathcal{B}$ horizontally on S, if the line joining the projected point and the corresponding corner point doesn't intersect any other obstacles in $\mathcal{B}$. Let $\mathcal{T} = \{t_1, t_2, \ldots, t_p\}$ denote the set of all such projected points. This set can be computed by the trapezoidation algorithm using the plane sweep as described in [45].

- **Step III.** Compute the Voronoi partition of $\mathcal{T}$ in the set $\mathcal{H}$. In other words we partition $\mathcal{C}$ into subsets $V_1, V_2, \ldots, V_p$ such that $\cup_{i=1}^{p} V_i = \mathcal{C}$ and for each corner point $v_j \in V_i$, t_i is geodesically nearer to it compared to any $v_j \neq v_i$. The details of this step will be described later.

- **Step IV.** Recursively build up **ASPST** on both B_1 and B_2. □

For the computation in **step III** in the above algorithm we will use the carrier graph instead of the sparse-visibility-graph. The details are as follows.

- Compute the carrier graph G_{-x} for $\mathcal{T} \cup B_1$ and G_{+x} for $\mathcal{T} \cup B_2$.

- Apply the algorithm of [35] as described in **Lemma 4.2** to both G_{-x} and G_{+x} with all vertices in $\mathcal{T}$ as the set V'.

The correctness of the above algorithm is ensured from the following lemma.

Lemma 4.7 *For each vertex $t_i \in \mathcal{T}$, the shortest paths to all the corner points of B_2 are maintained in G_{+x}.*

Proof : The proof of this lemma is similar to the proof of **Lemma 3.5**. But the only difference is that C_3 and C_8 which are maintained in G_{+x}, in this case will play the role of C_2 and C_1 of the corresponding lemma. The rest of the arguments are identical. □

Lemma 4.8 *The* **step III** *of the algorithm* **Preprocess** *can be computed in $O(n \log n)$ time at the first level of recursion.*

Proof : The computation of the carrier graph can be carried in $O(n \log n)$ time as mentioned in **Theorem 3.2**. Subsequently for the shortest path computation in **Lemma 4.2** on the carrier graph requires $O(n)$ time from **Lemma 3.8**. Thus the whole step can be carried out in $O(n \log n)$ time. ◫

Theorem 4.6 *The overall time and the space complexities of the algorithm* **Preprocess** *are* $O(n \log^2 n)$ *and* $O(n \log n)$ *repectively.*

Proof : The sorting of the endpoints of $\mathcal{B}$ by $x-$coordinate takes $O(n \log n)$ time. The computation of $\mathcal{T}$ and **step III** can be carried out in $O(n \log n)$ time from **Lemma 4.8**. Thus the overall time complexity $T(n)$, for the algorithm **Preprocess** comes from the following recurrence.

$$T(n) \leq 2 * T(n/2) + c * n \log n, \text{ where } c \text{ is a positive constant.}$$

Therefore, $T(n) \in O(n \log^2 n)$. At each level of recursion to store the approximate shortest-path information the space required is $O(n)$. Therefore the total space required is $O(n \log n)$. ◫

The algorithm to answer the query between two arbitrary corner points is as follows.

Algorithm Query :
Input : Two endpoints of $\mathcal{B}$, namely v_i, v_j; **Output :** $asd(v_i, v_j)$;

- **Step I.** Determine if the two points v_i and v_j lies on the same or on the opposite sides of the current median line S.

- **Step II.** If they lie on opposite sides of S or if at least one of v_i or v_j lie on S compute the geodesic Voronoi neighbours of v_i and v_j from **ASPST** built during the preprocessing step. Let t_k and t_l respectively be the geodesic Voronoi neighbours of v_i and v_j. We report $sd(v_i, t_k) + sd(t_k, t_l) + sd(t_l, v_j)$ as $asd(v_i, v_j)$ and **stop**. Otherwise if v_i and v_j lie on the same side of S then go to **step III**.

- **Step III.** Go to the corresponding child node in **ASPST**, depending on which side of S v_i and v_j are and repeat from **step I**.

Lemma 4.9 *Algorithm* **Query** *reports a distance which is at most three times the optimal path length between v_i and v_j in $O(\log n)$ time.*

Proof : The proof of the fact that the distance reported by the algorithm **Query** is at most three times the optimal path length follows from a similar argument of **Theorem 4.1**.

Since each step in the algorithm **Query** requires $O(1)$ time the total query time $Q(n)$ follows from the following recurrence.

$$Q(n) \leq Q(n/2) + c, \text{ where c is a positive constant.}$$

Therefore $Q(n) \in O(\log n)$. □

Thus we have established the following theorem.

Theorem 4.7 *Between two arbitrary endpoints belonging to $\mathcal{H}$, an approximate orthogonal shortest distance through a path, avoiding $\mathcal{B}$, whose length is at most three times the optimal length can be obtained in $O(\log n)$ query time using $O(n \log^2 n)$ preprocessing and $O(n \log n)$ space.*

To illustrate the fact that the ratio of the approximation can be three we refer to figure 4.8. The three vertical line segments b_1, b_2 and b_3 are denoted by ab, cd and ef respectively. Here b_2 is the median line segment. Let p and r be two corner points of b_4 and b_5 respectively. The shortest distance between p and r is $l_6+l_1+l_3+l_4+l_5+l_2+l_7$. But j being nearer to p than i and g being nearer r than h the reported path will be $pkbjihdgelr$ and its length is $l_6+(l_1-x)+l_3+(l_1-x)+l_1+l_4+l_2+(l_2-y)+l_5+(l_2-y)+l_7$ which is equal to $3l_1 + 3l_2 + l_3 + l_4 + l_5 + l_6 + l_7 - 2x - 2y$. Thus if l_1 and l_2 are much larger compared to l_3, l_4, l_5, l_6, l_7, x and y then the ratio of the reported distance to the optimal distance will approach to three. Thus this example shows that the constant of our approximation in this case is also tight.

Now we will relax the endpoint restriction and present an algorithm to answer the approximate shortest-distance query between two arbitrary points. The algorithm here is simpler than the algorithm used in the case of rectangles. Let s and t be two arbitrary points. Again without loss of generality we assume $s_x \leq t_x$ and $s_y \leq t_y$.

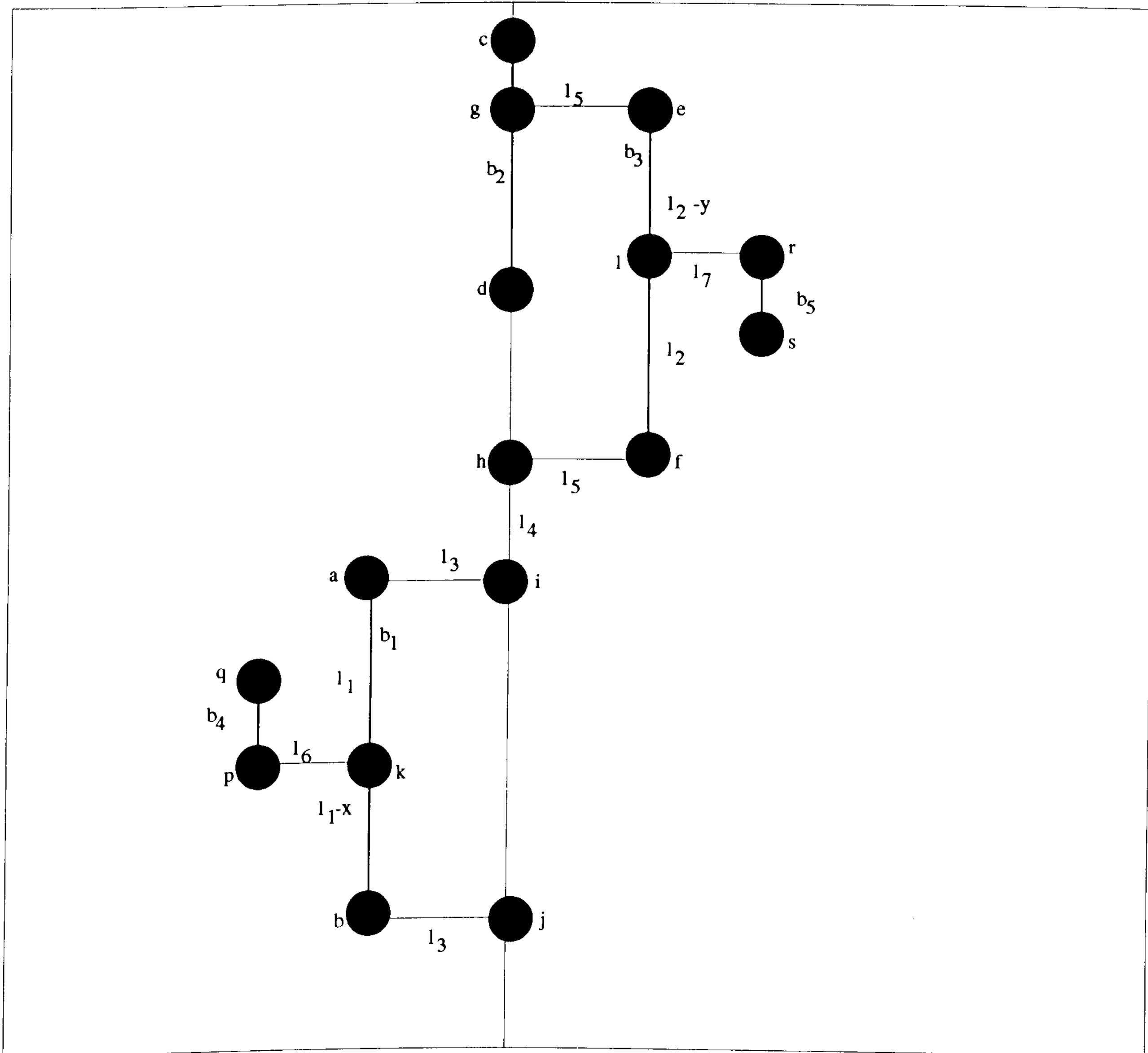


Figure 4.8: A Case in which the Constant of the Approximation For Vertical Line Segments is Tight

Also we will assume a bounding rectangle R around the line segments of $\mathcal{B}$. Here follows the details of the algorithm.

Algorithm General-Query :

- **Step I.** Shoot a horizontal ray towards right from s and another horizontal ray towards left from t. Let b_i and b_j, respectively be the two line segments hit by the two rays from s and t. Here we note that b_i or b_j may belong to $\mathcal{B}$ or may be a side of the enclosing rectangle R. Let u_1 and u_2 be the two endpoints of b_i and v_1 and v_2 be the two endpoints of b_j.

- **Step II.** If $b_{i_x} \leq b_{j_x}$ then compute $asd(s,t)$ as

  $$min\{L_1(s,u_i) + asd(u_i,v_j) + L_1(v_j,t)\} \text{ where } i \in \{1,2\} \text{ and } j \in \{1,2\}$$

  In the above expression $asd(u_i, v_j)$, where $i \in \{1,2\}$ and $j \in \{1,2\}$, is computed using the algorithm **Query**. **Stop**.

- **Step III.** If $b_{i_x} > b_{j_x}$ then compute $asd(s,t) = sd(s,t) = |s_x - t_x| + |s_y - t_y|$.

The proof of the correctness of the algorithm follows from a similar argument presented in **Lemma 4.6**, noting the fact that **case II** in the proof of the lemma will not arise here. Each ray shooting can be carried out in $O(\log n)$ time by preprocessing the planar subdivision formed by the horizontal trapezoidation using the algorithm of [31] or [15] for the point location query. Also the approximate shortest-path between s and t can be reported in $O(\log n + k)$ time. This follows from the fact that if $asd(s,t)$ is determined from **step II.** of **General-Query** then the path can be retrieved from the information maintained in **ASPST** during the preprocessing phase. Otherwise if $asd(s,t)$ is determined in **step III** of **General-Query** then the path can be computed using a similar procedure as described for the case of rectangles in the previous chapter. Thus we have established the following theorem.

Theorem 4.8 *Between two arbitrary query points an approximate orthogonal shortest distance through a path, avoiding $\mathcal{B}$, whose length is at most three times the optimal length can be obtained in $O(\log n)$ query time using $O(n \log^2 n)$ time and $O(n \log n)$ space. The actual path can be reported in $O(\log n + k)$ time where k is the size of the reported path.*

Chapter 5

Parallel Shortest Path Algorithms

In this chapter we will present parallel algorithms for the shortest-path problem. For the shortest-path computation inside simple polygons on the CREW PRAM model of computation Goodrich et al., [22] introduced a data structure, called stratified decomposition tree, which can be constructed in $O(\log n)$ time using $O(n)$ processors, that allows a single processor to construct an implicit representation of the shortest L_2 path between two points in $O(\log n)$ time. The result is based on the idea of constructing a hierarchy of nested subpolygons over an underlying triangulation, whose dual is known to have a tree form, such that any shortest path crosses a small number of subpolygons.

In case the environment consists of axes parallel boxes as the barriers Atallah and Chen [2], [3] removed the known origin restriction and described a preprocessing method, which can be performed in $O(\log^2 n)$ time using $O(n^2/\log n)$ processors in the CREW PRAM model, that allows one processor to report the length of the shortest path between arbitrary two barrier vertices in constant time or $O(n/\log n)$ processors to retrieve the shortest path in $O(\log n)$ time.

In this chapter we present efficient parallel algorithms for the **SPQ** problem where the barriers in $\mathcal{R}$ are disjoint planar rectangles whose sides are parallel to the coordinate axes, and subsequent queries ask for the shortest L_1 path between two arbitrary points which avoids the barriers in $\mathcal{R}$. We utilize the geometric property of monotonicity of the shortest path between every pair of co-planar points, in this setting,

to construct in parallel three carrier graphs, as described in **Chapter 3** which either contain the shortest path between every two corners of the rectangles in $\mathcal{R}$ or can be used to guide the search for such path. This differs from the approach of Atallah et al [2], [3] which is based on the use of *staircase separators* to recursively partition the set of rectangles. As will be demonstrated in the rest of the chapter, the construction of these graphs and subsequent search of the shortest path in each of those graphs can be performed efficiently in CREW PRAM model of computation. We present parallel preprocessing algorithms which allow for reporting the shortest distance between two arbitrary query points in $O(\log n)$ time with a single processor. The path itself can also be constructed in time proportional to its number of segments. Our method is again based on constructing three planar graphs, called carrier graphs, that contain the shortest path information in a succinct form. Each graph can then be searched using graph theoretic techniques. Using the same technique we also present a parallel algorithm for computing the orthogonal shortest distance between two points among rectangular obstacles which runs in poly-logarithmic time using sub-quadratic number of processors on the CREW PRAM model of computation.

In the following section we first present parallel algorithms for constructing these graphs to answer queries for length of the shortest L_1 path between any two corner points of rectangles in $\mathcal{R}$. Then we describe a preprocessing in $O(\log^3 n)$ time, using $O(n^2/\log^2 n)$ processors in the CREW PRAM, that allows one processor to answer each subsequent query in $O(\log n)$ time. The retrieval of the shortest path can be performed in time proportional to the number of its segments. The details of removing the restriction that query points are corner points of rectangles in $\mathcal{R}$ are exactly similar to that described in **Chapter 2**.

As a byproduct we show, in **section 5.2**, that the described graphs can be used to solve the single-shot shortest distance problem in $O(\log^3 n)$ time using $O(n^{1.5}/\log^2 n)$ processors in the CREW PRAM model of computation.

Lastly in **section 5.3** we consider the the parallel preprocessing algorithm for the vertical line segment obstacles. Here the preprocessing algorithm unlike the case of rectangles can be parallelized with subquadratic processor-time product. The processor and the time complexities of the parallel preprocessing algorithm are $O(n^{1.5}/\log^2 n)$

and $O(\log^4 n)$ respectively. This preprocessing enables us to answer the approximate shortest-distance query between two arbitrary points in $O(\log n)$ time and the approximate shortest-path query between those query points in $O(\log n + k)$ time.

5.1 Parallel Preprocessing and Sequential Query algorithms

Our approach of preprocessing the set of barriers $\mathcal{R}$, for the CREW PRAM model of computation, such that subsequent shortest-distance queries can be handled efficiently by a single processor follows the method of **Theorem 3.2**. That is, we rely on the transitive closure of the planar undirected graphs UG_{+x}, UG_{+y}, UG_{-y} as our main preprocessing operation. But first some necessary parallel operations.

Lemma 5.1 [2] *Given a set $\mathcal{R}$ of disjoint orthogonal rectangles and a point s. Each of the eight chains C_i, where $i = 1..8$, can be constructed in $O(\log n)$ time using $O(n)$ processors on the CREW PRAM model of computation.*

Lemma 5.2 [5] *Any synchronous parallel algorithm taking time T that consists of W operations can be simulated by P processors in time $O((W/P) + T)$.*

We now list the preprocessing steps of UG_{+x}, the underlying planar undirected graph of G_{+x}, which proceeds as follows :

- **Step I.** Construct the undirected graph UG_{+x}. This task can be performed by two applications of the shadow computation from [4].

- **Step II.** Triangulate the graph UG_{+x} using $O(n)$ processors and $O(\log n)$ time. This will transform the graph to be biconnected along with the fact that each face of the graph will be bounded by a constant number of vertices [33]. We assign high weights to the additional edges of the triangulation so that it never enters the shortest path.

We then compute a planar separator using the algorithm of Gazit and Miller [21]. This algorithm is a deterministic version of the previous randomized algorithm [20] which uses a deterministic algorithm for the computation of the maximal independent set [23]. The algorithm [21] runs with $O(n + f^{1+\epsilon})$ processors and $O(\log^3 n)$ time where f is the number of faces in the planar graph and ϵ is any positive constant.

- **Step III.** Compute the transitive closure of the graph described in [41] using $O(n^2/\log^2 n)$ processors and $O(\log^3 n)$ time.

- **Step IV.** For each of the corner points of $\mathcal{R}$ we compute and store the eight chains C_i, $i = 1, 2, \ldots, 8$. By **Lemma 5.1**, each of these chains can be constructed with $O(n)$ processors in $O(\log n)$ time. However, the result of **Lemma 5.2** allows for computing all the chains in $O(\log^3 n)$ time using $O(n^2/\log^2 n)$ processors.

Therefore preprocessing can be performed in $O(\log^3 n)$ time using $O(n^2/\log^2 n)$ processors.

Given two corner points $s, t \in \mathcal{R}$, we compute the shortest distance as follows :

(i) Check whether t lies in the region R_2 of s by performing binary search ober the chains C_2 and C_3 of s.

(ii) If t does not lie in the region R_2 of v_i, then we retrieve $sd(o, d)$ from the precomputed transitive closure. Otherwise, $sd(o, d) = |o_x - d_x| + |o_y - d_y|$.

Therefore the shortest distance query between two corner vertices can be processed in $O(\log n)$ time with a single processor. The correctness of the algorithm follows from the fact, proved in **Lemma 3.5**, that the shortest paths to all corner points in the region R_1 of s is maintained in G_{+x}, and thus the underlying undirected graph UG_{+x}. We have therefore established the following theorem :

Theorem 5.1 *An orthogonal shortest distance query between two arbitrary corner points amidst the set of rectangles $\mathcal{R}$ can be obtained in $O(\log n)$ time using a single processor after performing a preprocessing with $O(n^2/\log^2 n)$ processors in $O(\log^3 n)$ time on the CREW PRAM model of computation.*

Note that computing the transitive closure of the carrier graphs would have eliminated the need for the preprocessing in **step IV**, and reduced the query time to $O(1)$. However the best known parallel algorithm for performing such an operation is $O(\log^4 n)$ time using $O(n^2/\log^2 n)$ processors in the CRCW PRAM model [33].

5.1.1 Removal of Corner Point Restriction

In this section we describe the additional preprocessing required to remove the restriction that the origin and destination be corner points of the rectangular barriers. Again our approach is based on observing that each of the origin and destination points can be associated with a pair of vertices in each of the carrier graphs (as shown in figure 3.7 & figure 3.9). The shortest paths between the identified vertices in each graph are sufficient to support the shortest path computation from the origin to destination. There is a maximum of *four* such paths in each graph. Therefore, at most *eight* **SPQ** problems need to be solved.

Assume, without loss of generality, that t lies in the first quadrant with respect to s. We shoot a ray towards right from s and let u_1u_2 be the edge of the rectangle which it hits first. In a similar way we shoot a ray towards left from t and v_1v_2 be the edge of the rectangle which it hits first. In a similar way we obtain two more edges u_3u_4 and v_3v_4 by shooting two rays upwards and downwards from s and t respectively.

All the above computation can be efficiently performed during query time once we preprocess the horizontal and the vertical trapezoidation by constructing the hierarchical representation, introduced by Dadoun and Kirkpatrick [31]. For each subsequent query we perform point-location on the hierarchies to locate the rectangles in the subdivision that contain the point s and t.

The shortest distance between s and t can now be computed as follows

- **Step I.** Compute the minimum among four following expressions : $L_1(s, u_i) + sd(u_i, v_j) + L_1(v_j, t)$, where $i \& j \in \{1, 2\}$. Here $sd(a, b)$ denotes the shortest distance between a and b in G_{+x}.

  If the minimum value is defined then the shortest distance between s and t is found. Otherwise, we proceed to the next step.

- **Step II.** Compute the minimum among four following expressions : $L_1(s, u_i) + sd(u_i, v_j) + L_1(v_j, t)$ where $i\&j \in \{3, 4\}$. Here $sd(a, b)$ denotes the shortest distance between a and b in G_{+y}.

  If the minimum value is defined then the shortest distance between s and t is found. Otherwise, we proceed to the next step.

- **Step III.** The shortest distance between s and t equals $|s_x - t_x| + |s_y - t_y|$.

The above mentioned hierarchy can be constructed [11], with $O(n)$ processors and $O(\log n)$ time to support an $O(\log n)$ point location queries. Therefore, we can conclude that

Theorem 5.2 *The orthogonal shortest path between two arbitrary points can be determined by a single processor in $O(\log n + k)$ time with $O(\log^3 n)$ time preprocessing which uses $O(n^2/\log^2 n)$ processors in the CREW PRAM model. where k is the number of segments in the constructed shortest path.*

5.2 An Efficient Parallel Single-Source Shortest-Distance Algorithm

In this section we describe a CREW PRAM algorithm for computing the shortest distance among a set of barriers $\mathcal{R}$. The algorithm makes use of the knowledge of the origin and destination points, s and t respectively, to achieve a better performance.

The first part of this section is devoted to defining a planar graph which contains an orthogonal path of shortest length between two points among rectangular obstacles and to describing an algorithm for its construction. In the second part we describe the algorithms used for searching the constructed graph to compute the shortest distance and to report the actual shortest path.

5.2.1 Constructing the Graph

Given a set $\mathcal{R}$ of disjoint orthogonal rectangles and two points s and t. We assume, without loss of generality, that t lies in the first quadrant with respect to s[1]. Therefore t lies either in $R_{+x} = R_8 \cup R_1 \cup R_2$, where a monotone path in the $+x$ direction is known to exist, or in $R_{+y} = R_2 \cup R_3 \cup R_4$, where a monotone path in the $+y$ direction is known to exist. Since both situations are handled similarly, only the former will be discussed.

Let $\mathcal{Q}$ be the subset of $\mathcal{R}$ in the region R_{+x}. Construct the horizontal shadow decomposition of R_{+x} by connected each of the corner points in $\mathcal{Q} \cup C_8 \cup C_3$ and s to their right shadows. Partition the region further by connecting each of the corner points on the left vertical side of a rectangle in $\mathcal{Q}$ and t to their upper and/or lower shadows in the previously constructed partition. (Refer to figure 5.1 and figure 5.2). We now construct a graph $G = (V, E)$ such that:

- V consists of the corner points in $\mathcal{Q} \cup C_8 \cup C_3$, the points s and t, and the computed right, upper and lower shadows
- two elements $a, b \in V$ are connected by an edge in E if
  1. a is a shadow point and b is the corresponding corner point in $\mathcal{Q} \cup C_8 \cup C_3$, or s or t
  2. they form a segment in $C_8 \cup C_3$ or on the boundary of a rectangle in $\mathcal{Q}$

In addition, a *weight* is associated with each edge which equals the length of the corresponding segment.

Lemma 5.3 *The weighted graph G is a planar graph with $O(n)$ vertices which contains the shortest orthogonal path from s to t that avoids the set $\mathcal{R}$ of rectangular obstacles.*

Proof: That G is an planar graph with $O(n)$ vertices follows trivially from the construction. Only thing that remains to be shown is that the shortest path built up during the plane sweep algorithm of [12] from s to t lies in our graph G.

[1] That is, $x_t \geq x_s$, $y_t \geq y_s$, and $t \neq s$

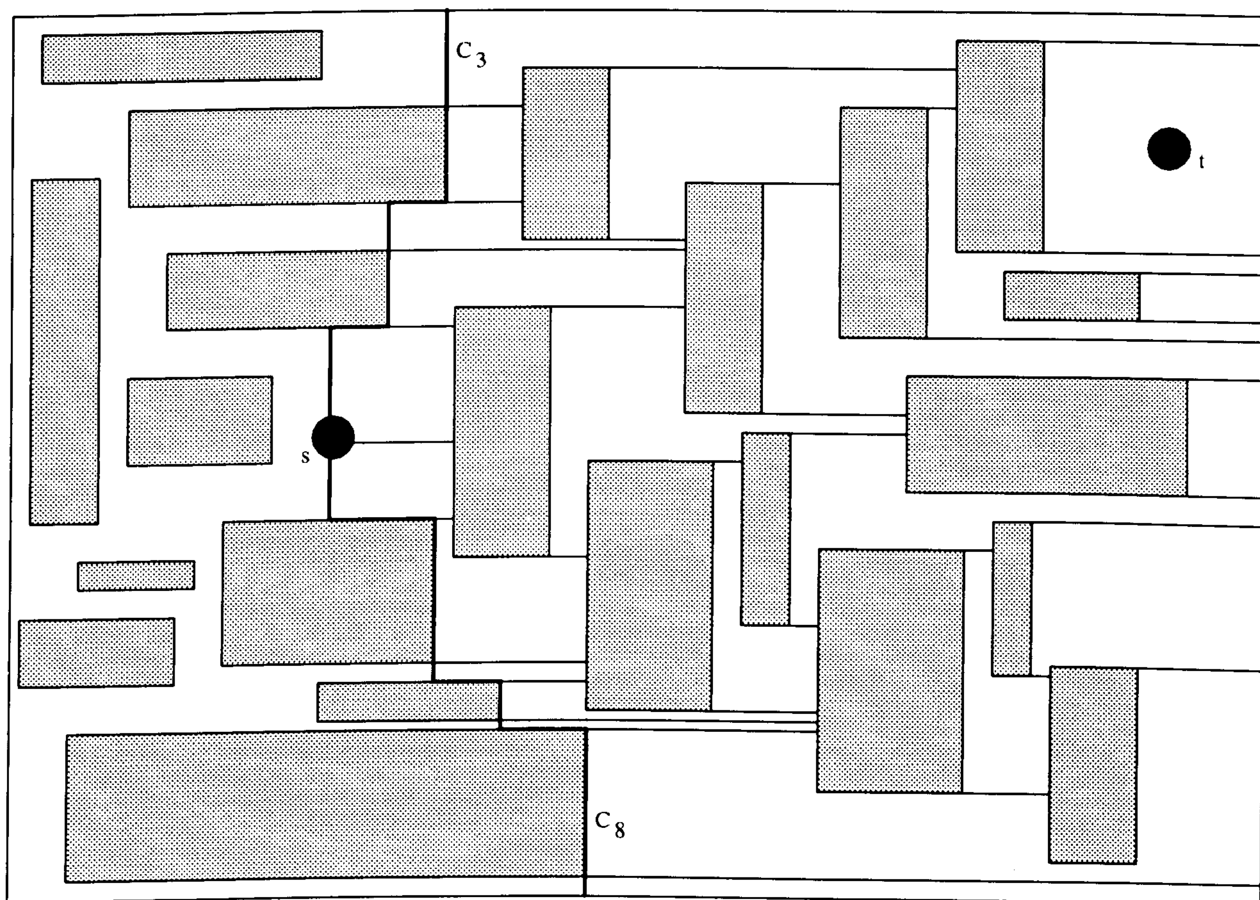


Figure 5.1: Construction of the Graph G : Step I - Right Shadows in R_{+x}

Figure 5.2: Construction of the Graph G : Step III - Upper and Lower Shadows in R_{+x}

We give a proof by induction. We denote the rank of a vertex pair in the region R_{+x} forming a left vertical side of a rectangle as the position in the sorted list of vertex pairs in the region R_{+x} where the sorting is done on x-coordinates excluding the two chains C_3 and C_8. For all vertices on chains C_3 and C_8 we assume that they have rank zero. Our assumption in the induction is for every vertex pair in V_l having rank less than or equal to $(m-1)$ the shortest path from s to that pair is maintained in G. We have to show that the assumption holds for the vertex pair with m th rank also.

The basis holds trivially for all vertices on chains C_3 and C_8. Let $v_i, v_j \in V_l$ be the vertex pair having m th rank. Without loss of generality we consider only v_i, because the argument holds for v_j also. Now there is always a shortest path from s which enters v_i through either of the two vertices u_a or u_b as described in **Lemma 3.4**. Since both u_a and u_b have rank less than m both the shortest paths i.e. from s to u_a and from s to u_b are maintained in G from the inductive assumption. Also it is quite apparent from the construction of G that v_i is joined to both u_a and u_b by shortest paths whose lengths are L_1 distance between them. So the shortest path from s to v_i is also maintained in our graph. This completes the proof our theorem. □

Lemma 5.4 *The weighted graph G can be constructed in $O(\log n)$ time using $O(n)$ processors in the CREW PRAM model of computation.*

Proof: The above graph can be constructed by first building up the two chains C_3 and C_8 according to **Lemma 5.1**. Then we can mask all the rectangles which doesn't belong to R_{+x} with $O(n)$ processors and $O(\log n)$ time. This task can be performed by assigning one processor to each rectangle. They can perform a binary search over the two chains to determine if the rectangle is in R_{+x} or not and correspondingly enable and disable them. The subsequent operation for building up the graph can be performed by two applications of [4] for shadow computations. □

5.2.2 Searching the Graph

The remaining part of the algorithm is the computation of the shortest path in parallel in the graph G. This operation can be carried out by an application of the algorithm of Pan and Reif [41]. We give a brief outline for this operation.

Step I. We triangulate the graph G using $O(n)$ processors and $O(\log n)$ time. This will transform the graph to be biconnected along with the fact that each face of the graph will be bounded by a constant number of vertices [33]. We assign high weights to the additional edges of the triangulation so that it never enters the shortest path.

Step II. We precompute the planar separator as required in [41] using an algorithm of [21]. This algorithm is a deterministic version of the previous randomized algorithm [20] which uses a deterministic algorithm for the computation of the maximal independent set [23]. The algorithm [21] runs with $O(n + f^{1+\epsilon})$ processors and $O(\log^3 n)$ time where f is the number of faces in the planar graph and ϵ is any positive constant.

Step III. Lastly we compute the shortest path between s and t using [41] in $O(\log^3 n)$ time with $O(n^{1.5}/\log^2 n)$ processors.

So the time and processor complexity of the overall algorithm is dominated by the shortest path computation in [41] using $O(n^{1.5}/\log^2 n)$ processors and $O(\log^3 n)$ time. Hence we have the following theorem :-

Theorem 5.3 *The rectilinear shortest distance between any two points in presence of rectangular obstacles can be computed in parallel with $O(n^{1.5}/\log^2 n)$ processors and $O(\log^3 n)$ time in CREW PRAM model of computation.*

5.3 Parallel Preprocessing Algorithm for Vertical Line Segments

In this section we will concentrate on vertical line segment obstacles. Here we present a parallel preprocessing algorithm to answer using a single processor the approximate

shortest-distance query between two arbitrary points. The work done in the preprocessing phase, i.e., the processor-time product of the preprocessing algorithm is much less than the processor-time product of the algorithm presented for rectangles to report the exact path. Here are the details of the algorithm.

Algorithm Parallel-Preprocess :

Comments : We presort the line segments by their $x-$coordinates using the algorithm of [6] with $O(n)$ processors in $O(\log n)$ time.

- **Step I.** Determine the line segment l_i with the median $x-$coordinate. Subsequently we compute $\mathcal{T}$ the set of all horizontal projections of the end points of $\mathcal{L}$ on the vertical line passing through l_i. This can done using the parallel trapezoidation algorithm of [4] using $O(n)$ processors and $O(\log n)$ time. Let L_1, L_2 respectively denote the set of line segments to the left and to the right of l_i.

- **Step II.** Compute the undirected carrier graphs UG_{-x} and UG_{+x} for $\mathcal{T} \cup L_1$ and $\mathcal{T} \cup L_2$. This can be done by the algorithm of [4] for shadow computation. We need to apply the algorithm twice for each UG_{-x} and UG_{+x}.

- **Step III.** Triangulate the graph UG_{+x} using $O(n)$ processors and $O(\log n)$ time. This will transform the graph to be biconnected along with the fact that each face of the graph will be bounded by a constant number of vertices [33]. We assign high weights to the additional edges of the triangulation so that it never enters the shortest path.

  We then compute a planar separator using the algorithm of Gazit and Miller [21]. This algorithm is a deterministic version of the previous randomized algorithm [20] which uses a deterministic algorithm for the computation of the maximal independent set [23]. The algorithm [21] runs with $O(n + f^{1+\epsilon})$ processors and $O(\log^3 n)$ time where f is the number of faces in the planar graph and ϵ is any positive constant.

- **Step IV.** Compute the geodesic Voronoi neighbours of $\mathcal{H}$ in $\mathcal{T}$. For this we use **Lemma 4.2**. The shortest-path computation in thta lemma can be carried out using the parallel single-source shortest-path algorithm of [41] with $O(n^{1.5}/\log^2 n)$ processors and $O(\log^3 n)$ time.

- **Step V.** Recursively apply the above four steps for L_1 and L_2 in parallel.

Thus the overall time complexity $T(n)$ of the above preprocessing algorithm can be obtained from the following recurrence :-

$$T(n) \leq T(n/2) + c * \log^3 n, \text{ where } c \text{ is a positive constant.}$$

Therefore $T(n) \in O(\log^4 n)$. The processor complexity of the algorithm remains $O(n^{1.5}/\log^2 n)$, since we will always have sufficient number of processors allocated at each subproblem.

To report the approximate shortest-path between two arbitrary query points we use the same algorithm as described in **section 4.4**. But to support the ray shooting query we preprocess the planar subdivision formed by horizontal trapezoidation using the algorithm of [11] with $O(n)$ processors and $O(\log n)$ time to support the point location query in $O(\log n)$ time. Thus we have established the following theorem.

Theorem 5.4 *Between two arbitrary query points an approximate orthogonal shortest distance through a path at most three times the optimal length can be obtained in $O(\log n)$ query time using a preprocessing algorithm which requires $O(n^{1.5}/\log^2 n)$ processors and $O(\log^4 n)$ time. The actual path can be reported in $O(\log n + k)$ time where k is the size of the output path.*

Chapter 6

Conclusion and Open Problems

We have considered the shortest-path problem where the barriers are disjoint planar rectangles whose sides are parallel to the coordinate axes, and subsequent queries ask for a shortest L_1 path between two arbitrary points which avoids the barriers. Our solutions are based on the use of *carrier graphs* which are planar graphs that contain shortest path information between every two corners of the rectangular barriers. Three such graphs either contain the shortest path between every two corners of the rectangular barriers or can be used to guide the search for such path. In addition, each carrier graph generates a decomposition of the plane which allows for shortest path queries between arbitrary two points to be answered efficiently.

More precisely, we used the planarity of the **carrier graphs** to achieve sequential preprocessing in sub-quadratic time and space, $O(n^{1.5})$, which can support sublinear shortest path queries, $O(\sqrt{n})$ time, when both *origin* and *destination* are both part of the query. In addition such preprocessing can be performed with $O(n^2/\log^2 n)$ processors in $O(\log^3 n)$ time in the CREW PRAM model. These parameters improve by a $O(\log n)$ factor the known processor complexity of the preprocessing, previously achieved in [2], [3]. But the total work, i.e., the processor-time product of the preprocessing phase of our algorithm equals the processor-time product of the algorithm of [3].

In **Chapter 4** we presented efficient preprocessing algorithms to answer the approximate shortest-path query between two arbitrary points. The approach here is

based on the use of staircase separator and the geodesic Voronoi diagram computation. Our approximation algorithm in **Chapter 4** for the case of rectangles uses $O(n\log^3 n)$ preprocessing, $O(n\log^2 n)$ space to answer the approximate shortest-distance query between two arbitrary points in $O(\log n)$ time. For the case of vertical line segments our approach uses $O(n\log^2 n)$ preprocessing time, $O(n\log n)$ space and $O(\log n)$ query time. Moreover the preprocessing algorithm for the case of vertical line segments can be parallelized using $O(n^{1.5}/\log^2 n)$ processors and $O(\log^4 n)$ time.

It is possible to extend the work done in this thesis in different ways. Here are a few possible directions.

- All our parallel algorithms to answer the optimal shortest-path query use the parallel shortest-path algorithms in planar graphs. So the complexities in terms of processor and/or time can be improved if it is possible to design more efficient parallel algorithms for the single-source and all-pairs shortest-path problems in planar graphs.

- The approximation algorithm in **Chapter 4** for the case of rectangles doesn't lend itself for efficient parallelization. This is due to the fact that there we require the shortest-path computation on the sparse visibility graph which is non-planar. This is in contrast to the approximation algorithm for vertical line segment obstacles which is more efficiently parallelizable owing to the fact that there we use carrier graphs for the shortest-path computation. With the existing techniques a naive parallelization of the approximation algorithm for the case of rectangles would not improve the processor-time product of the parallel preprocessing algorithm to report the optimal path. So an interesting open question is to design a parallel preprocessing algorithm with a low processor-time product to answer the approximate shortest-path query in the presence of rectangular obstacles.

- One natural open problem concerns the design of an algorithm with lower preprocessing and storage to answer the exact shortest-distance query between two arbitrary points.

- Other open problems concern the design of an efficient preprocessing algorithm to answer the rectilinear or Euclidean shortest-path query, between two arbitrary query points, in the presence of other types of polygonal obstacles.

Bibliography

[1] E. M. Arkin, J. S. B. Mitchell and C. D. Piatko, "Bicriteria shortest path problems in the plane", *Proceedings of the Third Canadian Conference on Computational Geometry* , pp. 153 – 156.

[2] M. J. Atallah and D. Z. Chen, "Parallel rectilinear shortest paths with rectangular obstacles," *Computational Geometry: Theory and Applications* , 1(2), 1991, pp. 79 – 113.

[3] M. J. Atallah and D. Z. Chen, "On Parallel Rectilinear Obstacle-Avoiding Paths," *Computational Geometry : Theory and Applications* , 3 (1993), pp. 307 – 313.

[4] M. J. Atallah R. Cole and M. J. Goodrich, "Cascading divide-and-conquer: a technique for designing parallel algorithms," *SIAM J. Comp.*, 1989, pp. 499 – 532.

[5] R. P. Brent, "The parallel evaluation of general arithmetic expressions," *J. of the ACM* 21(2), 1974, pp. 201–206.

[6] R. Cole, "Parallel merge sort," *SIAM J. Comp.*, 1988, pp. 770–785.

[7] K. Clarkson, "Approximation algorithms for shortest path motion planning", *Proc. 19*th *Annual ACM Symposium on Theory of Computing*, 1987, pp. 56 – 65.

[8] K. L. Clarkson, S. Kapoor and P. M. Vaidya, "Rectilinear shortest paths through polygonal obstacles", *Proc. of 3*rd *Symposium on Computational Geometry*, 1987, pp. 251 – 257.

[9] J. F. Canny and J. Reif, "New lower bound techniques for robot motion planning problems", *Proc. of 28th Annual Symposium on Foundations of Computer Science*, 1987, pp. 49 – 60.

[10] L. P. Chew, "There is a planar graph almost as good as the complete graph", *Proc. of 2nd Symposium on Computational Geometry*, 1986, pp. 160 – 177.

[11] N. Dadoun and D. G. Kirkpatrick, "Parallel processing for efficient subdivision search", *Proc. of 3rd Symposium on Computational Geometry*, 1987, pp. 205 – 214.

[12] P. J. de Rezende, D. T. Lee, and Y. F. Wu, "Rectilinear shortest paths with rectangular barriers," *Proc. of 1st Symposium on Computational Geometry*, 1985, pp. 204 – 213.

[13] D. P. Dobkin, S. J. Friedman and K. J. Supowit, "Delaunay graphs are almost as good as complete graphs", *Proc. of 28th Annual Symposium on Foundations of Computer Science*, 1987, pp. 20 – 26.

[14] E. W. Dijkstra, "A note on two problems in connexion with graphs", *Numer. Math. 1*, 1959, pp. 269 – 271.

[15] H. Edelsbrunner, L. J. Guibas and J. Stolfi, "Optimal point location in monotone subdivision," *SIAM J. Comp.*, 1986, pp. 317–340.

[16] H. ElGindy and M. Goodrich, "Parallel algorithms for shortest path problems in polygons", *Visual Computer* , 1988, pp. 371 – 378.

[17] H. ElGindy and P. Mitra, "Orthogonal shortest route queries among axes parallel rectangular obstacles", *Int. J. of Comput. Geom. and Applications*, 4(1), pp. 3–24.

[18] M. Fredman and R. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms", *Proc. 25th Annual IEEE Symposium on Foundations of Computer Science*, pp. 338 – 346, 1984.

[19] S. K. Ghosh and D. M. Mount, "An output sensitive algorithm for computing visibility graphs", *Proc. 28th Annual IEEE Symposium on Foundations of Computer Science*, 1987, pp. 11 – 19.

[20] H. Gazit and G. L. Miller, "A parallel algorithm for finding a separator in planar graphs," *Proc. of 28th Annual Symposium on Foundations of Computer Science*, 1987, pp. 238 - 248.

[21] H. Gazit and G. L. Miller, "A deterministic parallel algorithm for finding a separator in planar graphs," *manuscript*, 1990.

[22] M. T. Goodrich, S. B. Shauck and S. Guha, "Parallel methods for visibility and shortest path problems in simple polygons", *Proc. of 6th Symposium on Computational Geometry*, 1990, pp. 73 – 82.

[23] M. Goldberg and T. Spencer, "Constructing a maximal independent set in parallel," *SIAM J. DISC. Math.*, 1989, pp. 322 - 328.

[24] L. J. Guibas and J. Hershberger, "Optimal shortest path queries in a simple polygon", *Proc. 3rd Symposium on Computational Geometry*, 1987, pp. 50 - 63.

[25] L. Guibas, J. Hershberger, D. Leven, M. Sharir and R. Tarjan, "Linear time algorithms for visibility and shortest path problems inside simple polygons", *Proc. of 2nd Symposium on Computational Geometry*, 1986, pp. 1 - 13.

[26] J. Hershberger, "Optimal Parallel Algorithms for Triangulated Simple Polygons", *Proc. of 8th Symposium on Computational Geometry*, 1992, pp. 33 – 42.

[27] J. Hershberger and S. Suri, "Efficient Computation of Euclidean Shortest Paths in the Plane", *Proc. of 34th Annual Symposium on Foundations of Computer Science*, 1993, pp. 508 – 517.

[28] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *J. of the ACM* , 1977, pp. 1–13.

[29] T. Kameda, "On the vector representation of the reachability in planar directed graphs", *Information Processing Letters* , 3(3), 1975, pp. 75 – 77.

[30] M. Keil and C. Gutwin, "The Delaunay triangulation closely approximates the complete graph", *Proc. of 1st Canadian Workshop on Algorithms and Data Structures* , 1989, pp. 47 – 56.

[31] D. G. Kirkpatrick, "Optimal search in planar subdivisions," *SIAM J. Comp.* 1983, pp. 28-35.

[32] D. T. Lee and F. P. Preparata, "Euclidean shortest paths among rectilinear barriers", *Networks,* , 1983, pp. 393 – 410.

[33] A. Lingas, "Efficient parallel algorithms for path problems in planar directed graphs," *Proc. of SIGAL Conference on Algorithms,* August 1990, Tokyo, pp. 447 –457.

[34] R. J. Lipton and R. E. Tarjan, "A separator theorem for planar graphs," *SIAM J. Numer. Anal.*, 1979, pp. 177-189.

[35] K. Mehlhorn, A faster approximation algorithm for the Steiner problem in graphs, *Information Processing Letters 27*, 1988, pp. 125-128.

[36] J. S. B. Mitchell, "L_1 shortest paths among polygonal obstacles in the plane", *Algorithmica, 8(1)*, 1992, pp. 55 – 88.

[37] J. S. B. Mitchell, " Algorithmic approaches to optimal route planning", *Technical Report No. 937, School of Operations Research and Industrial Engineering, Cornell University*, 1990.

[38] J. S. B. Mitchell, G. Rote and G. Woeginger, "Minimum-Link paths among obstacles in the plane", *Proc. of 6th Annual ACM Symposium on Computational Geometry* , 1990, pp. 63 – 72.

[39] J. S. B. Mitchell and C. H. Papadimitriou, "The weighted region problem : finding smallest paths through a weighted planar subdivision", *JACM* , 1991 (38), pp. 18 - 73.

[40] J. S. B. Mitchell, "Shortest Paths Among Obstacles in the Plane", *Proc. of 9*th *Symposium on Computational Geometry*, 1993, pp. 308 - 317.

[41] V. P. Pan and J. Reif, "Fast and efficient solution of path algebra problems," *J. of Computer and System Sciences*, vol. 38, 1989, pp. 494 - 510.

[42] C. H. Papadimitriou, "An algorithm for shortest-path motion in three dimensions", *Information Processing Letters* , 20, 1985, pp. 259 - 263.

[43] C. H. Papadimitriou, "Shortest Path Motion", *Proc. FST-TCS Conference, New Delhi* , 1987, Springer.

[44] *Theoretical Computer Science* , 1991 (84), pp. 127 - 150.

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

[46] F. P. Preparata and R. Tamassia, "Fully dynamic techniques for point location and transitive closure in planar structures", *Proc. of 29*th *Annual Symposium on Foundations of Computer Science* , 1988, pp. 558 - 567.

[47] S. Suri, "A linear time algorithm for minimum link paths inside a simple polygon", *Computer Vision, Graphics, and Image Processing* , 35, 1986, pp. 99 - 110.

[48] Y. F. Wu, P. Widmayer, M. D. F. Schlag and C. K. Wong, Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles, *IEEE Transactions on Computers, 36(3)*, 1987, pp. 321 - 331.