Shortest Path Problems on Polyhedral Surfaces

By

Mark Anthony Lanthier

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Doctor of Philosophy

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

December 10, 1999

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

Shortest Path Problems on Polyhedral Surfaces

submitted by

Mark Anthony Lanthier

Dr. Evangelos Kranakis
(Director, School of Computer Science)

Dr. Jörg-Rüdiger Sack
(Thesis Supervisor)

Dr. Anil Maheshwari
(Thesis Co-Supervisor)

(External Examiner)

Carleton University

December 10, 1999

Abstract

Shortest path algorithms have been studied for many years, as they have applications in many areas of computer science. Recently, much of the research has been geared towards computing approximations of shortest paths in order to reduce the large time complexities which are inherent to exact solutions. In some instances, approximation algorithms may provide the only solution since there may be no existing algorithm that produces an exact solution.

In this work, we develop several algorithms for computing approximations of weighted shortest paths on polyhedral surfaces. Our techniques are mainly based on discretizing the polyhedron in order to reduce the problem to a graph shortest path problem. We give several schemes which differ in the way in which the polyhedron is discretized. The schemes are based on adding Steiner points to faces of the polyhedron, creating a graph by interconnecting these points and then searching this graph for a solution. We show through experimental evaluation that these techniques are practical.

All of our algorithms allow a tradeoff between accuracy and running time. We provide ϵ-approximation algorithms that improve upon previous work in terms of n (the number of faces of the polyhedron). In addition to Euclidean and weighted metrics, we also give algorithms for computing shortest anisotropic paths. Lastly, we show how these algorithms can be implemented in parallel. Through implementation, we show that our parallel algorithm achieves up to 64% efficiency.

Acknowledgments

To begin, I would like to thank everyone in my life for the support and confidence that they have shown throughout my post graduate life.

I would like to give tremendous thanks to Jörg-Rüdiger Sack for the support that he has given me throughout my degree programs here at Carleton. I have tremendous respect for Jörg and his work and I am happy to have been a part of his research group. In addition, he has given me great advice with respect to written papers and presenting my work to the computer science community. Jörg, your help both academically and financially has been greatly appreciated.

I would also like to give thanks to Anil Maheshwari for the countless hours of time that he has made available to me throughout this thesis preparation. His many constructive comments and technical advice have helped me to complete this thesis in addition to many papers.

Thanks should also be extended to those members on my examining board: Mark Keil, Paola Flocchini, John Goldak and Frank Dehne. It was a thick thesis to read and I really do appreciate their willingness to be on the committee.

I also give thanks to Dr. Lyudmil Aleksandrov for his useful ideas which led to the research on epsilon approximations as well as Dr. Stefan Schirra for his useful comments related to numerical stability as well as his help with LEDA's REAL numbers.

In addition, I have had many fruitful conversations with Doron Nussbaum who I would also like to thank. Doron and I have been doing our thesis work alongside

each other, providing moral support to each other when times seemed tough. Doron and I have grown to be good friends and his friendship and support have helped me in many, many ways.

I would also like to thank Paola Magillo for supplying us with TINs as well as Gerhard Roth from the NRC for providing us with general 3D polyhedral model data.

I would like to dedicate this thesis to my parents Marcel and Beverly Lanthier who have always believed in my abilities to succeed in everything that I have done. I hope this is the first of many Ph.D. degrees in our family. My wife Tricia also deserves acknowledgment for patiently awaiting my thesis completion. I love my family very much and I am glad they were there for me.

This research was partially supported by Almerco, Inc. and by the Natural Sciences and Engineering Research Council of Canada.

Contents

List of Tables

List of Figures

Contents

Chapter 1

Introduction

1.1 Overview

One of the fundamental problems studied in computational geometry and other areas such as graph algorithms, geographical information systems (GIS), circuit design and robotics is that of finding a shortest path between two points in either the plane or in three dimensions.

Shortest path problems have many applications such as traffic control, search and rescue, water flow analysis, road design, facility location, navigation and route planning, ..., to mention just a few that use GIS. For some applications, it may be necessary to determine a shortest cost path on a polyhedral surface. Terrains are a special subclass of polyhedral surfaces and in GIS, many applications (such as those just mentioned) require the computation of shortest paths on a terrain. In GIS, actual data is gathered to model terrain structure within a computer. There are typically thousands of sampled points that make up a terrain and hence a shortest path algorithm must be very efficient in order to be of practical use.

There are several variations of the shortest path problem which depend on different factors pertaining to:

- the cost metric (Euclidean, weighted, Manhattan distance, link distance, etc.).
- the dimension (2D, 2.5D, 3D, 3D polyhedral surfaces).
- specific geometric characteristics such as the presence/absence of obstacles, the type of obstacles (e.g., convex), etc.

In addition, there are issues pertaining to the algorithm itself such as:

- the amount of preprocessing allowed (single shot, fixed source, two-point queries),
- the availability of the queries (static or dynamic),
- the desired accuracy (optimal, approximate, ϵ-approximate),
- the number of processors allowed (sequential or parallel).

The general 3-dimensional Euclidean shortest path problem has been shown to be NP-hard by Canny and Reif [19]. The NP-hardness and the large time complexities of 3-d shortest paths algorithms even for special problem instances have motivated the search for approximate solutions to the shortest path problem. Since most application models are approximations of reality and high-quality paths are favored over optimal paths that are "hard" or expensive to compute, approximation algorithms are suitable and necessary.

The main results presented in this thesis are the design, analysis and implementation of algorithms for computing approximations of Euclidean and weighted shortest paths on polyhedral surfaces. The aim is to design simple and practical algorithms as well as provide algorithms with theoretical worst-case bounds which are efficient

in terms of the number of polyhedral faces. As will be shown, the algorithms can be applied in both the sequential and parallel settings.

Before presenting the algorithms, it is necessary to discuss definitions, notation and properties pertaining to shortest path problems. Also, since there are many variations on the shortest path problem (i.e., in graphs, within polygons, in the plane among polygons, on polyhedral surfaces, in 3-space among polyhedra, etc.) we give here a brief survey of the previous research. Our algorithms will make use of some of the results and techniques presented in this survey.

1.2 Preliminaries

1.2.1 Graphs and Subdivisions

A graph $G = (V, E)$ consists of a set of vertices, V, and a set of edges, E. A graph is *planar* if it can be embedded in the plane such that no two edges intersect. A straight line planar embedding of a planar graph determines a partition of the plane called a *planar subdivision*. The bounded regions of the subdivision are called *faces* and the remaining portion of the plane is termed the *outer face*. A planar subdivision is a *triangulation* if all of its bounded faces are triangles. The *dual graph* of a planar subdivision $\mathcal{S}$ is defined as a graph D in which a vertex of D is assigned to each face of $\mathcal{S}$ and two vertices of D are joined by an edge if and only if their corresponding faces share an edge in S.

1.2.2 Properties of Polyhedra

In $\Re^3$ a *polyhedron* is defined by a finite set of planar polygons such that every edge of a polygon is shared with exactly one other polygon. The polygons are called the *faces* of the polyhedron and the vertices and edges of the polygons are the vertices and edges of the polyhedron. Two faces are said to be *adjacent* if they share an edge, otherwise they are *non-adjacent*. A polyhedron is *simple* if there is no pair of non-adjacent faces that share a point in their interior (i.e., excluding vertices). From this point onwards, the term polyhedron will imply a simple polyhedron. A polyhedron $\mathcal{P}$ is *convex* if for any two points interior to $\mathcal{P}$, the line segment joining them is entirely contained in $\mathcal{P}$, otherwise it is *non-convex*. Let $\mathcal{S}$ be a triangulated planar subdivision lying on the XY plane in $\Re^3$. A *triangular irregular network* (or TIN for short) is a 2.5 dimensional polyhedral surface formed from S by assigning a Z coordinate to each vertex of $\mathcal{S}$ (see Figure 1.1). Note that no two points on the surface of a TIN have the same x and y coordinates.

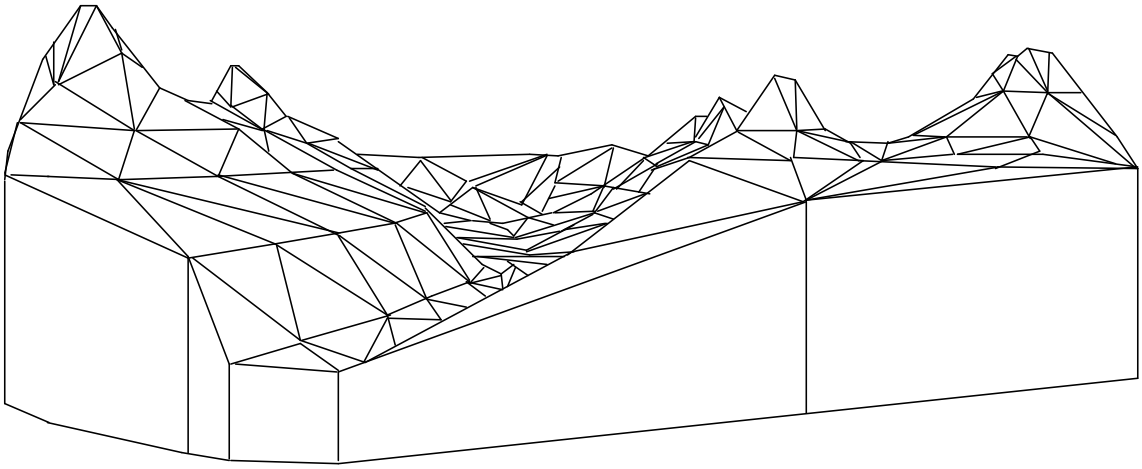

Figure 1.1: Example of a triangular irregular network (TIN).

There are certain geometric properties of $\mathcal{P}$ that we require for our analysis in

the later chapters. These properties require some additional terminology. Let $\mathcal{P}$ have n faces, each being a triangle. Denote by L, the longest edge of $\mathcal{P}$ with Euclidean length $|L|$. Consider a vertex v of $\mathcal{P}$. Define θ_v to be the minimum angle (measured in 3D) between any two edges of $\mathcal{P}$ that are incident to v. Denote the minimum such θ_v as θ. Define h_v to be the minimum distance from v to the boundary of the union of its incident faces.

Property 1.1 *The distance between any two vertices v_a and v_b of $\mathcal{P}$ is at least* $\max(h_{v_a}, h_{v_b})$.

Define a polygonal cap C_v, called a *sphere*, around v, as follows. Let $r_v = \epsilon h_v$ for some $0 < \epsilon < 1$. Let r be the minimum r_v over all v. Let $\triangle vuw$ be a face incident to v. Let u' (respectively on w') be at distance r_v from v on $\overline{vu}$ (respectively $\overline{vw}$). This defines a triangular subface $\triangle vu'w'$ of $\triangle vuw$. The sphere C_v around v consists of all such subfaces incident at v.

Property 1.2 *Let v be a vertex of $\mathcal{P}$ and e be an edge of $\mathcal{P}$ that is not incident to v. Let p be any point on e. The distance between C_v and p is at least $(1-\epsilon)h_v$, where* $0 < \epsilon < 1$.

Property 1.3 *The distance between any two spheres C_{v_a} and C_{v_b} is greater than $(1-2\epsilon)\max(h_{v_a}, h_{v_b})$ for any $\epsilon > 0$.*

In Chapter 3, we will need to ensure that the destination point on a polyhedron $\mathcal{P}$ is not too close to the source point. That, is we will define a region τ_s with respect to a point s on a polyhedron $\mathcal{P}$ as the union of all faces incident to:

i) s	if s is a vertex,
ii) vertices of f_i	if s lies interior to a face f_i,
iii) vertices of f_i and f_j	if s lies on an edge, say e with incident faces f_i and f_j.

Figure 1.2 shows how τ_s is defined when s lies on an edge of P.

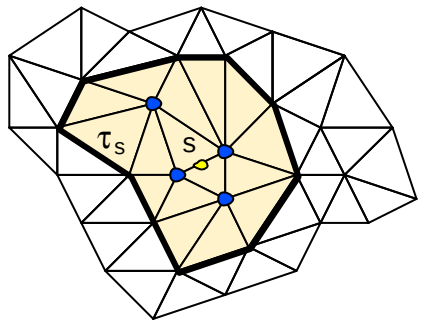

Figure 1.2: The region τ_s surrounding s.

Consider a triangulated polyhedral surface of $\mathcal{P}$. Let $F = f_1, f_2, \ldots, f_k$ be a sequence of faces of $\mathcal{P}$ such that f_i and f_{i+1} are adjacent, where $e_i = f_i \cap f_{i+1}$. An *unfolding* of two adjacent faces f_i and f_{i+1} is the set of points obtained by rotating f_{i+1} about e_i until all vertices of f_{i+1} lie in the same plane as f_i. The unfolding is said to be a *planar-unfolding* if the unfolded faces do not overlap. (Two faces are said to overlap if they have a common point in their interior). A sequence of faces F is unfolded by planar-unfolding the faces in order from 1 to k such that f_i is in the same plane as $f_{i-1}, f_{i-2}, ..., f_1$, where $i > 1$. Let $F' = f'_1, f'_2, ..., f'_k$ be the sequence of faces obtained by applying such an unfolding. F' is called a *simple sleeve* if it is a planar unfolding. An unfolding which is not planar is called a *non-simple sleeve*. The order in which the faces are unfolded defines a sequence of edges around which the faces are rotated. This sequence of edges is called an *edge sequence*. The edge sequence is considered to be a *straight line edge sequence* if and only if one can draw a line segment from a point q_1 on e_1 to a point q_k on e_k such that the segment crosses each edge of the sequence in order (see Figure 1.3).

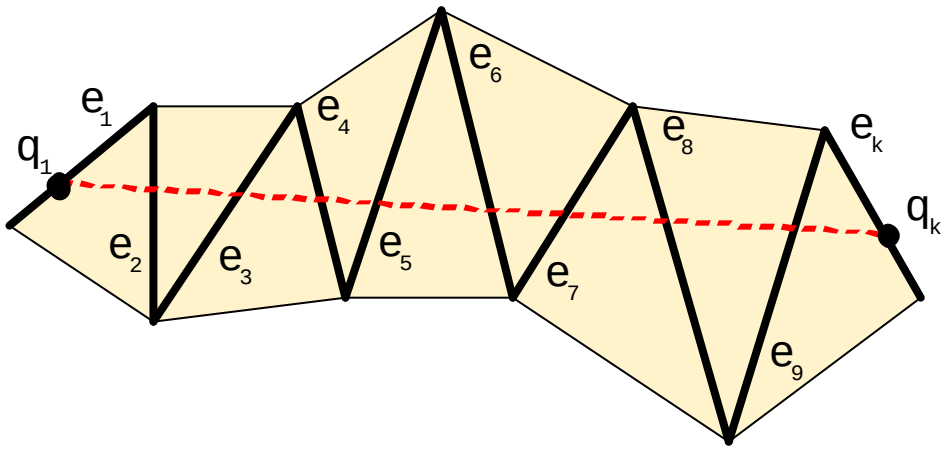

Figure 1.3: A straight-line edge sequence $e_1, e_2, \ldots, e_k$.

1.2.3 Shortest Path Properties

A *shortest path* between two points s and t is defined to be the path of least cost joining s and t, where the cost is determined by a suitable *cost metric* which may pertain to distance, time, energy, number of turns, or some other criteria. The most common cost metric studied is the Euclidean distance (also known as the L_2 metric). Here, a path of least cost is called a *Euclidean shortest path* and would be a path of minimal length. Denote a Euclidean shortest path as $\pi(s,t)$ and $\pi(s,t)$ is composed of a sequence of k straight line segments $s_1, s_2, s_3, ..., s_k$. The cost of this path will be denoted as $|\pi(s,t)| = |s_1| + |s_2| + |s_3| + ... + |s_k|$.

In certain instances, the cost of travel may vary between regions. This travel cost may be a function of the region attributes such as slope, friction coefficients or terrain type (e.g., water, grass, forest, desert, etc.). Let each region be assigned a positive non-zero real *weight*. A *weighted shortest path* is a path in which each segment has a cost which is the length of the segment times the weight of the region through which it passes. Weighted shortest paths will be denoted as $\Pi(s,t)$ with cost

$\|\Pi(s,t)\| = \|s_1\| + \|s_2\| + \|s_3\| + ... + \|s_k\|$, where double vertical bars ($\| \|$) are used to represent a weighted cost of a either a path or a single segment.

The last metric that will be considered is that of finding the *minimal energy path* (or one with minimal power requirements). Such a path must take into account the direction of travel through a face and depends on coefficients of friction as well as face slopes. Denote this path as $\Pi(s,t)$. Note that we use the same notation as the weighted shortest paths.

Consider the case of computing a Euclidean shortest path that remains on the surface of a polyhedron $\mathcal{P}$. Denote by $\text{dist}(s,t)$ the straight line distance between s and t. Define $\text{dist}(x,e)$ as the minimum distance from a point x on $\mathcal{P}$ to a segment e of $\mathcal{P}$. A point $z \in \mathcal{P}$ is defined to be a *ridge point* with respect to some point $s \in \mathcal{P}$ if there exist at least two shortest paths from s to z. Sharir and Schorr [111] presented the following useful properties of $\pi(s,t)$ in the case where $\mathcal{P}$ is convex:

Property 1.4 *$\pi(s,t)$ cannot pass through a vertex or ridge point with respect to s.*

Property 1.5 *The planar unfolding of a Euclidean shortest path is a straight line segment in that plane.*

These additional properties of Sharir and Schorr [111] apply to both convex and non-convex polyhedra:

Property 1.6 *$\pi(s,t)$ cannot cross an edge (or face) of $\mathcal{P}$ more than once.*

Property 1.7 *$\pi(s,t)$ cannot pass through more than n faces.*

Property 1.8 *No Euclidean shortest path on $\mathcal{P}$ intersects itself.*

Property 1.9 *Two Euclidean shortest paths on $\mathcal{P}$ intersect at most once.*

Property 1.10 *A Euclidean shortest path does not bend in the interior of a face.*

Property 1.11 *Given a point p on $\mathcal{P}$, then $|\pi(s,t)| \leq |\pi(s,p)| + |\pi(p,t)|$.*

Consider a weighted polyhedron $\mathcal{P}$ in which a weight $w_i > 0$ is associated with each face $f_i \in \mathcal{P}$ such that the cost of travel through f_i is the distance traveled through f_i times w_i. The weight of an edge of $\mathcal{P}$ is the minimum of the weights of its adjacent faces. Typically, $\|s_i\| = w_i|s_i|$, where $1 < i < k$. Define W and w to be the maximum and minimum of all $w_i, 1 \leq i \leq n$, respectively. Mitchell and Papadimitriou [90] have shown that the following properties hold:

Property 1.12 *An edge of $\mathcal{P}$ cannot have a weight greater than its adjacent faces.*

Property 1.13 *A weighted shortest path obeys Snell's law of refraction in which the path bends at the edges of $\mathcal{P}$. The amount at which it bends depends on the relative weights of the two faces adjacent to this edge (see Figure 1.4a).*

Property 1.14 *A weighted shortest path may critically use (i.e., edge-using) an edge which is cheaper and then reflect back into the face (see Figure 1.4b).*

Property 1.15 *A weighted shortest path $\Pi(s,t)$ may cross a face $\theta(n)$ times and so it may have $\theta(n^2)$ segments (see Figure 1.5). In Figure 1.5, the lightly shaded faces have low cost while the darkly shaded faces have very high cost. As can be seen, a shortest path crossing this area must weave in and out of the low cost regions. This can cause the long thin horizontal faces in the expensive region to be crossed $\theta(n)$ times each.*

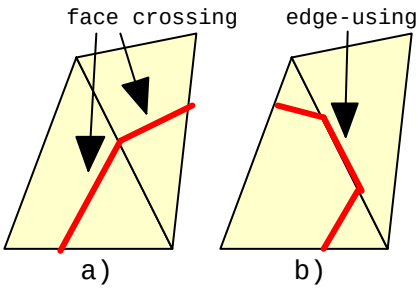

Figure 1.4: Characteristics of a weighted shortest path. a) adjacent face-crossing segments cause a bend in the path, b) edge-using segments cause a reflection back into the same face.

Property 1.16 *Given a point p on $\mathcal{P}$, then* $\|\Pi(s,t)\| \leq \|\Pi(s,p)\| + \|\Pi(p,t)\|$.

We distinguish between two types of path segments of a weighted shortest path:

1) *face-crossing* segments which cross a face and do not critically use an edge, and

2) *edge-using* segments which lie along an edge (critically using it).

In the unweighted domain, edge-using segments span the entire length of an edge in $\mathcal{P}$.

1.2.4 Shortest Path Approximation Factors

The quality of an approximate solution is assessed in comparison to the correct solution. Given a shortest path $\Pi(s,t)$ on a polyhedral surface $\mathcal{P}$, an approximate path $\Pi'(s,t)$ typically has a cost satisfying one (or more) of these three classes of approximations:

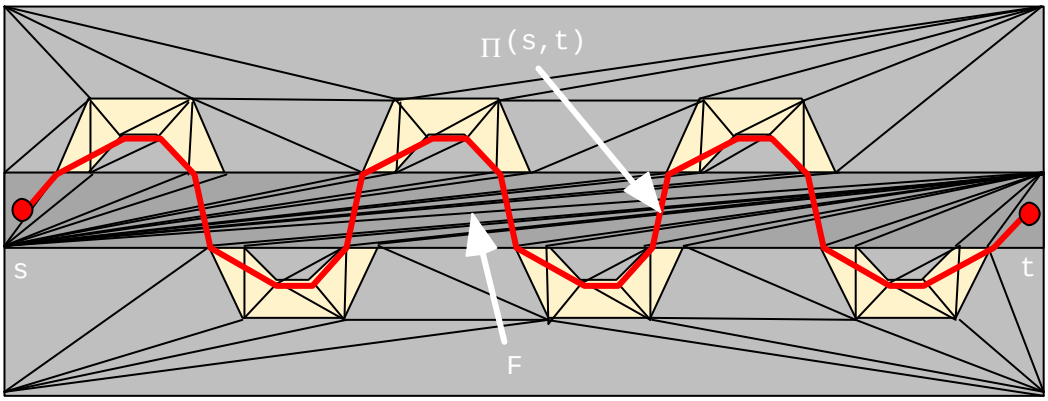

Figure 1.5: A weighted shortest path that can cross a face F, $\theta(n)$ times.

Additive Factor:	$\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + F(\mathcal{P})$
Multiplicative Factor:	$\|\Pi'(s,t)\| \leq F(\mathcal{P})\|\Pi(s,t)\|$
Epsilon Factor:	$\|\Pi'(s,t)\| \leq (1 + F(\mathcal{P},\epsilon))\|\Pi(s,t)\|$

Here, $F(\mathcal{P})$ is typically some function that depends on the size of $\mathcal{P}$ and/or some geometric parameters pertaining to $\mathcal{P}$. For example, in the case of shortest paths on polyhedral surfaces, $F(\mathcal{P})$ may depend on the longest edge length or minimum face angle of $\mathcal{P}$. The last of these three classes is known as an *ϵ-approximation*. An ϵ-approximation algorithm produces a result that is at most $1+\epsilon$ times the optimal solution for some ϵ which typically satisfies $0 < \epsilon < 1$. In ϵ-approximation algorithms, the arbitrarily high accuracy can be traded off against run-time. Such algorithms are appealing and are thus well studied, in particular, from a theoretical view-point.

Let S be a set of points in the plane. Consider a complete graph $G = (V, E)$ where the vertices are the points in S. Clarkson [29] has shown that one can compute a sparse graph H so that any segment joining two vertices in V can be approximated by a path joining the corresponding two vertices in H. A graph H is called a *β-spanner*

of a point G, if any edge e joining two vertices in V can be approximated by a path joining the corresponding vertices in H of length at most $\beta|e|$, where $\beta > 1$ is a constant. The number and the placement of edges in H depends upon the desired accuracy bounds.

1.3 Related Research

As mentioned, shortest path problems in computational geometry can be categorized by factors such as the dimensionality of space, the type and number of objects or obstacles (e.g., polygonal obstacles, convex or non-convex polyhedra, etc.) and the distance measure used (e.g., Euclidean, Manhattan, number of links, weighted distances or anisotropism). There has been a large amount of shortest path research pertaining to these different factors. More recent work has been oriented towards computing approximations of shortest paths. Approximations have the advantage that the algorithms are often simpler and more practical and in some instances (such as the weighted shortest path problem) an exact algorithm may not even be known. A brief survey of the previous research for computing both exact and approximate shortest paths is given here. Several research articles, including surveys, have been written presenting the state-of-the-art in this active field and we refer the interested reader to those (e.g., [92, 93]).

1.3.1 Shortest Paths in Graphs

Perhaps the most commonly studied shortest path problem is that of computing shortest paths in a graph. There are five possible variations of this problem depending on the number of source and destination nodes of G:

- *One-to-one:* Corresponds to computing a shortest path in G from a single source vertex s to a single destination vertex t.
- *One-to-all:* Corresponds to computing a shortest path in G from a single source vertex s to all other vertices.
- *All-to-one:* Corresponds to computing a shortest path in G from all vertices to a single destination vertex t. This is analogous to the one-to-all variation in which the source and destination vertices are reversed.
- *All-to-all:* Corresponds to computing a shortest path in G between every pair of vertices.
- *Few-to-all:* Corresponds to computing a shortest path in G from a constant number of source vertices to all other vertices.

In addition to the type of problem, there may be other issues pertaining to the structure and weights of the edges such as whether or not the edge weights are positive or negative, whether or not the graph is directed and/or acyclic, and the magnitude and type of weights (e.g., small integers, floats etc.). Given here is a brief survey of one-to-all and all-to-all sequential algorithms that have appeared in the literature. Also, we give a brief survey of parallel algorithms for solving this problem.

Probably the most famous of all graph algorithms is Dijkstra's algorithm [38]. Given a directed graph $G(V, E)$ with positive non-zero weights on the edges, Dijkstra's algorithm computes all shortest paths from a fixed source vertex to all other vertices of G. This is done by building up a list of processed vertices (those for which a shortest path has been found). Briefly, the algorithm progressively decreases an estimate on the weight of a shortest path from the source to each vertex of V until it achieves the actual shortest path weight. It repeatedly extracts the next closest vertex to the

source that has not been processed yet and then processes it by *relaxing* along all its incident edges. The technique essentially represents a propagating wavefront of expansion in the graph from the source vertex. The original implementation requires $O(V^2)$ time. Although it is not stated in the algorithm, the vertices to be processed can be stored in a priority queue such as a heap. Fredman and Tarjan [45] showed that if the priority queue of Dijkstra's algorithm is implemented with a Fibonacci Heap, the amortized cost of extracting the next minimum is $O(\log V)$. Hence the running time of their algorithm is $O(V \log V + E)$.

Theorem 1.1 *(Dijkstra [38]) Given two vertices s and t of G, a shortest path in G from s to t can be computed in $O(V^2)$ time.*

Theorem 1.2 *(Fredman and Tarjan [45] or Driscoll et al. [39]) Given two vertices s and t of G, a modification to Dijkstra's algorithm involving Fibonacci heaps [45] or relaxed heaps [39], can be employed to compute a shortest path in G from s to t in $O(V \log V + E)$ time.*

There are many variations of Dijkstra's algorithm which differ in the way vertices are removed from the queue. If the vertex removed is always the vertex with minimum cost, the algorithm is called a *label-setting* algorithm. As the costs are non-negative, each vertex will be removed at most once from the queue. *Label correcting* algorithms do not necessarily remove the vertex with minimum cost and so a vertex may be re-processed many times. In this case, the cost labels are all correct only when the cost of each vertex v is less than the cost of each of its neighbouring vertices u plus the cost from u to v. We denote the vertices that are currently in the queue as being the *active border*.

When the weights of the graph are relatively small integers, more efficient algorithms can be used such as that of Ahuja et al. [7] which runs in $O(E + V\sqrt{\log W})$

time, where W is the largest weight of any edge in the graph. In fact, there has been quite a bit or research in an attempt to improve on the algorithm of Ahuja et al. [7]. This research is shown in Table 1.1. Recent results of Thorup [117] shows that the problem can be solved in $O(E)$ time. The algorithm is based on building a hierarchical bucketing structure.

In the case in which G has negative weights, there may be negative cycles in the graph which are reachable from the source, which of course means that there is no solution. The Bellman-Ford algorithm (based on results of [17] and [42]) handles this special case of weights. It first checks to see if there are negative weight cycles reachable from the source. If not, then it applies a similar solution to that of Dijkstra. It runs in $O(VE)$ time. A more recent algorithm of Gabow and Tarjan [48] runs in $O(\sqrt{V}E\log(VW))$ time, where W is the magnitude of the largest-magnitude weight of any edge in G. Table 1.1 gives a summary of this research for solving the one-to-all (i.e., single source) shortest path problem for graphs.

For the special case of planar graphs, once again, more efficient algorithms can be used. Frederickson [43] was the first to apply the notion of separators to the shortest path problem. The resulting algorithm runs in $O(V\sqrt{\log V})$ time and depends on the fact that planar graphs have separators of size $O(\sqrt{n})$. Henzinger et al. [61] solved this problem in optimal $O(E)$ time, again using separator decompositions. For planar graphs with negative weights, Lipton et al. [84] solved the problem using planar separators in $O(V\sqrt{V})$ time. Goldberg [52] solved the problem in $O(V\sqrt{V}\log W)$ time. Recently, this problem has been solved in $O(V^{4/3}\log(VW))$ time by Henzinger et al. [61].

Consider the problem of computing shortest paths between all pairs of vertices of a graph. Most of the solutions to this problem use an adjacency matrix representation for storing the graph and then apply matrix multiplication. Floyd [41] presented

Run Time	Weights	Reference
$O(V^2)$	positive	[38]
$O(E \log V)$	positive	[38] + [119]
$O(E + V \log V)$	positive	[45]
$O(E\sqrt{\log V})$ randomized	positive int.	[46]
$O(E + \frac{V \log V}{\log \log V})$ randomized	positive int.	[47]
$O(E \log \log V)$	positive int.	[116]
$O(E + V\sqrt{\log V}^{1+\epsilon})$	positive int.	[116]
$O(E + V\sqrt{\log V \log \log V})$	positive int.	[106]
$O(E)$	positive int.	[117]
$O(E + V\sqrt{\log W})$	pos. int. (max. W)	[7]
$O(E + V\sqrt[3]{\log W \log \log W})$	pos. int. (max. W)	[23]
$O(E + V(logW)^{1/4+\epsilon})$ expected	pos. int. (max. W)	[107]
$O(E + V(loglogV)^{1/3+\epsilon})$ expected	pos. int. (max. W)	[107]
$O(VE)$	positive/negative	[17]
$O(\sqrt{V}E \log(VW))$	positive/negative (max. W)	[48]
$O(\sqrt{V}E \log W)$	positive/negative (max. W)	[52]

Table 1.1: Summary of one-to-all graph shortest path algorithms for a graph $G(V, E)$.

an algorithm for solving the all-pairs shortest path problem for graphs with possibly negative weights. It runs in $\theta(V^3)$ time. Johnson [69] solved this problem in $O(V^2 \log V + VE)$ time which can be more efficient if $|E| = O(V \log V)$. The recent work of Copersmith and Winograd[32], Alon et al. [11], [12] and Seidel[113] is dedicated to devising faster algorithms for solving this problem using matrix multiplication with small integer entries. They achieve a time bound of $O(V^{\omega}(polylogV))$ where ω represents the exponent in the running time of the matrix multiplication algorithm used. Currently, the best value of ω is 2.376, due to Copersmith and Winograd[32].

Unfortunately, the algorithms that use matrix multiplication are somewhat impractical since they suffer from large hidden (by the big 'Oh' notation) constants in

the running time bound. Aingworth et al. [8] took a different, combinatorial, approach and produced an approximation algorithm which computes a path with an additive factor of 2. The algorithm requires $O(V^{2.5}\sqrt{\log V})$ time and returns actual paths, not just distances. They claim that their algorithm is easily implemented and the constants are small. However, their results apply to unweighted and undirected graphs.

There has also been work pertaining to computing shortest paths in a graph within the parallel setting. When examining parallel algorithms, it becomes more difficult to compare and contrast them. This is mainly due to the different architectures that the algorithms have been designed for as well as the varying number of processors that are required and/or used by the algorithm.

Kumar et al. [73] showed that the one-to-all graph shortest path problem can be solved on three different architectures using a varying number of processors as shown in Table 1.2. As for the all-pairs-shortest-path problem, Kumar et al. [73] also gave various algorithms. They results are shown in Table 1.3 and clearly indicate the tradeoff between number of processors and running time.

Architecture	Number of Processors	Running Time
Ring	$\Theta(\sqrt{V})$	$\Theta(V\sqrt{V})$
Mesh	$\Theta(V^{2/3})$	$\Theta(V^{4/3})$
Hypercube	$\Theta(V/\log V)$	$\Theta(V \log V)$

Table 1.2: Parallel one-to-all shortest path running times on different architectures.

Pantziou et al. [99] presented an algorithm for solving the all-pairs-shortest-paths problem in planar digraphs by using a parallelization of the hammock decomposition technique of Frederickson [44]. Cohen [31] presented an algorithm which is based on using a separator decomposition. The algorithm is designed for the EREW PRAM

Architecture	Number of Processors	Running Time
Mesh	$\Theta(V^{0.22})$	$\Theta(V^{2.78})$
Mesh	$\Theta(V/\log V)$	$\Theta(V^2 \log V)$
Mesh	$\Theta(V)$	$\Theta(V^2)$
Mesh	$\Theta(V^{1.33})$	$\Theta(V^{1.67})$
Mesh	$\Theta(V^{1.67})$	$\Theta(V^{1.33})$
Mesh	$\Theta(V^2)$	$\Theta(V)$
Hypercube	$\Theta(V/\log V)$	$\Theta(V^2 \log V)$
Hypercube	$\Theta(V)$	$\Theta(V^2)$
Hypercube	$\Theta(V^2/\log^2 V)$	$\Theta(V \log^2 V)$
Hypercube	$\Theta(V^2/\log V)$	$\Theta(V \log V)$
Hypercube	$\Theta(V^2)$	$\Theta(V)$

Table 1.3: Parallel all-pairs-shortest-path algorithm running times on different architectures.

model of computation and requires $O(\log^2 V)$ time preprocessing using $O(V\sqrt{V}\log V)$ work. The algorithm uses $O(\log^2 V)$ time and $O(V^2)$ work to compute only distances but a claim is made that the algorithm can be easily adopted to explicitly find minimum weight paths.

In addition to these algorithms, there has also been a fair amount of research pertaining to distributed parallel graph shortest path algorithms (see [1], [18], [64], [65], [67], [66], [105], [118]). A more thorough discussion of this previous work is given in Chapter 5.

1.3.2 Euclidean Shortest Paths in 2D

In the geometry setting, early shortest path research began with two dimensional objects (i.e., planar surfaces) such as simple polygons, simple polygons with holes, rectilinear polygons and planar subdivisions. Selections from this research which are relevant to our work are discussed here.

1.3.2.1 Within a Simple Polygon

One well studied problem is that of determining a Euclidean shortest path between two points s and t inside a simple polygon with n vertices such that the path remains entirely inside the polygon. The algorithms to solve this problem usually begin with a triangulated polygon. A triangulation of the polygon can be obtained in linear time using the algorithm of Chazelle [20].

Lee and Preparata [81] solved the Euclidean shortest path problem between two points s and t inside a simple polygon. They showed that $\pi(s,t)$ is a polygonal path whose corners are vertices of P and that the union of all paths $\pi(s,v_i)$, where v_i is a vertex of P, is a planar tree rooted at s. This tree is called a *shortest path tree* of P with respect to s. Assume that P is triangulated, and that $\overline{vw}$ is a diagonal as well as the lowest common ancestor of v and w in the shortest path tree being u. The paths $\pi(u,v)$ and $\pi(u,w)$ are outward-convex. In the paper, they defined the union of these two paths to be a *funnel* with *cusp* u. They showed that a *funnel* data structure can be used to represent a shortest path from any point in a sleeve to a diagonal of the sleeve. The funnel consists of four components: a *tail* path, a *cusp* point and two convex chains of vertices forming the funnel borders. The two convex chains meet at the cusp and are joined at their other end by the *funnel diagonal (or lid)* (see Figure 1.6).

The algorithm maintains the funnel structure from s while expanding by one sleeve diagonal at a time until t is reached. The diagonals are processed consecutively (i.e., diagonal $i+1$ shares a face with diagonal i) as it traverses through the dual graph of the triangulation. Each time a sleeve diagonal is encountered, the convex chains on one or both sides of the funnel are updated and the tail may or may not be expanded. After the last diagonal is processed, the processing stops. The path is computed as the funnel tail joined with a single straight line segment from the cusp to t. If the

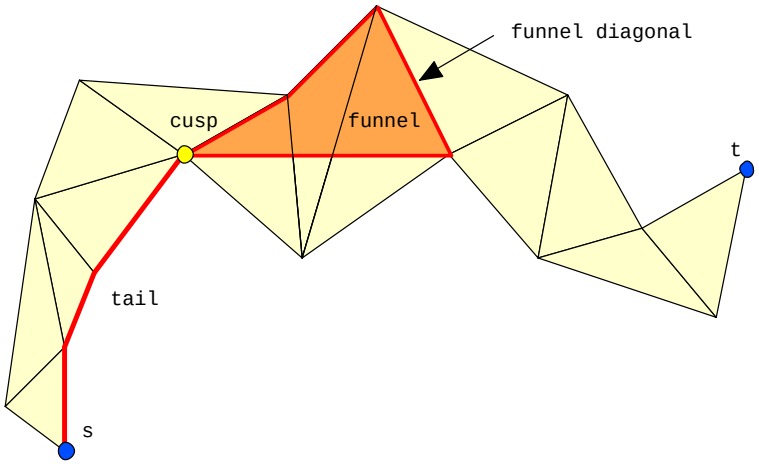

Figure 1.6: The funnel structure.

funnel is stored in a finger search tree, it allows $O(\log \delta)$ time for searching along the two funnel chains and splitting it, where δ is the distance from the search element to the nearest finger. The finger tree has the added advantage that all operations take $O(1)$ amortized time for this application since modifications to the tree are often done close to the fingers. The entire algorithm therefore requires $O(n)$ time.

Guibas et al.[54] presented an optimal linear time algorithm for computing the shortest paths from a source point s to all vertices inside a triangulated simple polygon. They also solved a variety of visibility related problems. Their algorithm essentially calculates a shortest path tree from s in linear time and is based on the results of Lee and Preparata [81].

Guibas and Hershberger [55] presented an optimal algorithm to preprocess the polygon in linear time such that a shortest path between any two query points can be found in $O(\log n)$ time. An actual shortest path can be found in time proportional to the number of path segments. They essentially used a divide and conquer strategy to decompose the polygon into smaller polygons (cells) by continually splitting

the triangulation along diagonals. A binary decomposition tree is kept to store the ordering of the polygonal pieces and it has logarithmic height. They kept for each cell an hourglass representing a shortest path between two bordering diagonals. The merging of two cells requires merging of two hour glasses which they can do in $O(1)$ amortized time. They showed that between any two query points, only $O(\log n)$ pairs of hour glasses need to be merged, and hence they can compute the shortest path length in $O(\log n)$ time.

A parallel algorithm for solving this problem was presented by ElGindy and Goodrich [40]. They solved the point to point problem for an n-vertex polygon in $O(\log n)$ time with $O(n)$ CREW PRAM processors. The algorithm uses a divide and conquer approach which is essentially the same as that of Guibas and Hershberger [55]. They first triangulate the polygon and compute its dual in $O(\log n)$ time. Next, they determine the sleeve of triangles representing the path from s to t in the dual. A shortest path must lie within this sleeve. Let $e_1, e_2, \ldots, e_m$ be the edge sequence of this sleeve. The algorithm uses a divide and conquer approach by finding the median edge, e_k, of the edge sequence and recursively solve the problem for edges $e_1, e_2, \ldots, e_k$ and $e_{k+1}, e_{k+2}, \ldots, e_m$. An hourglass is maintained for each sub-polygon and the merging phase consists of the combining of hour glasses from two adjacent sleeves. They show that there are only a constant number (five) of cases that could occur and the actual merged path is easily constructed in constant time. Thus, the merging can be done in constant time using $O(n)$ processors.

ElGindy and Goodrich [40] also presented an algorithm for computing all shortest paths from a fixed point s within the simple polygon. They solved this problem in $O(\log^2 n)$ time with $O(n)$ processors assuming again a shared memory CREW PRAM model. The algorithm also uses the triangulation and its dual graph. It essentially finds the centroid of the dual tree and performs a divide and conquer on the individual

pieces in parallel.

Goodrich et al. [53] presented an improved parallel algorithm to find a shortest path between two points in a simple polygon. Their algorithm requires $O(\log n)$ time with only $O(n/\log n)$ CREW processors to build a *stratified decomposition tree* which implicitly stores shortest path information. They assume that the polygon has initially been triangulated. Like ElGindy and Goodrich [40], they also use the notion of recursively finding the centroid in the dual of the polygon and combining hour glasses during the merge phase.

1.3.2.2 Among Polygonal Obstacles

There has also been research pertaining to the computation of shortest Euclidean paths in the plane among polygonal obstacles (or within a simple polygon with holes). Among the first algorithms to solve this problem was that of Lozano-Perez and Wesley [85]. They examine the problem of moving a polyhedral object among polyhedral obstacles while avoiding collisions. They use the notion of a *visibility graph* which encompasses the visibility information among the vertices of the polygon and its holes. They showed how to reduce the problem of moving a convex "robot" to that of moving a point among the obstacles by creating a new configuration space representing grown obstacles. Each edge of the graph is assigned a weight according to its Euclidean length. The algorithm of Dijkstra [38] can then be used on the visibility graph to determine a shortest Euclidean path. They do not discuss the time complexity of their algorithm, but it is easily seen that the time complexity is the sum of the times for computing the visibility graph and that of running Dijkstra's algorithm. Ghosh and Mount [51] have shown that the computation of the visibility graph (and hence this shortest path problem) can be done in $O(n \log n + k)$ time where k is the number of edges in the visibility graph.

Kapoor and Maheshwari [70] presented an $O(m^2 \log n + n \log n)$ time algorithm to solve the problem of computing shortest Euclidean paths in the plane among polygonal obstacles where m is the number of obstacles and n is the number of vertices of all obstacles. Their technique also computes the visibility graph through the computation of funnels and corridors (i.e., the area between two convex chains). Since the visibility graph can have $O(n^2)$ edges in worst case, they compute a reduced version containing a subset of the $O(m^2)$ visibility edges and prove that this is sufficient.

The computation of a visibility graph is a common strategy for computing shortest paths in 2D among obstacles even though its size may be quadratic in the worst case. A different approach to solving the problem is that which is termed the *continuous Dijkstra* method. This approach involves propagating a wavefront from s. The wavefront represents all points in the plane that have some fixed distance, say d, from s. The structure of the wavefront changes as it passes certain "event points" such as vertices or edges. It is easily seen that there are at most $O(n)$ such events that may occur. An efficient algorithm that uses this method of propagation must have an efficient technique to predict the occurrences of such events and to be able to process them as they occur. Hershberger and Suri [62] used this continuous Dijkstra approach to solve this shortest path problem in near-optimal $O(n \log n)$ time and $O(n \log n)$ space.

Clarkson [29] took a different approach to this problem and produced an algorithm that generates an ϵ-approximate path. The algorithm takes $O(\frac{n \log n}{\epsilon})$ time to build a data structure so that the path between two query points can be computed in $O(\frac{n}{\epsilon} + n \log n)$ time.

Very recently, Chiang and Mitchell [25] have presented algorithms for solving this problem. They present various methods for solving the problem with query times times such as $o(n)$, $O(\log n + h)$, $O(h \log n)$, $O(\log^2 n)$ and $O(\log n)$ using polynomial

space data structures.

There has also been a significant amount of research in computing rectilinear shortest paths within rectilinear polygons in the plane among rectilinear polygonal obstacles. Several efficient sequential and parallel algorithms exist which solve a variety of these rectilinear shortest path problems. Some of the algorithms build a data structure during a preprocessing stage from a fixed source point and then report shortest k-link paths to a query point in $O(k + \log n)$ time, while others do not have a preprocessing stage and hence cannot answer queries efficiently. The algorithms are designed to produce shortest paths that avoid simple polygonal obstacles, rectangular obstacles or rectilinear polygonal obstacles. Some of the algorithms produce shortest paths that combine shortest Euclidean length as well as the shortest number of links. With these algorithms a constant can usually be varied, producing a shortest Euclidean length path or shortest link path if desired. Table 1.4 gives a summary of the time complexities of various sequential rectilinear shortest path algorithms. In the table, n denotes the number of vertices (or edges) of the obstacles and E denotes the number of extreme edges of the obstacles.

Metric	Simple Poly.	Rectangles	Rectilinear Poly.
L_1 F. Source	$\Omega(n^2)$ [91] $O(n \log n)$ [37]	$O(En + n \log n)$ [123]	$O(n^2)$ [34] $O(En + n \log n)$ [123]
L_1	$O(n^2)$ [80] $\Omega(n \log^2 n)$ [91] $O(n \log^{3/2} n)$ [30]	$O(En + n \log n)$ [123] $O(n \log n)$ [37]	$O(n^2)$ [34] $O(En + n \log n)$ [123] $O(n \log^{3/2} n)$ [30] $O(n \log n + E^2 \log E)$ [120]
L_1 + Link F. Source		$O(En + n \log n)$ [123]	$O(n^2)$ [34] $O(En + n \log n)$ [123]
L_1 + Link		$O(En + n \log n)$ [123]	$O(n^2)$ [34] $O(En + n \log n)$ [123]

Table 1.4: Summary of sequential 2D rectilinear shortest path algorithms

As the table shows, a great deal of work has been done in this area and many of the problems have been efficiently solved. The table only shows sequential algorithms although there are also parallel algorithms. Atallah and Chen [14] presented a parallel algorithm that preprocesses a rectilinear convex polygon with rectilinear holes for rectilinear shortest path queries. They constructed a data structure in $O(\log^2 n)$ time with $O(n^2)$ processors with the CREW PRAM model. They can reduce the number of processors if the source and destination lie on the boundary to $O(n^2/\log^2 n)$ processors. With the data structure, they can produce the length of a shortest path in constant time using one processor, or compute an actual shortest path in logarithmic time using $O(k/\log n)$ processors. They later improved this algorithm [15] by reducing the required number of processors for preprocessing to $O(n^2/\log n)$.

Lingas et al. [83] also presented a parallel algorithm but for a different problem. They solved the problem of determining a shortest path from a vertex of a rectilinear polygon to all other vertices in $O(\log n)$ time using $O(n/\log n)$ processors for the EREW PRAM model. They also solved rectilinear link distance problems. They show that a data structure can be built with a total work of $O(n)$ such that rectilinear link distance queries can be answered in $O(\log n)$ time with a uniprocessor.

1.3.3 Euclidean Shortest Paths in 3D

Due to their relevance in practice, 3-dimensional shortest path problems have received considerable attention. The 3-dimensional Euclidean shortest path problem is stated as follows: Given a set of pairwise disjoint polyhedra in $\Re^3$ and two points s and t, compute a shortest Euclidean path between s and t, that avoids the interiors of the polyhedra. A brief survey of the research is given here for computing exact and approximate shortest paths in the case where the polyhedra are convex.

A special subclass of this problem class is to find a shortest path that remains on the surface of a single polyhedron $\mathcal{P}$ composed of n triangular faces between two points s and t which both lie on $\mathcal{P}$. Some of the algorithms crucially exploit the convexity properties of convex polyhedra and hence may not be extendible to non-convex polyhedra. Clearly, those applicable to non-convex polyhedra apply to convex polyhedra as well, but are typically less efficient since they cannot exploit convexity.

A brief survey of previous research on these two problems is given here. For the second problem (i.e., polyhedral surfaces), we will discuss the research that applies to only convex polyhedra and then that which applies to arbitrary (possibly non-convex) polyhedra. For the convex case, we further divide the research into cases where s is fixed and those which precompute edge sequences.

To our knowledge, our algorithms in Chapter 5 represent the first parallel algorithms for computing weighted shortest paths on convex or non-convex polyhedra. Thus, all algorithms surveyed here are sequential.

1.3.3.1 On a Convex Polyhedron

Consider the problem of computing shortest paths that remain on the surface of a single convex polyhedron $\mathcal{P}$. Sharir and Schorr [111] gave an algorithm for solving this problem that runs in $O(n^3 \log n)$ time where n is the number of vertices of $\mathcal{P}$. After preprocessing, a k-length shortest path from s to any query point $t \in \mathcal{P}$ can be computed in $O(\log n + k)$ time using their data structure of size $O(n^2)$. Their strategy is based on *peeling* the polyhedron with respect to s in a similar fashion to star unfolding (described later). The algorithm first preprocesses the polyhedron by computing the ridge points (ridge lines) with respect to s (see Figure 1.7a). The ridge lines together with the vertices of $\mathcal{P}$ define a tree structure of $O(n^2)$ edges in which

the vertices are at the leaves. Also computed, is the set of shortest paths from s (shortest path tree) to every vertex of $\mathcal{P}$. Figure 1.7b shows this shortest path tree.

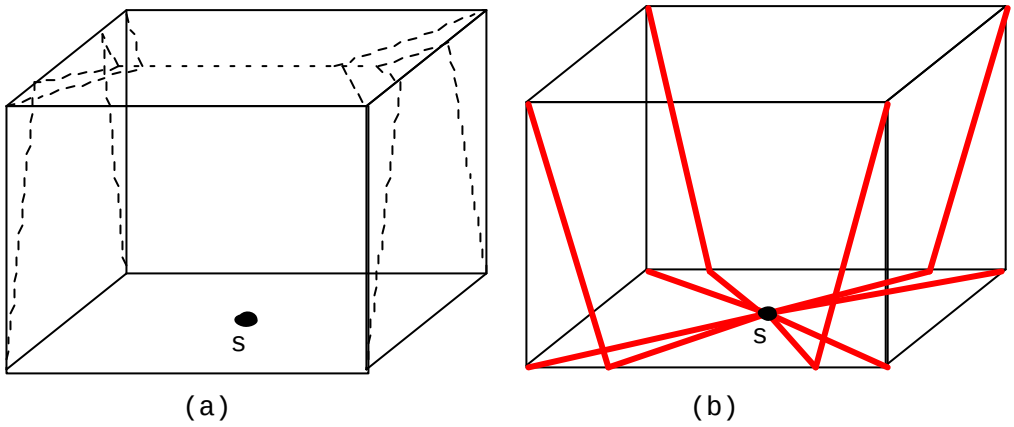

Figure 1.7: a) Ridge lines of a convex polyhedron with respect to s. b) Shortest paths from s to all vertices.

The ridge lines together with these precomputed shortest paths define the boundaries of n regions that they call *peels*. Figure 1.8 shows an example of the resulting peels of the box used in Figure 1.7.

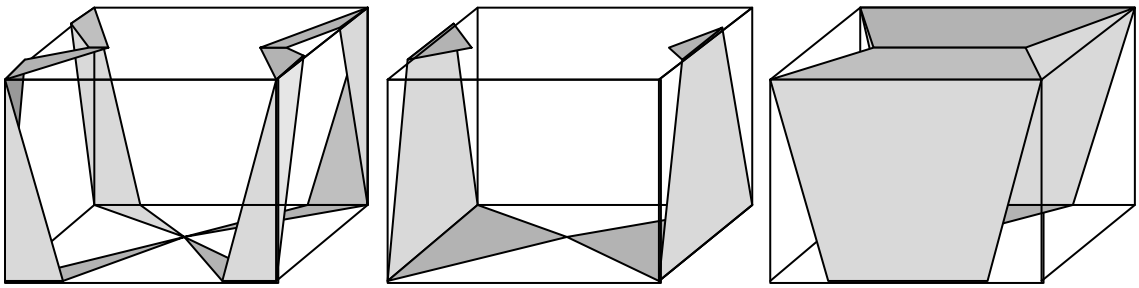

Figure 1.8: The peels resulting from the ridge lines and shortest paths to vertices.

By definition, no vertex or ridge point lies interior to a peel. A shortest path between any two points within a peel is entirely contained within a peel. Since there

can be $O(n)$ regions (*slices*) in a peel, each peel gives rise to $O(n)$ shortest-path edge sequences. Since, there are n peels, there are $O(n^2)$ slices and hence $O(n^2)$ shortest-path edge sequences. Their $O(n^3 \log n)$ time complexity arises from the partitioning of $\mathcal{P}$ into peels. The algorithm builds a *slice tree* of $O(n^2)$ slices, each one taking $O(n \log n)$ insertion time. They essentially propagate outwards from s, building up slices. The expensive slice-adding operation is due to a priority queue which represents the boundary of a wavefront propagating from s. Once the peels are computed, a point location algorithm is used to determine in which slice the destination point lies. Once the slice is determined, so is the peel, and hence a shortest-path edge sequence is known. An actual shortest path can then be computed directly from the unfolded peel in $O(k)$ time, where k is the number of segments in the resulting path. Mount [94] proposed an improvement on this algorithm, reducing it to $O(n^2 \log n)$ by simplifying the storage of the tree structure. Later, the space requirement was reduced by Mount [95] to $O(n \log n)$. O'Rourke et al. [98] then extended the algorithm to work for non-convex polyhedra as well.

In order to reduce the high time complexity of computing the exact solution for this problem, Hershberger and Suri [63] provided an algorithm that computes an approximate shortest path between two query points on a convex polyhedron in linear time. The algorithm is quite simple and is based on taking the half planes passing through the faces containing s and t, computing a path on the surface of these half-planes and then projecting the path back onto the polyhedron. The resulting path has a length of at most two times the length of an actual shortest path.

Har-Peled et al. [57] extended this result and provided an algorithm to compute an ϵ-approximation of a shortest path; it runs in $O(n \min\{1/\epsilon^{1.5}, \log n\} + 1/\epsilon^{4.5} \log(1/\epsilon))$ time. They used the algorithm of Hershberger and Suri [63] to obtain an approximate shortest path. They expanded the polyhedron by a distance $r = \epsilon^{1.5}\delta$, where δ is the

initial distance estimate. They "rounded-off" the new polyhedron by intersecting it with a grid of unit distance proportional to r. The exact path was then computed between two points that are close to s and t and then projected this path back onto the original polytope. Agarwal et al. [4] then improved the algorithm to run in $O(n \log \frac{1}{\epsilon} + \frac{1}{\epsilon^3})$ time. Har-Peled [58] solved the query point version of this problem using the previous results of Agarwal et al. [4]. He provided an algorithm to preprocess the polyhedron in linear time such that two-point shortest path queries could be answered in $O(\frac{\log n}{\epsilon^{1.5}} + \frac{1}{\epsilon^3})$ time. Once again, the resulting path is an ϵ-approximation.

In the case where the source point is fixed, Har-Peled [59] gave an algorithm that computed a shortest path map on a convex polytope $\mathcal{P}$ with respect to a fixed source point s on $\mathcal{P}$. The map is actually a subdivision of $\mathcal{P}$ with size $O(\frac{n}{\epsilon} \log \frac{1}{\epsilon})$ where n is the number of edges of $\mathcal{P}$ and $0 < \epsilon < 1$. The map can be used to answer efficiently approximate shortest path queries and can be computed in $O(\frac{n}{\epsilon^3} \log \frac{1}{\epsilon} + \frac{n}{\epsilon^{1.5}} \log \frac{1}{\epsilon} \log n)$ time. Given a destination query t, the Euclidean length of an ϵ-approximate path on $\mathcal{P}$ from s to t can be computed in $O(\log \frac{n}{\epsilon})$ time.

Computing Edge Sequences

A shortest path between s and t on any convex polyhedron has the property that it will unfold into a straight line. One approach taken by many researchers is to compute a superset of all possible shortest path edge sequences during a preprocessing step. Given this set of edge sequences, the problem is then to choose the appropriate sequence given s and t and hence unfold it to produce a shortest path.

Mount [96] provided an analysis of the number of shortest paths on the surface of a convex polyhedron, although he does not give an algorithm to compute them. He gave a $\Theta(n^4)$ tight bound on the number of possible shortest-path edge sequences. His analysis attempts to associate the shortest path properties with paths in the dual D of the triangulated polyhedral surface. He showed that for any two edges of the

polyhedron, there are at most $O(n^2)$ distinct edge sequences between them that can give rise to a shortest path. Thus, for all pairs of possible edges, there are $O(n^4)$ possible shortest-path edge sequences in total. He started by proving that any set of distinct non-crossing paths in D emanating from a source vertex contains $O(n)$ elements. By considering two particular edges of D, he showed that the removal of certain edges in the polyhedron causes some of the edge sequences to become equal. He showed that for each edge removed, at most $O(n)$ paths can become equal. This represents a reduction in the number of paths. If this is done for each edge, at most $O(n^2)$ paths are removed, showing that there are $O(n^2)$ paths between the two edges. To prove the lower bound of $\Omega(n^4)$, Mount provides an example of a family of polyhedra (see Figure 1.9). At least n^4 different shortest path edge sequences can be formed by varying the location of s and t. Each of these edge sequences consist of three portions: a first portion passing through the upper set of horizontal segments, a second portion passing through the inner set of vertical segments and finally a portion passing through the lower set of horizontal segments.

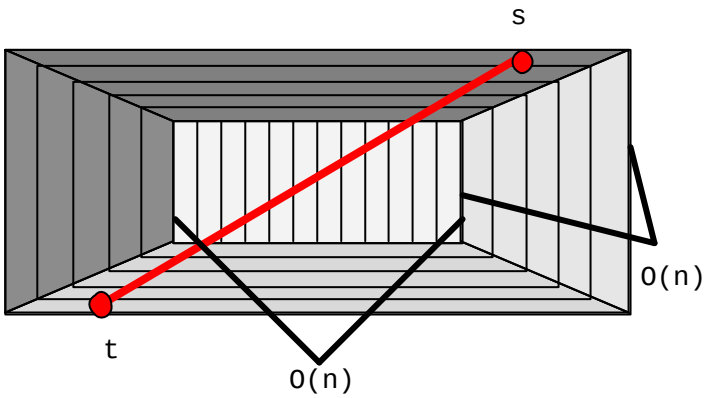

Figure 1.9: A polyhedron that can have $\Omega(n^4)$ shortest path edge sequences.

Sharir [112] gave an algorithm to compute an $O(n^7)$ superset of the possible edge

sequences that runs in $O(n^8 \log n)$ time. He proved the correctness of his algorithm by first examining the case where the two chosen query points, s and t, lie on edges of the polyhedron. If the shortest paths from s and t are computed to every vertex v and paths from s to v and from v to t are combined, then these shortest paths divide the polyhedron into $\Omega(n^2)$ simply connected regions. Figure 1.10 gives an example of a polyhedral cone in which two edges on the base face exhibit these $\Omega(n^2)$ regions.

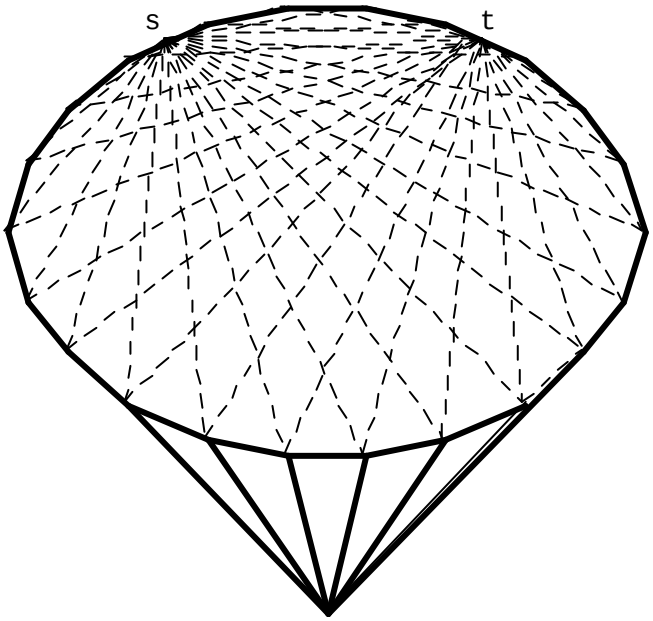

Figure 1.10: The $\Omega(n^2)$ regions produced on the base of a cone between two points s and t.

He showed that a shortest path from s to t must lie completely within one of these regions. In fact, he proved that a shortest path between these two points cannot cross any of the shortest paths from s or t to the vertices. Hence, there are only $O(n)$ of the $O(n^2)$ regions that can contain a shortest path between s and t as shown in Figure 1.11. Furthermore, a path through each of these regions can cross at most $O(n)$ edges.

If s and t are varied slightly along their edges, the edge sequences themselves do

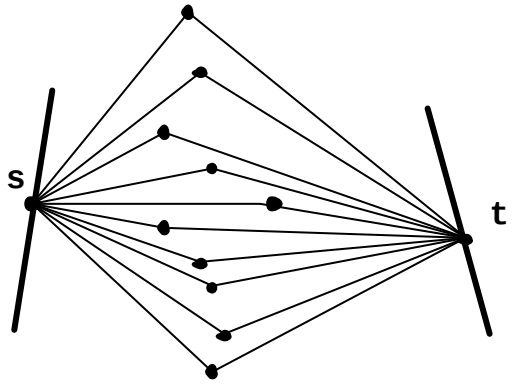

Figure 1.11: The $O(n)$ possible regions allowing a shortest path between s and t.

not change. He showed that the edge sequences will only vary when s and t cross what he calls *critical points*. The critical points are defined on each edge by the ridge lines with respect to each vertex. Since each vertex can produce $O(n)$ ridge points on each edge, there are at most $O(n^2)$ critical points on an edge which divide the edge into closed intervals in which the edge sequence remains the same for any point in that interval. If an interval is chosen on the edge containing s and one on the edge containing t, there are a total of $O(n^4)$ possible combinations. Since there are $O(n)$ edge sequences for any two intervals, this produces $O(n^5)$ possible edge sequences in total for a specific pair of source and destination edges. It can be easily seen that for all of the $O(n^2)$ pairs of possible source/destination edges, there is a total of $O(n^7)$ edge sequences. An $O(n^8 \log n)$ time algorithm is given to compute these $O(n^7)$ edge sequences.

Agarwal et al. [3] improved upon the work of Sharir [112] by providing an algorithm to compute $O(n^6)$ edge sequences in $O(n^6)$ time using the notion of *star unfolding*. A convex polyhedron K can be *unfolded* with respect to some non-ridge point x on its surface. Consider the shortest paths from x to all vertices of K. The

segments of these shortest paths are called *cut lines*. These cut lines together with the edges of K induce a decomposition of K into *cells* which are convex polygons. A 2D complex is formed from the cells by isometrically embedding the cells into the plane. The resulting 2D complex is called a star unfolding (see Figure 1.12). The star unfolding exhibits the following interesting properties:

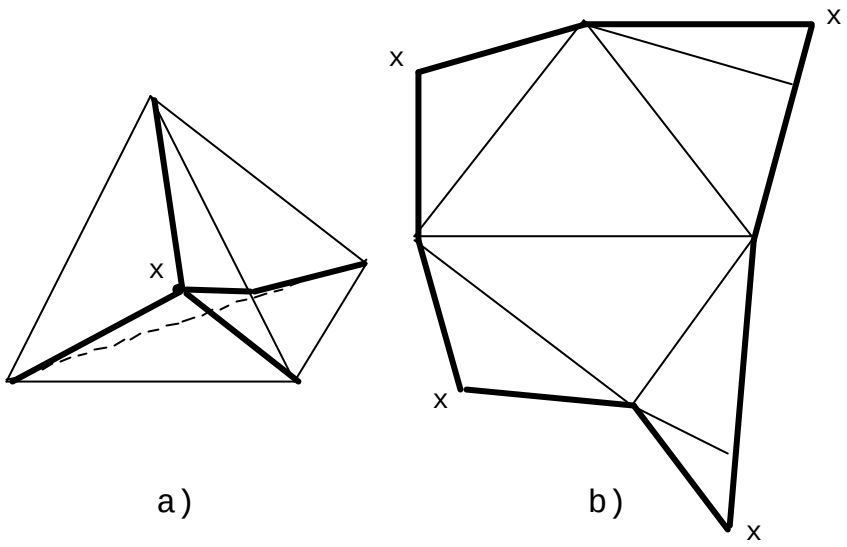

Figure 1.12: a) The shortest paths on a convex polyhedron from a point x to each vertex. b) The star unfolding with respect to x.

Property 1.17 *The boundary of a star unfolding is comprised solely of cut lines.*

Property 1.18 *A star unfolding can have $\Theta(n^2)$ convex regions in the worst case.*

They used the dual D of the star unfolding with respect to a non-ridge point x. This dual graph represents a tree of at most $O(n^2)$ vertices where the leaves (at most n) of D correspond to regions containing x. From each image of x in the unfolding, there are $O(n^2)$ possible edge sequences; one from x to each other vertex of D. Since

there are $O(n)$ images of x, there are a total of $O(n^3)$ edge sequences from x. Consider the distinct dual trees of unfoldings that arise from non-ridge points on edges of P. The ridge lines with respect to a particular vertex of P, can intersect an edge of P in $O(n)$ points. Since there are $O(n)$ vertices and $O(n)$ edges, the edges of P are partitioned into $O(n^3)$ open segments after computing the ridge lines for each vertex. Each of these segments are free of ridge points. The paper showed that all points within one of these segments share the same set of edge sequences. By computing the star unfolding with respect to a selected point in each of these sub-segments, $O(n^6)$ edge sequences are induced since each unfolding produces $O(n^3)$ edge sequences. The algorithm produces an edge sequence tree with $O(n)$ depth and $O(n^6)$ nodes. A later paper by Aronov and O'Rourke [13] proved the non-overlapping property of the star unfolding and helped provide a little more intuition by showing that the ridge tree of x is nothing but the Voronoi diagram of the star-unfolded images of x.

Schevon and O'Rourke [110] did provide an algorithm to produce $O(n^4)$ edge sequences but their algorithm requires $O(n^9 \log n)$ time and $O(n^8)$ space. Furthermore, after this preprocessing step, their algorithm can report the edge sequences traversed by all shortest paths connecting a given pair of query points lying on edges of the polyhedron in logarithmic time. Agarwal et al. [5] then provided a modification to their algorithm of [3] so as to compute the exact set of at most $O(n^4)$ edge sequences in $O(n^6 \beta(n) \log n)$ time, where $\beta(n)$ is an extremely slow growing function (uses inverse Ackerman function).

1.3.3.2 Among Multiple Convex Polyhedra

The problem of finding a shortest path from a point on one convex polyhedron to a point on another disjoint polyhedron has been studied by Baltsan and Sharir [16]. In their paper they showed that a shortest path in such a scenario consists of 3 portions:

a shortest path on the starting polyhedron from the source to a *take off* point, a straight line path from the take off point to some *landing* point on the destination polyhedron, and finally, a shortest path from the landing point to the destination. Figure 1.13 shows an example of the structure of a shortest path between three polyhedra in which there are two takeoff (t_1, t_2) and two landing (l_1, l_2 points).

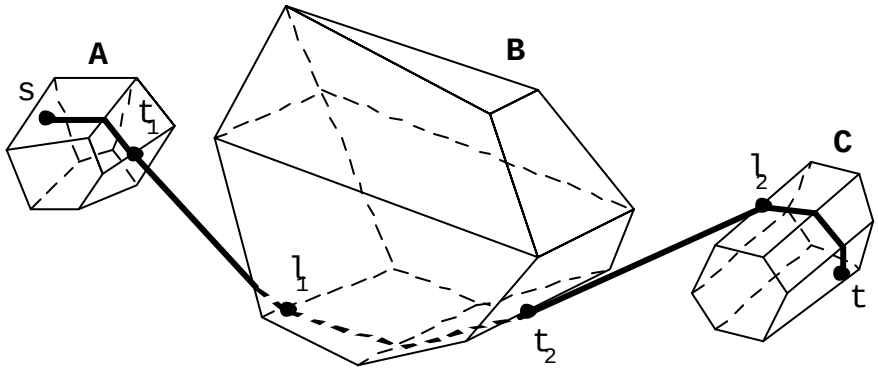

Figure 1.13: The structure of a shortest path between three polyhedra.

Calculation of the takeoff and landing points essentially solves the problem since this leaves us with an already solved problem of computing a shortest path between two points on a convex polyhedron, which has already been discussed. They solve this problem in $O(n^3 * 2^{O(\alpha(n)^4)} \log n)$ time where $\alpha(n)$ is the inverse of the Ackerman function. They begin by partitioning the start and end polyhedra into slices as done by Mount [94]. The next step is to determine the two sequences of edges that are passed through on the first and last portions of a shortest path. Since the ridge lines split the polyhedral edges into at most n intervals each, there are at most $O(n^2)$ intervals per polyhedron. The basic idea of determining takeoff and landing points is to try all pairs of these ridge-defined edge intervals (one from each polyhedron) for a total of $O(n^4)$ pairs. They showed however, that there are only $O(n^2)$ pairs of edges that are not totally obscured from one another. Their algorithm reduces the

number of candidate pairs by examining the portions of edges that are visible from each other. For each pair, the sequence of edges lying in the same peel as the chosen intervals are found and the chosen pair can be ruled out if the shortest path does not remain within the peel.

Sharir [112] also addressed the problem of computing a shortest path amidst a fixed number k of convex polyhedra having altogether n vertices. He claimed that this problem produces $O(n^{7k})$ possible edge sequences and can be solved in $O(n^{O(k)})$ time. He does not address the problem of determining which of the convex polyhedra are intersected by a shortest path from the source to destination, assuming he is already given these polyhedra. He then showed that a shortest path is nothing but subpaths that alternate between shortest paths on a polyhedron and straight line segments that join these shortest paths between two polyhedra. Again, the problem essentially involves choosing appropriate takeoff and landing points for each polyhedron. These points define edge sequences and therefore, his technique for computing edge sequences on a single polyhedron is also applicable to this case of multiple polyhedra.

Papadimitriou [100] provided an algorithm to compute ϵ-approximations of shortest paths amidst polyhedral obstacles in 3-space. The algorithm runs in $O(n^4(L + \log(n/\epsilon))^2/\epsilon^2)$ time, where n is the number of elements (vertices, faces and edges) of all polyhedra and L represents the bit precision. The algorithm divides the edges of the polyhedra into intervals based on a geometric progression which depends on ϵ. Choi et al. [26] later presented a refinement to this algorithm. In follow-up work, Choi et al. [27] took a closer look at the precision issues (bit complexity) of their earlier work in [26]. They examined the low level operations of their algorithm in order to determine the complexity with respect to precision.

Clarkson [29] also provided an algorithm to compute an ϵ-approximation for

paths amidst polyhedral obstacles in 3-space which runs in $O(n^2\lambda(n)\log(n/\epsilon)/\epsilon^4 + n^2 \log np \log(n \log p))$ time. Here, p is the ratio of the length of the longest obstacle edge to the distance between s and t. The function $\lambda(n) = \alpha(n)^{O(\alpha(n)^{O(1)})}$, where $\alpha(n)$ is a form of inverse of Ackermann's function.

1.3.3.3 On a Non-Convex Polyhedron

The approach of pre-computing edge sequences becomes impractical when the polyhedron is non-convex. Although Mount [96] has shown that there are in the worst case $\Theta(n^4)$ shortest-path edge sequences for an n-vertex convex polyhedron, this does not hold for non-convex polyhedra. The main problem is that shortest paths on non-convex polyhedra may pass through vertices and hence does not necessarily unfold into straight lines. Figure 1.14 shows an example of a non-convex polyhedron which can lead to an exponential number of possible shortest path edge sequences. To see this, think of each peak as being a "pencil tip". The path may go left or right around each pencil tip, leading to $\Theta(2^k)$ possible shortest path edge sequences, where k is the number of peaks.

The computation of Euclidean shortest paths on non-convex polyhedra has been investigated by [71, 21, 88, 98, 6]. Two of these algorithms ([21],[88]) use the notion of a *projection*. Consider an unfolded sleeve of triangles in the plane, representing a particular shortest-path edge sequence. When the face containing the source point is unfolded onto the plane, the point representing the position of s is called the *image* of s, and it is denoted here as s'. A *projection* of s' through a sleeve onto an edge e_i is essentially, the portion of edge e_i that is visible from s'. The projection is denoted as $proj(s', e_i)$. The projection together with the image of the source point essentially define a cone in which an unfolded shortest path from s' to e_i must lie. A further projection through e_i casts a *shadow* on one or both of the remaining two edges of an

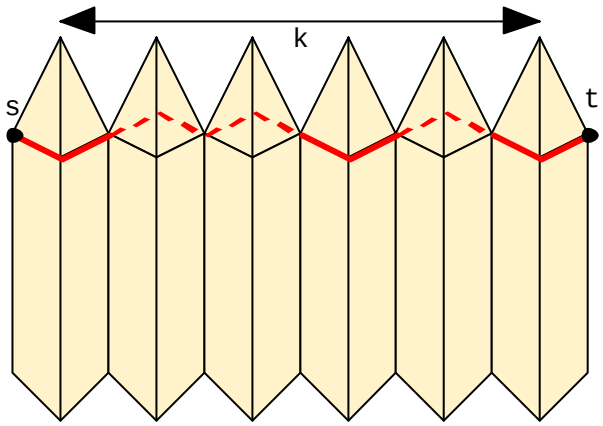

Figure 1.14: An example in which there may be $\Theta(2^k)$ possible shortest path edge sequences between two points.

outside face. This shadow is essentially one or two new projections representing the updated visibility from s'. Figure 1.15 shows an example of a projection of a source image s' onto an edge e_i and its shadow projections.

Mitchell et al. [88] provided an $O(n^2 \log n)$ algorithm for computing shortest paths on a non-convex polyhedron which also used $O(n^2)$ space. Their approach uses the continuous Dijkstra technique. As the wavefront expands from s, it keeps for each edge a set of projections (intervals of optimality) emanating from s' (i.e., the unfolded image of s) through a sleeve. Once completed, each edge contains up to n intervals of optimality. The path from the source to each projection represents a straight-line edge sequence. However, since the projections are propagated in increasing order of their distance from s, only shortest-path edge sequences are produced. A shortest path of k segments to any query point can then be found in $O(k + \log n)$ time.

Chen and Han [21] presented an algorithm to compute a shortest path on a convex polyhedron from a fixed source point s. Their algorithm preprocesses the polyhedron

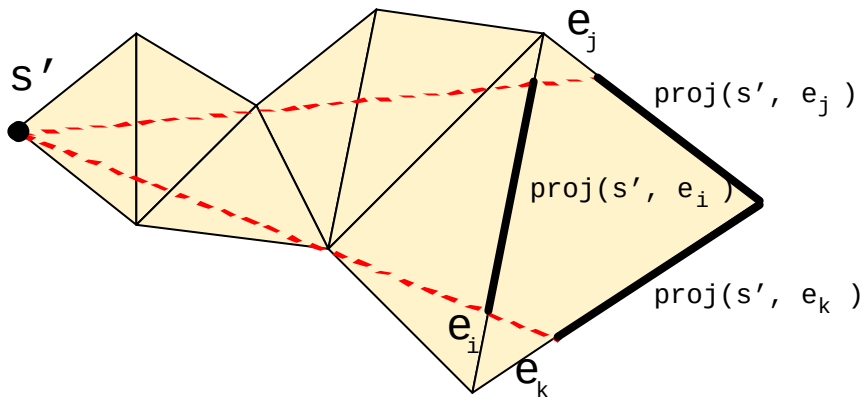

Figure 1.15: The projection of a source image s' onto an edge e_i and its shadow projections.

in $O(n^2)$ time and $O(n)$ space. The algorithm essentially unfolds faces of the polyhedron with respect to s such that only sleeves representing shortest-path edge sequences are produced. Each sleeve is equivalent to a path in the dual of the polyhedron emanating from the same source vertex. The algorithm begins by propagating outwards from s through its adjacent triangles. During propagation, each encountered edge maintains the image s' of s and its corresponding projection through the sleeve (see again Figure 1.15). Each new projection is created from an expansion of an edge sequence by one edge. Only sleeves which produce non-empty projections of the image are candidates for expansion. Figure 1.16 shows the first three stages of the algorithm which produce edge sequences of sizes 1,2 and 3. The shaded cones represent valid shortest path regions.

To avoid an exponential number of unfoldings, a crucial lemma is given which reduces the number of shortest-path unfoldings to be of linear size. Consider the triangle $\triangle ABC$ with two projections on edge $\overline{BC}$ emanating from two separate source images I_1 and I_2 (see Figure 1.17). Assume that both projections have shadows

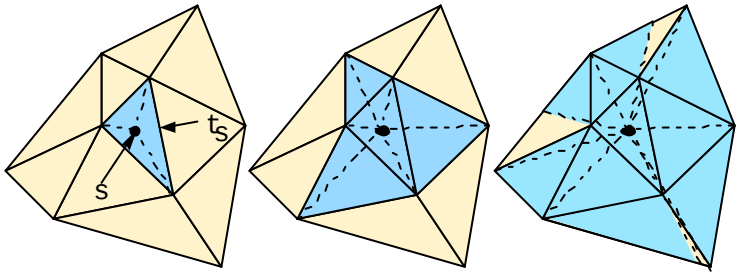

Figure 1.16: The first three stages of the algorithm of Chen and Han. Shaded cones represent possible unfolded shortest path sleeves from s.

covering A. Let d_i be the distance from image I_i, where $i = 1, 2$ and without loss of generality let $d_2 < d_1$. The lemma shows that although the shadow of both projections

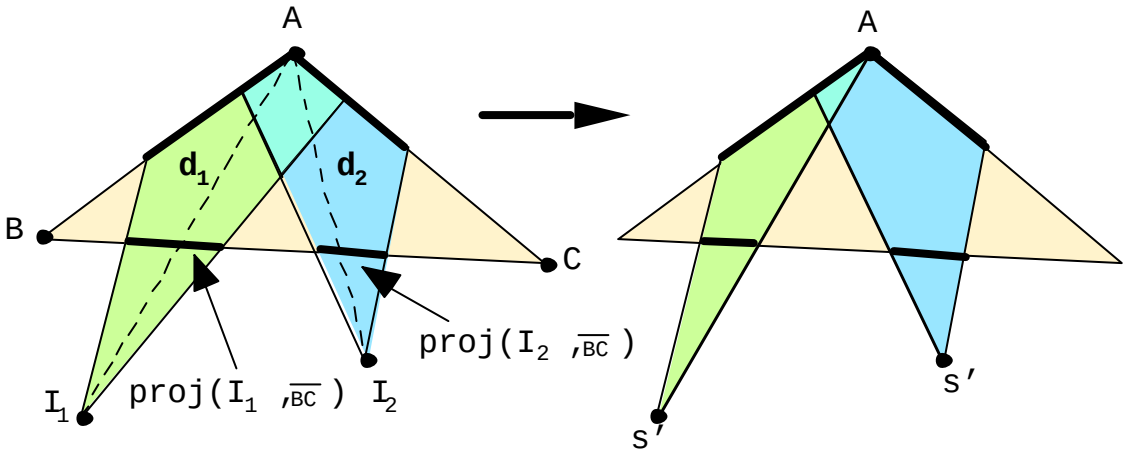

Figure 1.17: The crucial "one-angle-one-split" lemma of the Chen and Han algorithm.

covers A, only one of the projections will allow further expansion through both edges $\overline{BA}$ and $\overline{AC}$. Since it has been assumed that $d_2 < d_1$, then the path from I_2 to A is shorter than the path from I_1 to A. Consider paths from I_1 and I_2 to points on $\overline{AC}$. It can be shown that these points are closer to I_2 than I_1 and so the projection from

I_1 will not propagate through $\overline{AC}$. The projection from I_2 may however propagate through $\overline{BA}$ and $\overline{AC}$. In the case where $d_2 = d_1$, then I_1 would propagate through only $\overline{BA}$ and I_2 through only $\overline{AC}$. An alternate proof of this lemma follows from Agarwal et al. [3].

Chen and Han's algorithm builds a *sequence tree* representing the edge sequences with the root being the face containing s. A downward path from s in the tree represents a sleeve of triangles in which there exists a straight line segment which passes through each face of the sleeve (i.e., a shortest-path edge sequence). To keep the tree linear in size, only leaves and interior nodes (representing polyhedral faces) that have more than one child are kept. A shortest path to a query point t is found by searching the tree for a path containing the shortest of all the edge sequences from s to t. This can be done in $O(k)$ time for a k-link shortest path. Chen and Han state that their technique and lemma can be slightly modified to handle shortest paths on a non-convex polyhedra, although no direct proof is given [2]. For non-convex polyhedra, they take into account paths that go through vertices of the polyhedron. A path going from s to t through a vertex v is viewed as two separate sub-paths from s to v and from v to t. They modify their tree to have two kinds of nodes: a *vertex* node and an *edge* node. Vertices (hence vertex nodes) essentially act as pseudo sources for which propagation occurs in a similar manner to s. Projections keep track of their source image (which may be an intermediate vertex). They show that the sequence tree still has at most $O(n)$ leaves at any time during the algorithm and that the time and space complexities do not change.

Very recently, Kapoor [71] provided a more time-efficient algorithm for solving this problem. His algorithm requires only $O(n \log^2 n)$ time. The algorithm is based on a method which is termed the "waveform propagation" method. This method is based

[2]Some researchers in the field doubt the correctness of Chen and Han's algorithm for the non-convex case.

on maintaining a sequence of arcs of circles with centers as vertices of the polyhedron. As with the work of Mitchell et al. [88], the waveform is expanded and changes only at certain event points. The algorithm uses unfolding techniques which are similar to that of Chen and Han [21] and also uses a hierarchical convex hull structure which is crucial to obtain efficient determination of events points.

Varadarajan and Agarwal [6] provided an algorithm that computed a path on a, possibly non-convex, polyhedron that has at most $7(1+\epsilon)$ times the shortest path length; it runs in $O(n^{5/3}\log^{5/3} n)$ time. They also presented a slightly faster algorithm (i.e., $O(n^{8/5}\log^{8/5} n)$ time) that returns a path which is at most $15(1+\epsilon)$ times the shortest path length. They partitioned $\mathcal{P}$ into $O(n/r)$ regions, each with at most r faces. This is done using the notion of planar separators such that there are at most $\sqrt{r}$ border vertices per region. The algorithm computes shortest path approximations among all bordering vertices of region and then merges all of these such graphs. Dijkstra's algorithm is then applied to the merged graph to obtain their approximation.

In the case where the source point is fixed, Har-Peled [59] gave an algorithm that computed a shortest path map on a polyhedral surface $\mathcal{P}$ with respect to a fixed source point s on $\mathcal{P}$. The map is actually a subdivision of $\mathcal{P}$ with size $O(\frac{n}{\epsilon}\log\frac{1}{\epsilon})$ where n is the number of edges of $\mathcal{P}$ and $0<\epsilon<1$. The map can be used to answer efficiently approximate shortest path queries and can be computed in $O(n^2\log n+\frac{n}{\epsilon}\log\frac{1}{\epsilon}\log\frac{n}{\epsilon})$ time. Note that this computation of the map has a better time complexity in the case where $\mathcal{P}$ is convex. Given a destination query t, the Euclidean length of an ϵ-approximate path on $\mathcal{P}$ from s to t can be computed in $O(\log\frac{n}{\epsilon})$ time.

1.3.4 Weighted Shortest Paths in 2D and 3D

Consider a triangulated planar subdivision $\mathcal{S}$. The problem of determining a Euclidean shortest path between any two points can be solved using similar algorithms to those for simple polygons. If however, each face of $\mathcal{S}$ is assigned a weight denoted by a real number $w_i > 0$ and the weighted shortest path metric is used, then the problem becomes quite different.

As Mitchell [89] has shown, a weighted shortest path can bend at each diagonal. In his paper, he examined the problems that are faced during optimal path computations when taking into account the weights of the faces. He made use of *Snell's Law* which defines precisely how a shortest path should move from face to face depending on the uniform weights that are assigned to the faces. He showed that there are three possible *bendings* that a shortest path can take when encountering an edge between two faces. Figure 1.18 shows the three cases. Let α denote the weight of the face that a shortest path is leaving and let β denote the weight of the face that the shortest path is entering. Let $\alpha < \beta$ and let ϕ be the angle of incidence and θ be the angle of refraction. In the refraction case, the path is *refracted* and the path obeys $\alpha \sin\theta = \beta \sin\phi$. He showed that the angle of incidence of a shortest path cannot be greater (in absolute value) than the *critical angle* defined at that edge. The critical angle from a face f with weight α_f to a face f' with weight $\alpha_{f'}$ is denoted as $\theta a_c(f, f')$ and is defined as $\theta a_c(f, f') = \sin^{-1}(\alpha_{f'}/\alpha_f)$.

If a shortest path from f hits the edge at the critical angle, then it may critically use the edge and be *critically reflected* back into f. This can only occur if $\alpha_{f'} < \alpha_f$. If the edge is allowed to have a weight that is different to that of its adjacent faces, then a shortest path that hits the edge at the incoming critical angle can critically use the edge and then exit into face f' at the outgoing critical angle.

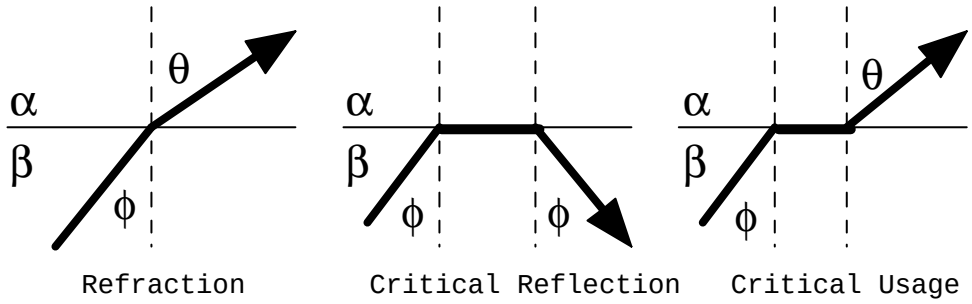

Figure 1.18: The three types of bending properties of a shortest path intersecting an edge.

Mitchell and Papadimitriou [90] presented an algorithm to solve the shortest path problem in a weighted planar subdivision $\mathcal{S}$ from a fixed source point (known as the weighted region problem). The algorithm computes a path which has a cost at most $(1+\epsilon)$ times the shortest weighted path cost, where $0 < \epsilon < 1$. The algorithm requires $O(n^8 L)$ time in the worst case, where $L = \log(nNW/w\epsilon)$ is a factor representing the bit complexity of the problem instance. Here N is the largest integer coordinate of any vertex of the triangulation and W (respectively, w) is the maximum (respectively, minimum) weight of any face of the triangulation. They claim however, that the algorithm would perform much better in practice, however, they did not complete their implementation. The algorithm is based on the continuous Dijkstra technique of Mitchell et al. [88] and also takes into account the various refraction and reflection characteristics. The algorithm actually computes *intervals of optimality* for each edge with respect to a fixed source point. These intervals define the intersections of channels which emanate from the source point. The channels represent passage ways such that all shortest paths from the source point that remain in a particular channel pass through the same sequence of edges. The algorithm begins by propagating from

the source point to the edges of the face in which it is contained. This defines three channels. For each edge of that face, the point with minimum cost (say x_{min}) to the source is computed. Propagation then proceeds by expanding outwards from the edge which has the closest x_{min} to the source. Upon expansion, channels are split into two and its boundaries are recomputed. The shape of the channel boundaries is determined by Snell's law of refraction. Expansion continues in this manner until all vertices of the subdivision are reached. There are special cases involving propagation from a vertex as well as handling paths that are reflected back into a face. The algorithmic details are non-trivial and technical and are therefore omitted here.

Gewali et al. [50] presented an algorithm for a specialized case of the weighted region problem in which the planar subdivision consists of regions with weights either 0, 1 or ∞. Their algorithm takes $O(n^2)$ time and is based on computing a specialized visibility graph. This specialization ensures that 0-weighted regions can be entered from a vertex or perpendicular to an edge of the region. The graph can also be built in time $O(n \log n + K)$ time, where K is the number of edges of the 0-weighted regions.

In the rectilinear setting, there has also been some work in computing shortest weighted rectilinear paths, where rectilinear obstacles have weights associated with them. In this scenario, the shortest rectilinear paths are allowed to pass through a weighted obstacle with a cost combining the distance and weight metric. Table 1.5 gives a summary of the time complexities of various sequential rectilinear shortest weighted path algorithms. In the table, n denotes the number of vertices (or edges) of the obstacles.

In 3D, there has been substantially less research for developing algorithms to compute weighted shortest paths. According to Mitchell and Papadimitriou [90], their work pertaining to the weighted region problem can be modified to work for non-convex polyhedra since it is based on the continuous Dijkstra approach of Mitchell et

Metric	Weighted Rect.
L_1 F. Source	$O(n \log^{3/2} n)$ [22]
L_1	$O(n^2)$ [124] $O(n \log^{3/2} n)$ [22] $O(n \log^{3/2} n)$ [82] $O(n \log n)$ [122]
L_1 + Link	$O(n^2)$ [124]

Table 1.5: Summary of sequential 2D rectilinear weighted shortest path algorithms.

al. [88]. However, the large time complexity of roughly $O(n^8 \log n)$ motivates study for more efficient solutions.

Mata and Mitchell [86] presented independently and at the same time as our work (see [76]) an alternate algorithm that constructs a graph which can be searched to obtain an approximate path; their path accuracy is $(1 + \frac{W}{kw\theta_{min}})$, where θ_{min} is the minimum angle of any face of $\mathcal{P}$, W/w is the largest to smallest weight ratio and k is a constant that depends upon ϵ. Their time complexity is $O(kn^3)$. Their approach is based on discretizing the number of possible orientations (i.e., k) that a shortest path can take from the source point. To determine the boundaries of each of the *cones* created during this process, they determine a path to another vertex (or critical point) by tracing out the path through the cone, obeying Snell's law of refraction along the polyhedral edges. The cones are built using an outward propagation strategy. At certain points during the propagation, cones are split into two thinner cones. Once propagation is complete, the result is called a *pathnet* and it is searched using Dijkstra's algorithm for a path from s to t.

In three dimensions, the problem of determining a shortest rectilinear path among a set of boxes has also received some attention. Clarkson et al. [30] presented an

algorithm to solve this problem in $O(n^2(\log n)^3)$ time. M. deBerg et al. [35] also solved this problem, but they use a combined L_1 and link distance metric. In addition, their algorithm is generalized to work for higher dimensions. Their running time is $O(n^3 \log n)$ which is based on the building of a data structure. If preprocessing is allowed, then they can answer shortest path queries in $O(\log^2 n + k)$ time for a *k-link path*. Choi and Yap [28] presented an improved algorithm that requires only $O(n^2 \log n)$ preprocessing and shortest path queries can be answered in $O(\log n + k)$ time.

1.3.5 Point Location Algorithms

Shortest path algorithms that allow source/destination queries either require knowledge of the face(s) containing the query point(s) or point location algorithms to locate the face containing the query point(s). The following observation follows from [101].

Observation 1.1 *Given a polyhedron $\mathcal{P}$ with $O(n)$ faces, a data structure can be built in $O(n^2)$ time such that for a given query point q on $\mathcal{P}$, we can determine the face of $\mathcal{P}$ in which q lies in $O(\log n)$ time. In the special case in which $\mathcal{P}$ is a terrain, more efficient planar point location algorithms can be used (e.g., see [74]).*

1.4 Contributions

In order to help compare the results of the work presented in this thesis to the previous work, Table 1.6 shows the work pertaining to computing shortest paths on arbitrary (i.e., possibly non-convex) polyhedral surfaces. Our work focuses on the weighted shortest path metric. As can be seen from the table, the work presented in this thesis, improves upon the running time of existing algorithms in terms of n.

Run Time	Metric	Accuracy	Reference	New
$O(n^2 \log n)$	Euclidean	Exact	[88]	-
$O(n^2)$	Euclidean	Exact	[21]	-
$O(n \log^2 n)$	Euclidean	Exact	[71]	-
$O(n^{5/3} \log^{5/3} n)$	Euclidean	$7(1+\epsilon)\|\Pi\|$	[6]	-
$O(n^{8/5} \log^{8/5} n)$	Euclidean	$15(1+\epsilon)\|\Pi\|$	[6]	-
$O(n^8 \log n)$	Weighted	$(1+\epsilon)\|\Pi\|$	[90]	-
$O(kn^3)$	Weighted	$(1+\frac{W}{kw\theta_{min}})\|\Pi\|$	[86]	-
$O(n^5)$	Weighted	$\|\Pi\| + W\|L\|$	[76], [77], [79]	Ch. 2
$O(n^3 \log n)$	Weighted	$\beta\|\Pi\| + \beta W\|L\|$	[76], [77], [79]	Ch. 2
$O(mn \log mn + nm^2)$	Weighted	$(1+\epsilon)\|\Pi\|$	[9], [10]	Ch. 3
$O(n^n)$	Anisotropic	Exact	[108]	-
$O(nk \log nk + nk^2)$	Anisotropic	$\frac{\|\Pi\|}{\sin(\alpha/2)} + \frac{W\|L\|}{\sin(\alpha/2)}$	[78]	Ch. 4
$O(nk \log nk + nk^2)$	Anisotropic	$(1+\frac{3W\epsilon}{w \sin(\alpha/2)}\epsilon)\|\Pi\|$	[78]	Ch. 4

Table 1.6: Summary of shortest path algorithms on arbitrary polyhedral surfaces.

In Chapter 2 we present an algorithm for computing weighted shortest paths on polyhedral surfaces and provide an experimental analysis to validate its usefulness. To our knowledge, this work provides the first algorithm for computing approximations of weighted shortest paths on arbitrary polyhedral surfaces [1]. This work was initially developed as a technical report in [75]. It was presented in both paper [76] and video [77] form and finally submitted to a journal [79]. In the chapter we provide a worst case theoretical run time analysis and verify its usefulness through experimentation with geographical terrain data. Variations on the algorithm are also described.

In Chapter 3 we provide a theoretical improvement on the algorithm that is capable of generating an ϵ-approximate solution to the same problem. The approach is quite similar to that in Chapter 2, and the majority of the chapter is used to provide a theoretical analysis of both path accuracy and running time. This work was first developed as a technical report for the special case in which the source and destination

[1] The work of Mata and Mitchell [86] was developed at the same time.

are vertices of the polyhedron. We presented an extension to the algorithm which allowed arbitrary queries in [10]. Our vertex-to-vertex work (i.e., the technical report) was published later in [9]. Although the ϵ-approximation algorithms of Mitchell and Papadimitriou [90] and Mata and Mitchel [86] had existed at this time, our aim was to provide an algorithm that reduced the run-time dependency on n.

In order to provide an algorithm that was capable of producing a more realistic cost measure, we investigated the problem of computing shortest energy paths that take into account the direction of travel along the polyhedral face inclines. This work was presented in Chapter 4 which uses a cost metric based on the model of Rowe and Ross [108]. Although Rowe and Ross had previously investigated this problem, the exponential time complexity of their algorithm made this problem a good candidate for further research. It was our intent to reduce dramatically the dependence on n in the runtime of the algorithm. The approach we take expands on that of Chapters 2 and 3. This work has been presented recently in [78].

Due to the large size of the terrain data sets that we have encountered, a limit arose pertaining to the maximum terrain size that the implementation could handle. This was mainly due to the limit of available internal and virtual memory of the test machine. In addition, the large data sizes can result in large running times. To help alleviate the storage constraints and runtime delays, a parallel algorithm was developed. In fact, since our work is based on decomposing the shortest path problem into a graph problem, there are many possible parallel algorithms that apply to our techniques. Chapter 5 investigates a parallel simulation algorithm for computing a weighted shortest path on polyhedral surfaces. The work can be applied to all of our algorithms in the previous chapters. Experimental results are presented for various types of terrain data. We show that a key factor in the performance of the parallel algorithm is in the decomposition of the terrain data (i.e., the partitioning).

Although there has been extensive work in computing shortest paths in parallel for transportation networks, there has been very little in computing shortest paths on non-grid-like terrains. It was our intent to show that all of the work in the previous chapters can be parallelized and that efficient speedup can be obtained.

Finally, Chapter 6 attempts to briefly summarize the results of the thesis and present a few open problems as well as describe work that is currently being done.

Chapter 2

Algorithms Based on Edge Decomposition

Our approach to solving the weighted shortest path problem is to discretize the input polyhedron in a natural way, by placing Steiner points along the edges of the polyhedron. We construct a graph containing the Steiner points as vertices and edges as the interconnections between Steiner points that correspond to segments which lie completely in the triangular faces of the polyhedron. We describe here the design and analysis of several schemes for placing the Steiner points and edges on each face of the polyhedron. The key idea is to transform geometric shortest path problems into graph-theoretic problems so that the following goals are achievable:

1. provable bounds on the approximation factors and the path cost can be established,
2. experimentally verifiable high quality paths can be obtained,
3. existing shortest path graph algorithms can be used,

4. the schemes are easily implementable, and appealing to practitioners.

We point out that, in any of our schemes, we can store the graph as it pertains to Steiner points implicitly. In an iteration of Dijkstra's algorithm, adjacency information can be computed on the fly at a small additional cost. Thus, the graph needs neither to be precomputed nor to be stored.

In addition to the theoretical contributions, we have implemented all of our schemes, a point-to-point shortest Euclidean path algorithm for sleeves, graph spanners on faces of polyhedra and Chen and Han's algorithm. We performed extensive experiments on triangular irregular networks (TINs) and established excellent performance of the schemes in path quality and run time; both being better than the theoretical worst case bounds. We show experimentally that a constant number of Steiner points (i.e., six) per edge suffice implying an $O(n \log n)$ run time as was also observed experimentally. In the unweighted case, a direct comparison to Chen and Han's algorithm is given. Here our schemes show a fast convergence to optimal in accuracy with a much improved running time over Chen and Han. Using an additional post-processing step the exact shortest paths are frequently obtained. In the weighted scenario, as far as we are aware of, this work represents the first adequately documented implementation [3]. Here also the path accuracy convergences rapidly at a fast running time. We conclude that the schemes presented here are of high value for practitioners from inside and outside the scientific community and for researchers in the above mentioned fields.

The chapter is organized as follows: Section 2.1 describes our approximation algorithms for shortest paths computations. We begin with a simple approach and then describe improved schemes. Section 2.2 presents the testing procedures, experimental setup and results that were obtained. Section 2.3 discusses an extension to our

[3] At the same time as this work [76], an alternate approach of [86] was proposed.

algorithm to accommodate arbitrary query points as well as an analysis for achieving ϵ-approximations when constraints are set for s and t. Section 2.4 discusses possible future work in the form of other approximation schemes.

2.1 Shortest Path Approximation Schemes

In this section, we describe various *schemes* for approximating Euclidean or weighted shortest paths between vertices on the surface of a polyhedron $\mathcal{P}$. All schemes are based on adding Steiner points to the edges of $\mathcal{P}$ and then building a graph on $\mathcal{P}$ from which the approximation will be a subgraph. First we describe a simple approach and show that it may produce poor approximations. Then we describe two edge-decomposition schemes and provide an analysis of the approximation quality.

2.1.1 A Simple Approximation Scheme

A natural approach to approximating paths on a given polyhedron $\mathcal{P}$ is to choose a path which is restricted to traveling along edges of $\mathcal{P}$. That is, one could compute a graph G as follows. The vertices in G correspond to the vertices of $\mathcal{P}$ and there is an edge between two vertices in G if the corresponding vertices in $\mathcal{P}$ are connected by a polyhedral edge. The weight of an edge in G is the Euclidean length of the corresponding edge in $\mathcal{P}$ times the minimum weight of its two adjacent faces. For simplicity, assume that s and t are vertices of $\mathcal{P}$. An approximate shortest path $\Pi'(s,t)$, between s and t can then be computed by using Theorem 1.2.

Since this scheme confines the path to traveling on edges of $\mathcal{P}$, the quality of the approximation depends on the given triangulation, which could be bad in the worst case (as is illustrated in Figure 2.1). Assume that the faces of the polyhedron

in Figure 2.1 have equal weight. If $\mathcal{P}$ is sufficiently compressed horizontally then $|\Pi(s,t)| \to 0$. Each segment of a shortest path is approximated by an excursion to a vertex and back to another vertex, resulting in an unnecessary traversal of distance $|L|$. The approximate path $\Pi'(s,t)$ with k links has length $k \cdot |L|$ and this could be much larger than $|\Pi(s,t)|$.

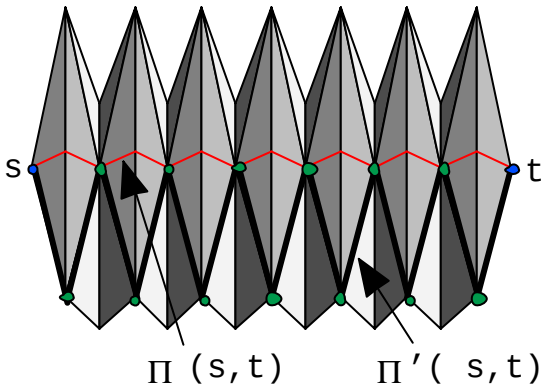

Figure 2.1: The approximated path length $|\Pi'(s,t)|$ is unbounded with respect to $|\Pi(s,t)|$.

Although this method of approximation can be bad in worst case, we can bound the path cost with respect to parameters of $\mathcal{P}$. We give here a bound on this type of path which depends on θ_{min} (i.e., the minimum angle between any two adjacent edges of $\mathcal{P}$).

Property 2.1 *Let ABC be a triangle. Let X be a point on $\overline{AC}$ and Y be a point on $\overline{AB}$ such that $|\overline{AX}| = \delta|\overline{AC}|$ and $|\overline{AY}| = \delta|\overline{AB}|$ for some $0 < \delta < 1$. Then $\overline{XY}$ is parallel to $\overline{BC}$. Furthermore, $|\overline{XY}| = \delta|\overline{BC}|$.*

Claim 2.1 *Let ABC be a triangle such that $\angle CAB = \theta$. Let $\overline{XY}$ be a line segment with endpoints X and Y lying on $\overline{AC}$ and $\overline{AB}$, respectively. Given some constant $0 < \delta < 1$, the following is true:*

i) If $|\overline{AX}| \geq \delta|\overline{AC}|$ *and* $|\overline{AY}| \leq \delta|\overline{AB}|$ *then* $|\overline{AC}| \leq \frac{|\overline{XY}|}{\delta \sin\theta}$

ii) If $|\overline{AX}| \leq \delta|\overline{AC}|$ *and* $|\overline{AY}| \geq \delta|\overline{AB}|$ *then* $|\overline{AB}| \leq \frac{|\overline{XY}|}{\delta \sin\theta}$ *(symmetrical to i above)*

iii) If $|\overline{AX}| \geq \delta|\overline{AC}|$ *and* $|\overline{AY}| \geq \delta|\overline{AB}|$ *then* $|\overline{BC}| \leq \frac{|\overline{XY}|}{\delta}$

Proof: Let P be the point on line AB such that $|\overline{XP}|$ is minimized (i.e., $\overline{XP}$ is a perpendicular of line AB which may or may not lie on segment $\overline{AB}$). Define Y' to be the point on segment $\overline{AB}$ such that $|\overline{XY'}|$ is minimized ($Y' = A, P$, or B). Since $|\overline{XY'}| \leq |\overline{XY}|$, it is sufficient to prove the claim for $|\overline{XY'}|$. Lastly, let D and E be the points on $\overline{AC}$ and $\overline{AB}$, respectively, such that $|\overline{AD}| = \delta|\overline{AC}|$ and $|\overline{AE}| = \delta|\overline{AB}|$. (See Figure 2.2 where δ is set to $\frac{1}{2}$).

i) (case ii is symmetrical) By definition $|\overline{XY}| \geq |\overline{XY'}| \geq |\overline{XP}|$. If $Y' = A$ then we are done since by assumption $|\overline{AX}| \geq \delta|\overline{AC}|$. Hence $|\overline{AC}| \leq \frac{|\overline{XY}|}{\delta}$. If $Y' = P$, then $\sin\theta = \frac{|\overline{XP}|}{|\overline{AX}|}$ and from our assumption that $|\overline{AX}| \geq \delta|\overline{AC}|$ we have $|\overline{AC}| \leq \frac{|\overline{XP}|}{\delta \sin\theta} \leq \frac{|\overline{XY}|}{\delta \sin\theta}$.

iii) From Property 2.1 $|\overline{DE}| = \delta|\overline{CB}|$. Due to the constraint that $|\overline{AX}| \geq \delta|\overline{AC}|$ and $|\overline{AY}| \geq \delta|\overline{AB}|$ then we can infer that $|\overline{XY}| \geq |\overline{ED}| = \delta|\overline{CB}|$. Therefore, $|\overline{CB}| \leq \frac{|\overline{XY}|}{\delta}$.

□

Lemma 2.1 *A Euclidean shortest path* $\pi(s,t)$ *between two vertices* s *and* t *of* $\mathcal{P}$ *can be approximated by a path* $\pi'(s,t)$ *consisting only of edges of* $\mathcal{P}$ *such that* $|\pi'(s,t)| \leq \frac{2}{\sin\theta_{min}}|\pi(s,t)|$.

Proof: We will show that for each segment s_i of $\pi(s,t)$ passing through face f_j $(1 \leq j \leq n)$, there is a corresponding segment s_i' of $\pi'(s,t)$ which represents an edge of $\mathcal{P}$ and is bounded by $\frac{2}{\sin\theta_j}|s_i|$, where θ_j is the minimum angle between any pair

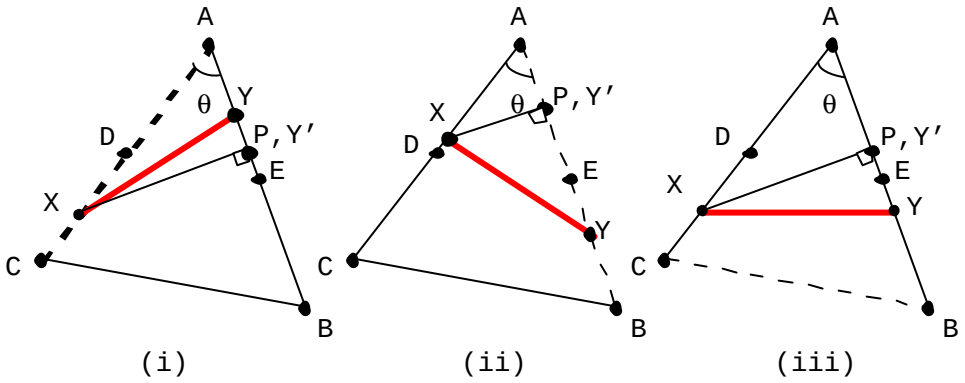

Figure 2.2: The three cases in which an edge of $\triangle ABC$ (shown dashed) has length bounded by a segment $\overline{XY}$ crossing the triangle.

of edges of f_j. Let $s_i = \overline{xy}$ and $s_{i+1} = \overline{yz}$ be two consecutive segments of $\pi(s,t)$. Let x,y and z lie on edges e_x, e_y and e_z, respectively. Also let v_x, v_y and v_z be the vertices of e_x, e_y and e_z that are closest to x,y and z, respectively. Claim 2.1 ensures that $|\overline{v_x v_y}| \leq \frac{|xy|}{\frac{1}{2}\sin\theta_j} \leq \frac{|xy|}{\frac{1}{2}\sin\theta_{min}}$ and that $|\overline{v_y v_z}| \leq \frac{|yz|}{\frac{1}{2}\sin\theta_j} \leq \frac{|yz|}{\frac{1}{2}\sin\theta_{min}}$. Let $s'_i = \overline{v_x v_y}$ and $s'_{i+1} = \overline{v_y v_z}$ be segments of $\pi'(s,t)$ that approximate s_i and s_{i+1}, respectively. This bound applies to every segment $|s'_i|$ of $\pi'(s,t)$ such that $|s'_i| \leq (\frac{2}{\sin\theta_{min}})|s_i|$. Since the cost of $\pi(s,t)$ is the sum of the cost of its segments, then $|\pi'(s,t)| \leq (\frac{2}{\sin\theta_{min}})|\pi(s,t)|$. Connectivity is ensured since every pair of consecutive segments of $\pi'(s,t)$ share a vertex.

□

Lemma 2.2 *A shortest cost path $\Pi(s,t)$ between two vertices s and t of a weighted polyhedral surface $\mathcal{P}$ can be approximated by a path $\Pi'(s,t)$ consisting only of edges of $\mathcal{P}$ such that $\|\Pi'(s,t)\| \leq (\frac{2}{\sin\theta_{min}})\|\Pi(s,t)\|$.*

Proof: We have shown through Lemma 2.1 that the length of an approximation segment s'_i that passes through a face is at most $\frac{2}{\sin\theta_{min}}$ times the actual shortest

path segment length. In addition, s_i' is an edge of $\mathcal{P}$. From Property 1.12, s_i' cannot have a weight greater than the weight of its adjacent faces since it would be cheaper to travel "just inside" the face than along the edge. Hence, for any segment s_i of $\Pi(s,t)$ that passes through a face, we approximate it with a segment s_i' of $\Pi'(s,t)$ with cost at most $(\frac{2}{\sin\theta_{min}})\|s_i\|$. If there are no reflected segments in $\Pi(s,t)$, then the bound of Lemma 2.1 also applies in the weighted case since edges of the terrain cannot have a cost higher than its incident faces. We must now show that if paths are reflected, then the bound also holds.

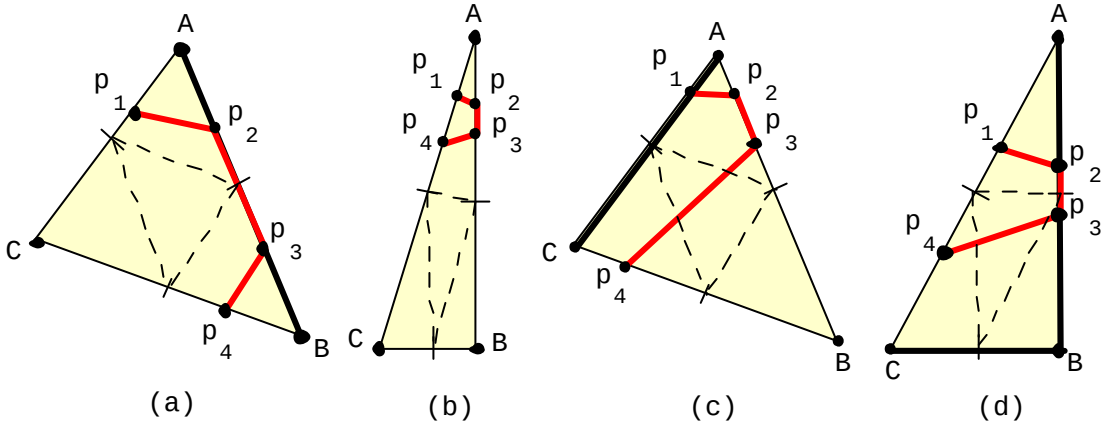

Figure 2.3: The four cases in which a weighted path can reflect back into a face.

Let $\triangle ABC$ be a face of $\mathcal{P}$. Let p_1, p_2, p_3 and p_4 be consecutive points of $\Pi(s,t)$ joining consecutive segments of $\Pi(s,t)$. Without loss of generality, let p_1 lie on $\overline{AC}$, p_2, p_3 lie on $\overline{AB}$ and p_4 lie on either $\overline{AC}$ or $\overline{BC}$. In addition, we will assume that $|\overline{Ap_1}| < |\overline{Cp_1}|$. (we can apply a similar proof when $|\overline{Ap_1}| \geq |\overline{Cp_1}|$. Also, a similar proof can be constructed for when p_2 and p_3 lie on $\overline{CB}$ while p_4 lies on $\overline{AB}$). Let M_{ab}, M_{ac} and M_{bc} be the midpoints of $\overline{AB}$, $\overline{AC}$ and $\overline{BC}$, respectively. We will examine four cases which differ in the way that the subpath $\Pi(p_1,p_4)$ intersects these line segments forming $\triangle M_{ab}M_{ac}M_{bc}$ as shown in Figure 2.3. We will prove for each of these cases that the

subpath $\Pi'(p_1, p_4)$ produced, by dragging these 4 points to the closest vertex of the edge on which they lie, will be bounded such that $\|\Pi'(p_1, p_4)\| \leq (\frac{2}{\sin\theta_{min}})\|\Pi(p_1, p_4)\|$.

Consider the case of Figure 2.3a in which p_1 lies on $\overline{AM_{ac}}$, p_4 lies on $\overline{BM_{bc}}$ and both p_2 and p_3 lie on $\overline{AB}$. Here $\Pi(p_1, p_4)$ intersects the two segments of $\triangle M_{ab}M_{ac}M_{bc}$ that share M_{ab}. Let $\Pi'(p_1, p_4) = \overline{AB}$. Since $\overline{AB}$ necessarily has weight cheaper than the face (Property 1.12), then the result of Claim 2.1 proves that $|\overline{AB}| \leq \frac{2}{\sin\theta}|\overline{p_1p_4}| \leq \frac{2}{\sin\theta_{min}}|\overline{p_1p_4}|$. By triangle inequality, $|\overline{p_1p_4}| \leq |\overline{p_1p_2}| + |\overline{p_2p_3}| + |\overline{p_3p_4}|$ and hence $|\overline{AB}| \leq \frac{2}{\sin\theta_{min}}|\Pi(p_1, p_4)|$. Since $|\overline{AB}|$ has weight which is no more than any portion of $\Pi(p_1, p_4)$ then $\|\Pi'(p_1, p_4)\| = \|\overline{AB}\| \leq \frac{2}{\sin\theta_{min}}\|\Pi(p_1, p_4)\|$.

Consider the second case (Figure 2.3b) in which both p_1 and p_4 lie on $\overline{AM_{ac}}$ and hence $\Pi(p_1, p_4)$ does not intersect $\triangle M_{ab}M_{ac}M_{bc}$. This represents a degenerate case in which $\|\Pi(p_1, p_4)\|$ has no corresponding approximation segment. We can assume that this portion of the path never "leaves" A and hence has a cost of zero. Thus, $\|\Pi'(p_1, p_4)\| = 0 < \|\Pi(p_1, p_4)\|$.

The third case as shown in Figure 2.3c where $\Pi(p_1, p_4)$ intersects two segments of $\triangle M_{ab}M_{ac}M_{bc}$, and one of them is $\overline{M_{ac}M_{bc}}$. Let $\Pi'(p_1, p_4) = \overline{AC}$. By applying Claim 2.1 one can show that $|\overline{AC}| \leq \frac{2}{\sin\theta}|\overline{p_3p_4}|$, where $\theta = \angle ABC$. Since $\overline{p_3p_4}$ is internal to the face and $\|\overline{p_3p_4}\| < \|\Pi(p_1, p_4)\|$, then $\|\overline{AC}\| \leq \frac{2}{\sin\theta_{min}}\|\overline{p_3p_4}\|$.

The last case (see Figure 2.3d) is when $\Pi(p_1, p_4)$ intersects all three segments of $\triangle M_{ab}M_{ac}M_{bc}$. Here, let $\Pi'(p_1, p_4) = \overline{AB}, \overline{BC}$ (i.e., the approximated path has two segments). Claim 2.1 ensures that $|\overline{BC}| \leq \frac{2}{\sin\theta_{min}}|\overline{p_3p_4}|$ and since $\overline{p_3p_4}$ is internal to the face then, $\|\overline{BC}\| \leq \frac{2}{\sin\theta_{min}}\|\overline{p_3p_4}\|$. As for $|\overline{AB}|$, we make use of the fact that $|\overline{p_1p_3}| < |\overline{p_1p_2}| + |\overline{p_2p_3}|$, and bound $|\overline{AB}|$ with respect to $|\overline{p_1p_3}|$ by again using Claim 2.1. Once again, the cheapest weight is along edge $|\overline{AB}|$ and so $\|\overline{AB}\| \leq \frac{2}{\sin\theta_{min}}\|p_1p_3\|$. Hence $\|\Pi'(p_1, p_4)\| = \|\overline{AB}\| + \|\overline{BC}\| \leq \frac{2}{\sin\theta_{min}}\|\Pi(p_1, p_4)\|$.

□

In the following theorem we present an upper bound on the approximation quality obtained by using the simple approach.

Theorem 2.1 *A shortest cost path $\Pi(s,t)$ between two vertices s and t of a weighted polyhedral surface $\mathcal{P}$ with n faces can be approximated by a path $\Pi'(s,t)$ such that $\|\Pi'(s,t)\| \leq (\frac{2}{\sin\theta_{min}})\|\Pi(s,t)\|$, where θ_{min} is the minimum interior angle of any face of $\mathcal{P}$. Furthermore, $\|\Pi'(s,t)\|$ can be computed in $O(n \log n)$ time.*

Proof: The proof of the accuracy bound follows from Lemma 2.2. The time complexity follows from Theorem 1.2 since $\mathcal{P}$ is planar (i.e., $|E| = O(|V|)$).

□

Although we provided an approximation bound, the worst case can be bad, especially due to the dependency on θ_{min}. In general, the polyhedron may have thin triangles resulting in a small value of θ_{min}. Hence this algorithm may produce a path with unreasonable accuracy. For example, if $\theta_{min} = 5^\circ$ then $\frac{2}{\sin\theta_{min}} \approx 23$. Our bound is an upper bound and assumes that every edge of $\|\Pi'(s,t)\|$ has this worst case approximation factor, which is very unlikely in practice. Nevertheless, to avoid this pessimistic bound, we investigated a different scheme of edge decomposition that does not depend on the geometric parameter of θ_{min}.

2.1.2 Edge Decomposition Schemes

We describe here two different schemes for placing Steiner points along the edges of $\mathcal{P}$ and describe how they are interconnected on each face to form each G_i using one of two interconnection strategies. We then derive bounds on the portion of a shortest path going through a face and prove the stated approximation bounds for paths computed by the respective schemes. We do this by showing the existence of a

path approximating a shortest path within the stated bounds. The result of applying Dijkstra's algorithm may be this path or a path with equal or better cost.

2.1.2.1 Building the Graph

We begin by placing Steiner points evenly along each edge of $\mathcal{P}$ using one of two placement schemes:

- *Fixed* placement - exactly m Steiner points are placed along the edge.
- *Interval* placement - place only enough (up to m) Steiner points such that the distance between adjacent Steiner points on an edge is at most $\frac{|L|}{m+1}$. (Note that if only $m-1$ Steiner points are placed on the edge, then the distance between adjacent Steiner points would be greater than $\frac{|L|}{m+1}$).

For each face $f_i, 1 \leq i \leq n$ of $\mathcal{P}$, compute a face graph G_i as follows. The Steiner points, along with the original vertices of f_i, become vertices of G_i, denoted as V_i. Connect a vertex pair v_a, v_b of V_i to form an edge $\overline{v_a v_b}$ of G_i if v_a and v_b represent points that are adjacent on the same edge of f_i. We then provide further interconnection among the vertices in V_i by using exactly one of two connection schemes:

- *Complete* interconnection - connect a vertex pair $v_a, v_b \in V_i$ to form an edge $\overline{v_a v_b}$ of G_i if and only if v_a and v_b represent Steiner points that lie on different edges of f_i.
- *Spanner* interconnection - Let $\mathcal{C}$ be the set of planar (with respect to f_i) cones with apex at all $v_a \in V_i$ and the conical angle $\theta = \frac{\pi}{\rho}$ for an integer constant $\rho > 4$. For each $v_a \in V_i$ perform a radial sweep of the elements of V_i. During this sweep determine the vertex $v_b \in V_i$ that has minimal distance to v_a in each

of the ρ cones and add edge $\overline{v_a v_b}$ to G_i. Note that if two consecutive cones share the same v_b, then only one of the $\overline{v_a v_b}$ edges is added to G_i.

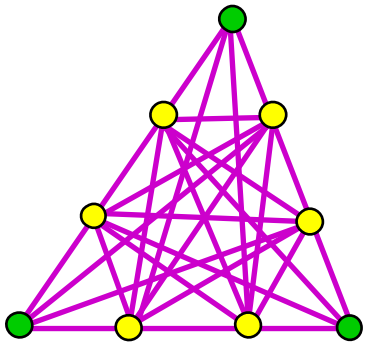

Figure 2.4: Adding Steiner points and edges to a face using the complete interconnection scheme.

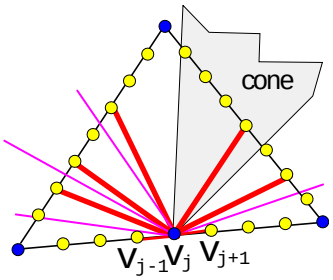

Figure 2.5: The spanner edges added from a vertex v_j with $\theta = 30°$.

For an example of creating G_i using the fixed placement scheme and complete interconnection strategy, Figure 2.4 shows how six Steiner points ($m = 2$) and 27 edges are added to a face to form G_i. Note that each G_i has exactly $3(m+1)^2$ graph edges. Figure 2.5 shows how edges are formed from a single vertex v_j using the spanner interconnection scheme. The weight on a graph edge $\overline{v_a v_b}$ is the Euclidean distance between v_a and v_b times the weight of f_i, and the weighted cost is denoted as $\|\overline{v_a v_b}\|$.

A graph G is computed by forming the union of all face graphs $G_i, 1 \leq i \leq n$, making sure to eliminate the duplicate edges that are shared along the edges of $\mathcal{P}$ between subgraphs of two adjacent faces of $\mathcal{P}$. An approximate shortest path $\Pi'(s,t)$ in $\mathcal{P}$ is then computed as before by first determining a shortest path in G using Theorem 1.2. It can be shown that all edges of G lie on the surface of $\mathcal{P}$. Hence, any path in G (i.e., our approximation) can be transformed to a path on the surface of $\mathcal{P}$. In the sections to follow, we show that such a path exists in G and analyze its cost.

2.1.2.2 Bounding the Approximation

We begin the analysis by first considering the fixed or interval placement schemes using the complete interconnection strategy.

Claim 2.2 *Given a segment s_j crossing face f_i, there exists an edge s'_j in G_i such that $\|s'_j\| \leq \|s_j\| + w_{f_i} \cdot \frac{|L|}{m+1}$.*

Proof: Each edge in $\mathcal{P}$ is divided into at most $m+1$ intervals which have length at most $\frac{|L|}{m+1}$. Let $s_j = \overline{ab}$, where a and b are the end points of s_j lying on edges e_a and e_b of f_i, $e_a \neq e_b$, respectively. Let c (respectively, d) be a Steiner point in f_i, where c (respectively, d) is closest to a (respectively, b) among Steiner points on e_a (respectively, e_b). Since c and d lie on different edges of f_i, we know that there is an edge $s'_j \in G_i$ joining them (see Figure 2.6). The triangle inequality ensures that $|s'_j| \leq |\overline{ca}| + |s_j| + |\overline{bd}|$. Since we chose the closer interval endpoints, then $|\overline{ca}| \leq \frac{|L|}{2(m+1)}$ and $|\overline{bd}| \leq \frac{|L|}{2(m+1)}$. Hence

$$|s'_j| \leq |s_j| + \frac{|L|}{m+1}. \tag{2.1}$$

Now, multiplying by w_{f_i} we have

$$w_{f_i} \cdot |s'_j| \leq w_{f_i} \cdot |s_j| + w_{f_i} \cdot \frac{|L|}{m+1}. \tag{2.2}$$

Notice that the cost of graph edge s'_j equals the length of the segment s'_j on $\mathcal{P}$

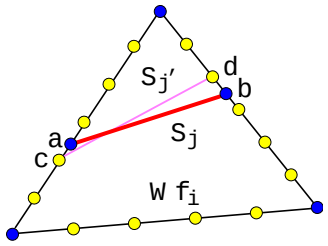

Figure 2.6: A face-crossing segment s_j of a weighted shortest path.

times the weight w_{f_i} of the face. Note also that we are using an upper bound that assigns a weight of w_{f_i} to each of $\overline{ca}$ and $\overline{bd}$. If the faces adjacent to f_i have a cheaper weight, then the weights on $\overline{ca}$ and $\overline{bd}$ are reduced and the bound on $\|s'_j\|$ is better than stated here. Moreover, the above arguments can be used to show that if s_j is edge using, then there exists a sequence of adjacent collinear Steiner edges joining the corresponding Steiner points, and we can view these collinear edges as a single segment s'_j.

□

Lemma 2.3 *Given two vertices s and t of G, there exists a path $\Pi'(s,t)$ in G such that $\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + \frac{|L|}{m+1} \cdot k \cdot W$, where k is the number of segments of $\Pi(s,t)$.*

Proof: Let $\Pi(s,t) = \{s_1, s_2, \cdots, s_k\}$. For each $s_j \in \Pi(s,t)$, it follows from Claim 2.2 that there exists an edge $s'_j \in G$ that approximates s_j. Let $\Pi'(s,t) = \{s'_1, s'_2, \cdots, s'_k\} \in G$. Observe that due to the construction, $\Pi'(s,t)$ is a connected path. By applying the results of Claim 2.2 to each segment of $\Pi'(s,t)$ we have:

$$\sum_{i=1}^{k} \|s'_i\| \leq \sum_{i=1}^{k} \left(\|s_i\| + w_{f_{s_i}} \cdot \frac{|L|}{m+1} \right)$$

where $w_{f_{s_i}}$ denotes the weight of the face through which s_i passes. Now rewrite this as:

$$\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + \frac{|L|}{m+1} \cdot \sum_{i=1}^{k} (w_{f_{s_i}}).$$

Since $w_{f_{s_i}} \leq W$ for all f_{s_i} by definition, then $\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + \frac{|L|}{m+1} \cdot k \cdot W$. □

Note that if we consider the edges $e_1, e_2, e_3, ..., e_k$ through which $\Pi(s,t)$ crosses (with respective weights $w_1, w_2, w_3, ..., w_k$), then we can rewrite the bound more precisely as

$$\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + \frac{1}{m+1} \sum_{i=1}^{k} w_i |e_i|$$

Theorem 2.2 *Using the complete interconnection strategy, we can compute an approximation $\Pi'(s,t)$ of a weighted shortest path $\Pi(s,t)$ between two vertices s and t of G such that $\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + W|L|$, where L is the longest edge of $\mathcal{P}$ and W is the maximum weight among all face weights of $\mathcal{P}$. Moreover, we can compute this path in $O(n^5)$ time.*

Proof: In Lemma 2.3 we have shown that there is a path $\Pi'(s,t)$ in G that approximates a shortest path $\Pi(s,t)$ on P. We use Theorem 1.2 to compute a shortest path between the vertices corresponding to s and t in G. This will result in a path in G that has either the same cost as $\Pi'(s,t)$ or even less. Since any path in G can be mapped to a path on P, we obtain an approximate path on P. From Property 1.15 it follows that $\Pi(s,t)$ may have $\Theta(n^2)$ segments. In Lemma 2.3, set $m = k - 1$ to obtain $\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + W|L|$.

Now we analyze the time complexity of the algorithm. Each edge of $\mathcal{P}$ contributes $O(n^2)$ graph vertices and each face contributes $O(n^4)$ graph edges, yielding a total of $O(n^5)$ edges for G. Theorem 1.2 is then applied which runs in $O(n^5)$ time. (The path

from G can be mapped to the path on $\mathcal{P}$ in the time proportional to the number of links in the path. Although in our implementation this additional step is avoided.) $\square$

The bound of the above theorem can be alternatively rewritten as $\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + \frac{W|L_T|}{|E|}$, where $|L_T|$ is the sum of all edge lengths of $\mathcal{P}$ and $|E|$ denotes the number of edges of $\mathcal{P}$. This bound uses the average edge length as opposed to the pessimistic longest edge length. In the worst case, however, this bound is the same as stated in the theorem. This bound can actually be made tighter and written as $\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + \frac{W|L_\Pi|}{|E_\Pi|}$ where $|L_\Pi|$ is the length of all edges that $\Pi(s,t)$ passes through and $|E_\Pi|$ is the number of edges that $\Pi(s,t)$ passes through. These bounds hold throughout the chapter.

Corollary 2.1 *Using the complete interconnection strategy, we can compute an approximation $\Pi'(s,t)$ of a weighted shortest path $\Pi(s,t)$ between two vertices s and t on a polyhedral surface $\mathcal{P}$ with n faces such that $\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + W|L|$, where L is the longest edge of $\mathcal{P}$ and W is the maximum weight among all face weights of $\mathcal{P}$. Moreover, we can compute this path in $O(n^5)$ time.*

In our analysis, we made the assumption that each edge crossed by a shortest path was of length $|L|$. In reality there may be many edges of $\mathcal{P}$ with small length compared to $|L|$. The interval placement scheme allows less Steiner points per edge while maintaining the same worst-case bounds. Figure 2.7 shows an example of how Steiner points are added to faces using a) the fixed placement scheme where $m = 7$ and b) the interval placement scheme. As can be seen in the figure, the interval scheme allows a significant decrease in the number of Steiner points placed while maintaining nearly the same path accuracy as with the fixed scheme.

We now provide an analysis for the case in which the spanner interconnection strategy is used instead of the complete interconnection strategy. This spanner scheme

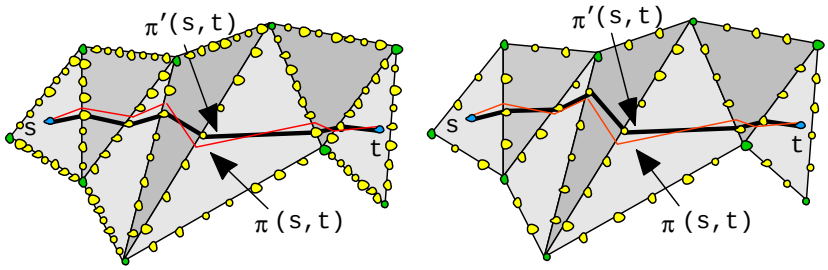

Figure 2.7: The difference in the layout of Steiner points (m=7) for a) the fixed placement scheme, b) the interval placement scheme.

improves upon the time complexity of the complete scheme; though the approximation achieved is not quite as good. The intuition behind this is that the spanner scheme graph is sparse while the complete scheme graph is much more dense; and so Theorem 1.2 will search the sparse graph in less time.

Smid [114] states that each G_i is a β-spanner, where $\beta = \frac{1}{cos\theta - sin\theta}$. The graph consists of $O(m)$ vertices and edges (recall that $|V_i| = O(m)$). Since we compute a spanner for each face of $\mathcal{P}$ individually and then merge each $G_i, 1 \leq i \leq n$ to form the union G, the resulting graph G has $O(mn)$ vertices and edges.

Theorem 2.3 *Using the spanner interconnection strategy, we can compute an approximation $\Pi'(s,t)$ of a weighted shortest path $\Pi(s,t)$ between two points s and t on a polyhedral surface $\mathcal{P}$ with n faces such that $\|\Pi'(s,t)\| \leq \beta(\|\Pi(s,t)\| + W|L|)$, where $\beta > 1$, L is the longest edge of $\mathcal{P}$ and W is the maximum weight among all face weights of $\mathcal{P}$. Moreover, we can compute this path in $O(n^3 \log n)$ time.*

Proof: In Claim 2.2 we can replace s'_j by an approximated path, say p_j, in G, where $|p_j| \leq \beta \cdot |s'_j|$, and use this value in Theorem 2.2, to obtain $\|\Pi'(s,t)\| \leq \beta(\|\Pi(s,t)\| + W|L|)$, where $\beta > 1$. Now we analyze the time complexity of the

algorithm. We apply the β-spanner scheme to obtain G_i for each face f_i of $\mathcal{P}$ where $1 \leq i \leq n$. Using $O(n^2)$ Steiner points per edge, each subgraph G_i contains $O(n^2)$ graph vertices and each vertex has a constant degree. Hence, G_i contains $O(n^2)$ graph edges. The graph G_i can be computed as follows. Observe that the vertices (or points) lie on the boundary of the triangular face, and they are at fixed intervals. For each vertex v_a all the cones ($O(1)$ in all) with apex at v_a can be computed in $O(1)$ time. Moreover the closest vertex to v_a in each cone can be computed in $O(1)$ time, by observing the relative location of vertices (or points) in the cone with respect to the perpendicular from v_a to the edges of f_i. This implies that each G_i can be computed in $O(n^2)$ time and G can be computed in $O(n^3)$ time. A shortest path in G can be computed by using Theorem 1.2 and it runs in $O(n^3 \log n)$ time.

□

2.1.2.3 Fine Tuning the Approximation - An Additional Sleeve Computation

Here we describe how an approximated Euclidean (unweighted) path can be fine-tuned to be of near-optimal length and sometimes be optimal. This technique is based on choosing an edge sequence in $\mathcal{P}$ according to a preliminary approximated path $\pi'(s,t)$ which is computed as mentioned earlier. We then determine a sleeve $\mathcal{S}$ by unfolding the faces along the edge sequence of $\pi'(s,t)$ and computing a shortest path $\pi_{\mathcal{S}}(s,t)$ that lies within $\mathcal{S}$. This path is then projected back onto $\mathcal{P}$ to obtain a refined approximation $\pi''(s,t)$. If the sleeve we choose happens to coincide with a shortest path edge sequence the final approximation is optimal.

In order to construct the sleeve, we choose a sequence of faces through which $\pi'(s,t)$ passes (i.e., for each segment s_i of $\pi'(s,t)$ we determine the face through which it passes). If s_i is not incident to a vertex of the face we append that face to

our list of faces for the sleeve. If s_i passes through a vertex, say v, we must make a decision as to whether or not the edge sequence should go around v in the clockwise (CW) or counter-clockwise (CCW) orientation. The example of Figure 2.8 shows how this decision can affect the overall accuracy of $\pi_{\mathcal{S}}(s,t)$. If we choose the shaded faces, $\pi_{\mathcal{S}}(s,t)$ will not be as good as if we choose a CCW path around v_3 since $\pi(s,t)$ does not pass through all of the shaded faces.

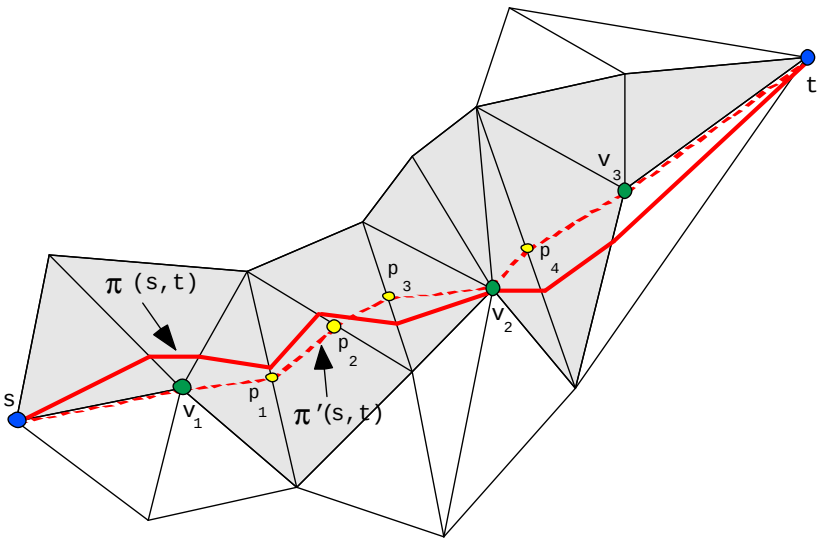

Figure 2.8: Choosing an edge sequence from the path $\pi'(s,t)$.

We attempt to choose the good edge sequence (hence sleeve) by applying a simple heuristic for the special class of polyhedra: TINs. Let s_i and s_{i+1} be consecutive edges of $\pi'(s,t)$ such that their shared endpoint lies at a vertex v of $\mathcal{P}$. Determine the turn type (i.e., left, right or collinear) between the projections of s_i and s_{i+1} onto the XY plane. If it is a left turn, we chose a CW path around the vertex, otherwise we chose a CCW path. In Figure 2.8 we can see that all three vertices that were crossed result in left turns and we have chosen the CW path around each. Obviously, at v_3 the heuristic has chosen badly and we will never obtain an optimal path from

$\pi'(s,t)$. However, we have observed that this heuristic performs well in practice on terrain data (see Section 2.2).

Given the unfolded sleeve, we compute a shortest path $\pi_{\mathcal{S}}(s,t)$ in the sleeve from s to t using the algorithm of Lee and Preparata [81]. One potential problem is that the algorithm applies to simple sleeves (non-intersecting). Since $\mathcal{P}$ is non-convex in general, $\mathcal{S}$ may be non-simple. We must show that even though the sleeve may be self-overlapping, it does not affect the algorithm correctness. We must also show that the resulting path is non-overlapping when projected back onto the surface of $\mathcal{P}$.

Property 2.2 *$\pi'(s,t)$ does not pass through a face of $\mathcal{P}$ more than once.*

Proof: The proof is by contradiction. Assume that $\pi'(s,t)$ is a shortest path in G that enters through an edge e of a face f_i at some Steiner point a and exits f_i at some Steiner point b. Assume that $\pi'(s,t)$ enters f_i again, say through Steiner point c. Since a and c are both Steiner points on edges of the same face f_i, there exists a Steiner edge $\overline{ac} \in G$ which is clearly a shortest path from a to c. Hence, the portion of $\pi'(s,t)$ from b to c cannot be a shortest path in G and we have a contradiction. □

Claim 2.3 *No two segments of $\pi_{\mathcal{S}}(s,t)$ lie in the same face of $\mathcal{P}$.*

Proof: Denote the consecutively computed funnels by $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_n$ for an n-face sleeve such that $\mathcal{F}_{i+1}$ is formed from $\mathcal{F}_i$ through the expansion of one face (i.e., extending the funnel by one sleeve diagonal). By definition, none of $\mathcal{F}_i, 1 \leq i \leq n$ are self-intersecting. However, it is possible that a funnel may intersect the tail which it is connected to. Hence, $\pi_{\mathcal{S}}(s,t)$ may be non-simple due to the constraints that force it to pass through adjacent faces of $\mathcal{S}$. By construction of any funnel $\mathcal{F}_i$, any convex chain of segments from $cusp(\mathcal{F}_i)$ to $lid(\mathcal{F}_i)$ can be formed by line segments which

pass through unique faces of $\mathcal{S}$. The left and right convex chains of $\mathcal{F}_i$ also consists of segments that lie in different faces of $\mathcal{S}$. Since the tail of any funnel is constructed by appending the convex chains of previous funnels, it is composed of segments that lie in different faces of $\mathcal{S}$. Therefore, $\pi_{\mathcal{S}}(s,t)$ contains segments that lie in different faces of $\mathcal{S}$ since it consists of pieces from $tail(\mathcal{F}_n)$, $chain(\mathcal{F}_n)$ and a path consisting of segments passing through the unique funnel faces.

□

Lemma 2.4 *The projected path $\pi''(s,t)$ is simple.*

Proof: The proof follows from Property 2.2, and Claim 2.3.

□

Section 2.2 shows that these approximated paths are more accurate with this additional computation at a negligible increase in execution time. In our algorithm, as m increases the difference $|\pi'(s,t)|-|\pi(s,t)|$ decreases (provided that the increase in m does not alter the previous m Steiner point locations). For some value of m, $\pi'(s,t)$ will pass through the same edge sequence as $\pi(s,t)$. Since the sleeve computation unfolds the sleeve which contains both $\pi'(s,t)$ and $\pi(s,t)$, path $\pi''(s,t)$ will exactly match $\pi(s,t)$. Hence, in some instances, the edge sequence of the approximated path is identical to that of $\pi(s,t)$ and the sleeve computation produces an exact shortest path.

There is no efficient algorithm for computing shortest paths in weighted sleeves. Hence, we apply a different approach, namely that of continuously refining approximations based on a selected region of the terrain. To do this, we first compute a preliminary approximation $\Pi'(s,t)$ as before. We then form $\mathcal{P}'$ as the union of all faces that $\Pi'(s,t)$ intersects. If $\Pi'(s,t)$ passes through a vertex v of $\mathcal{P}$, we include all faces incident to v. This union of faces forms a non-convex polyhedral surface $\mathcal{P}'$ but

in general it does not form a closed polyhedron. We call this union a *buffer* around $\Pi'(s,t)$. We then apply the approximation scheme on $\mathcal{P}'$ with an increased number of Steiner points per edge. As a result, we obtain a refined path. The refinement can be iterated allowing a trade-off between path accuracy and running time.

2.2 Experimental Results

In this section, we describe implementation issues, our testing procedures and experimental results. Due to their practical relevance, and in the context of our R&D [68], our experimental results are carried out on the subclass of non-convex polyhedra: TINs. In Geographic Information Systems, Cartography and related areas, shortest path problems arise on terrains which are often modeled using TINs as shown in Figure 2.9. The algorithms presented in the previous section apply to any non-convex polyhedron. In addition to the tests explained here, we have verified that our implementation also works on 3D model data (non-convex polyhedra) which we obtained from Gerhard Roth of the National Research Council of Canada. Figure 2.10 shows the results of applying the algorithm on two non-convex polyhedral models.

2.2.1 Implementation Issues

The implementation of our various algorithms involved implementing a variant of Dijkstra's algorithm, computing the unfolding of edge sequences (only for refinement stage), computing a shortest path in a planar sleeve (subset of the algorithm to compute shortest paths in a polygon) and finally a modification to store an implicit graph representation.

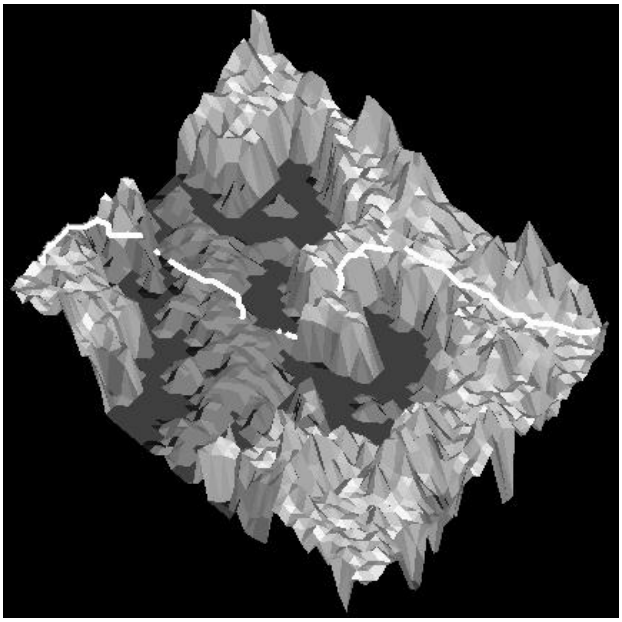

Figure 2.9: A weighted shortest path on a terrain in which traveling on water is expensive.

2.2.1.1 A Variation of Dijkstra's Algorithm

The variant of Dijkstra's algorithm used was that of the well-known A* algorithm (see [56]), which incorporates a "distance-to-goal estimate" during the search. An additional weight was associated with each vertex, namely its Euclidean straight line distance to the destination vertex. Then, during each iteration, we chose the vertex which minimized the sum of its cost from the source vertex plus the Euclidean distance to the destination vertex, over all candidate vertices. The use of this distance estimate allows the shortest path propagation to reach the destination point sooner. During implementation, we noticed a running time performance which was sometimes half that of the standard Dijkstra algorithm implementation; a substantial improvement.

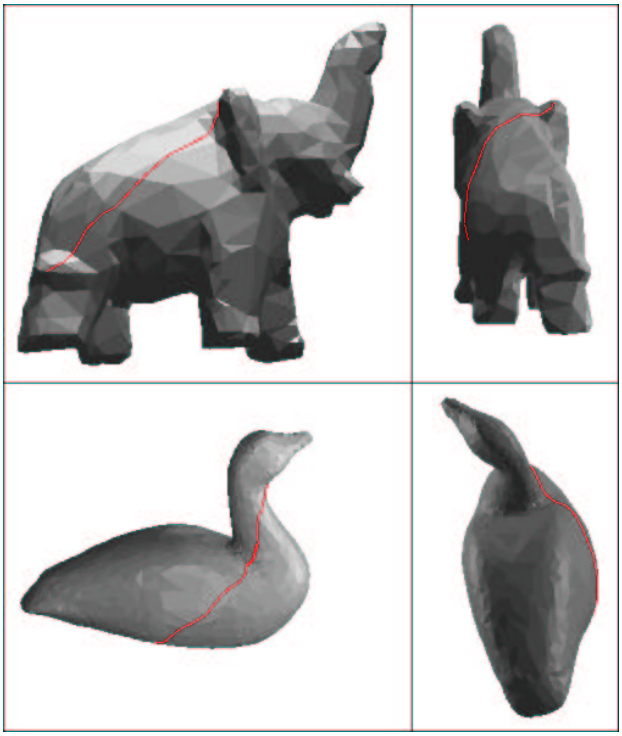

Figure 2.10: Shortest path approximations applied to non-convex polyhedral models.

2.2.1.2 Numerical Issues

Suri [115] points out that the Chen and Han algorithm [21] (which is based on unfolding) is sensitive to numerical problems, mainly due to the fact that 3D rotations are performed and errors accumulate along the paths and geometric structures computed. In our schemes the paths go through vertices or Steiner points (with the exception of the variation using the sleeve computation as its final step) thus reducing the chances of accumulating numerical errors. When doing the final step in which a sleeve in 3D is unfolded onto a plane, we compute an exact shortest path within the resulting sleeve. Although this is an exact path computation, it too is susceptible to numerical errors since the unfolding process may have generated discrepancies in the sleeve itself.

We have shown however, that this final stage of refinement does provide significant improvement in the resulting accuracy.

Our implementation of Chen and Han's algorithm was designed to enable time and approximation quality comparisons with our algorithms. We took care of numeric stability issues as required. When using LEDA's data types [87], the running time of our implementation of Chen and Han's algorithm deteriorated drastically. Thus, by using LEDA (or similar) the comparison to our algorithms would have become worse for Chen and Han.

2.2.1.3 Implicit Graph Storage

All of our timing results here are based on the computation of a shortest path in a pre-computed and explicitly stored graph (i.e., the time for computing the graph was not considered). Since our graphs are created based on a well defined geometric sequence of points along an edge, it is possible to implicitly store the graph vertices and edges and only compute them as needed. That is, the graph can be computed "on the fly" during the execution of Dijkstra's algorithm.

In addition to the explicit storage scheme, a partially implicit storage scheme was also implemented. In this partial storage scheme, the Steiner points are computed and each edge of $\mathcal{P}$ is assigned a list of the Steiner points (in order from start to end) that lie on it. Also, each edge of $\mathcal{P}$ keeps pointers to the edges that are clockwise and counter-clockwise from its endpoints (see Figure 2.11a). Each Steiner point keeps a pointer to the edge on which it lies as well as its index in the list for that edge. Vertices of $\mathcal{P}$ keep pointers to all edges of the faces incident to it (see Figure 2.11b).

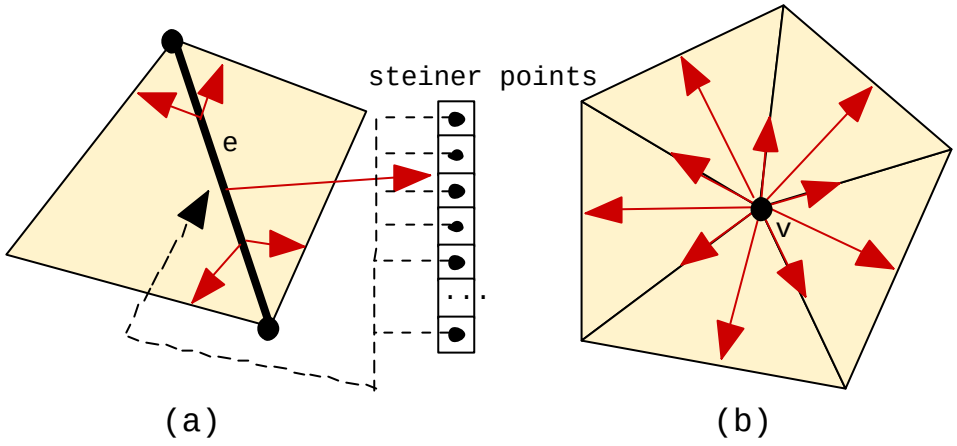

Figure 2.11: Pointers required in the partially implicit graph storage scheme. a) for each edge of $\mathcal{P}$, b) for vertices of $\mathcal{P}$.

In order to simplify the code, this improved implementation still added the vertices to the priority queue during the initial stage of Dijkstra's algorithm. It should be pointed out that a further improvement can be made by only adding the vertices to the priority queue as they are encountered during propagation. Essentially, a graph is still stored containing the same Steiner points as before, however, it contains only n edges (instead of $O(nm^2)$ as in the explicit storage scheme). The $O(m^2)$ edges per face are stored implicitly through the use of the constant number of pointers per edge. We ran tests which showed that this implicit edge representation improved the preprocessing time (i.e., the graph construction) by a factor of roughly between two and three. We also found that this implicit graph storage strategy resulted in a negligible (barely noticeable) increase in the run time of Dijkstra's algorithm. In addition, the implicit scheme allowed more Steiner points to be added before the use of virtual memory was required. At this point, the implicit graph had better query runtime performance.

2.2.2 Test Data and Testing Procedure

One of the main difficulties in presenting experimental results is the lack of data, in general, and here of *benchmark* TINs. It is conceivable that different TIN characteristics could affect the performance of an algorithm. We have attempted to accommodate different characteristics by performing our tests with TINs that have different sizes (i.e., number of faces), height characteristics (i.e., smooth or spiky as modeled by accentuating the heights), and data sources (i.e., random or sampled from Digital Elevation Models (DEM)) etc.. Table 2.1 shows the attributes of the TINs that we tested. Figure 2.12 depicts screen snapshots of the last 10 TINS of Table 2.1. TINs with stretched heights were created by multiplying their heights by five.

No. of FACES	STRETCHED	DATA SOURCE
1,012	NO	DEM
1,012	YES	DEM
5,000	NO	RANDOM
5,000	YES	RANDOM
10,082	NO	DEM
10,082	YES	DEM
9,799	NO	DEM of partial Africa
9,788	NO	DEM of partial North America
9,944	NO	DEM of partial Australia
9,778	NO	DEM of partial Brazil
9,817	NO	DEM of partial Europe
9,690	NO	DEM of partial Greenland
10,952	NO	DEM of partial Italy
2,854	NO	DEM of partial Japan
9,839	NO	DEM of partial Madagascar
9,781	NO	DEM of partial Northwest Territories

Table 2.1: The data used for the experiments showing the TINs and their attributes.

For the weighted domain, we used the same TINs and set the weight of each face to be the slope of the face. Thus, steeper faces have higher weight. Each edge of the

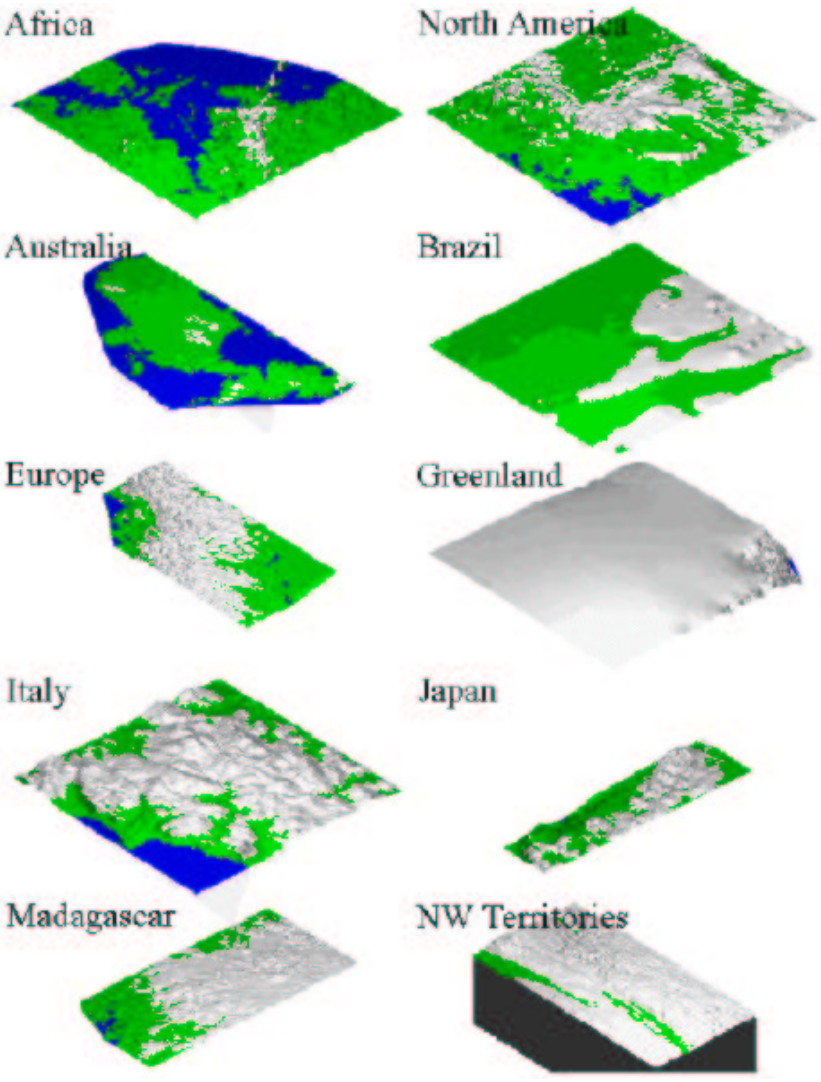

Figure 2.12: Snapshots showing various TINs that were used for testing.

TIN is given weight equal to the minimum of its adjacent faces. To determine if the results were biased due to our choice of weights, we ran additional tests in which the weights were chosen at random for each face.

For each TIN, we computed a set of 100 random vertex pairs. We then tested each of the approximation schemes listed in Table 2.2. We give the legend id of each scheme as they appear in the graphs (Throughout the remainder of this chapter, we use the terms "interval scheme" and "fixed scheme" to represent the interval and fixed Steiner point placement schemes, respectively. With the exception of the section on spanners, these terms refer to the complete interconnection strategy). For each test, we computed the path cost between each of the 100 vertex pairs and then obtained an "average path cost" for these pairs. We also computed the average computation time for the 100 pairs. The timing results presented here include the time required to compute the path itself, not just to produce the cost. The tests were performed in iterations based on the number of Steiner points per edge. Each scheme was tested for both weighted and unweighted scenarios (with the exception of the sleeve computations which were only computed in the unweighted case; a second approximation using a buffer was applied in the weighted case).

LEGEND ID	PLACEMENT	INTERCONNECTION	SLEEVE COMPUTATION
INT	INTERVAL	COMPLETE	NO
FIX	FIXED	COMPLETE	NO
INTSLV	INTERVAL	COMPLETE	YES
FIXSLV	FIXED	COMPLETE	YES
x degree	BOTH	SPANNER	NO

Table 2.2: The different approximation schemes.

2.2.3 Path Accuracy

We first examine the Euclidean shortest path problem. Figure 2.13 depicts the results of these tests for the first six terrains of Table 2.1. The approximated path length rapidly converges to the actual path length (as computed using Chen and Han's algorithm [21]). The graphs show that six Steiner points per edge suffice to obtain close-to-optimal approximations. The path accuracy observed is far better than the theoretical bound derived.

To understand this, recall that our theoretical worst-case analysis assumes that all edges intersected by the approximate path are long. In most applications this will be unlikely as short edges are common. (Long edges, if they are present, tend to be near the boundary and will therefore not be crossed by a shortest path.) We have examined edge-length histograms for all of our TIN data and show a typical histogram in Figure 2.14. As can be seen in this example, very few edges are near the longest edge length of 205.3, most are much shorter.

2.2.3.1 Sleeve Computation

The graphs of Figure 2.13 also illustrate that the additional sleeve computation helps to obtain even better approximations. We concluded that the best of our unweighted schemes is the interval scheme with the sleeve computation (IntSlv).

2.2.3.2 Effects of Stretching

We now discus the effects on the accuracy due to stretching of the TIN. As mentioned, for this test, the heights of the vertices were multiplied by a factor of five. Figure 2.13 allows a comparison of accuracy for unstretched versus stretched TINs. In both

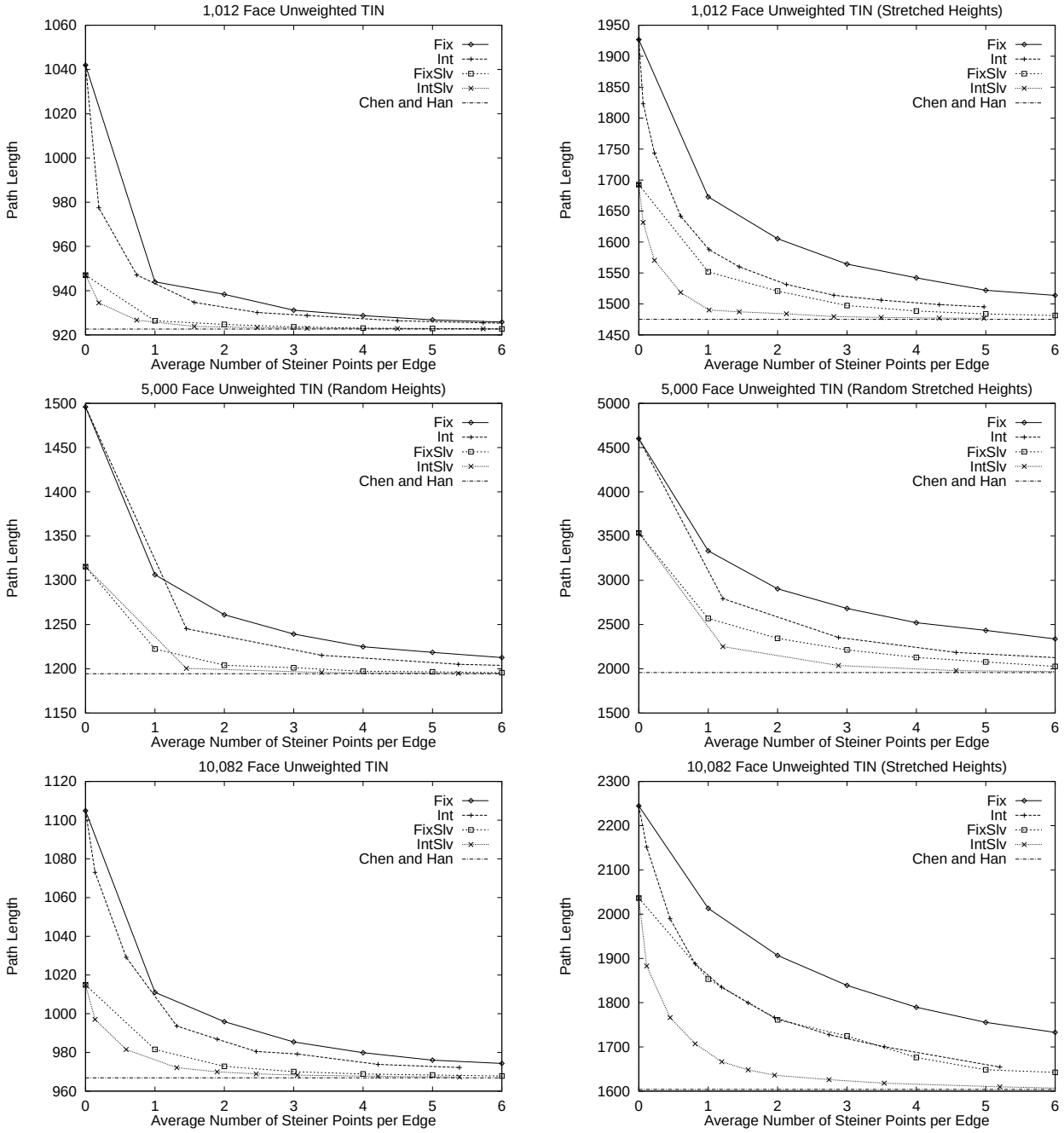

Figure 2.13: Graphs showing average path length for six selected terrains.

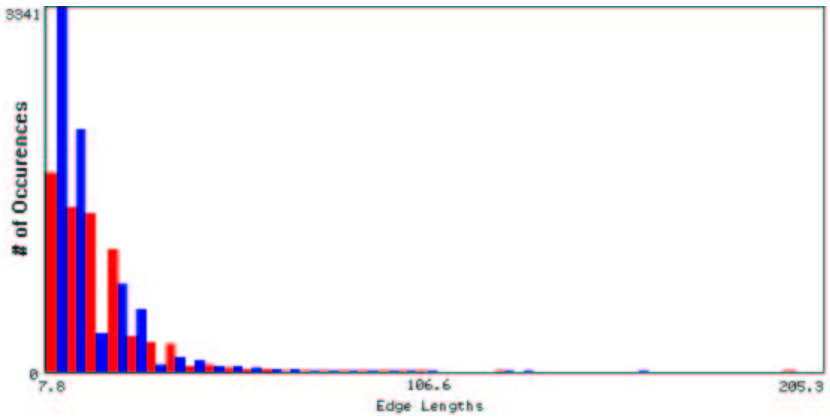

Figure 2.14: Histogram of edge lengths for one of our TINs.

instances, the approximate path length converges after only a few Steiner points per edge have been added. One may notice however, a slightly slower convergence for stretched input TINs. This is mainly due to the fact that Steiner points are placed further apart along the "now longer" edges. Therefore, it requires more Steiner points to reduce the interval size to that of the flatter TIN. The interval scheme performs better than the fixed scheme, since the interval scheme favours placement of Steiner points on longer edges and longer edges are more likely to be crossed by the set of paths.

Since the additional sleeve computation can produce an exact path in some cases, it is no surprise that it provides a significant improvement on the average path cost. Figure 2.15 shows the results of running sleeve match tests on a 1012 face terrain, a 1012 face stretched terrain and a 10082 face terrain. In each case, we determined the percentage of iterations that converged to the exact same edge sequence as a shortest path computed using our implementation of Chen and Han's algorithm. As can be easily seen, the smaller terrain had more sleeve matches and the stretched 1012 face terrain had less than the unstretched 1012 face terrain. In any case, we see that with

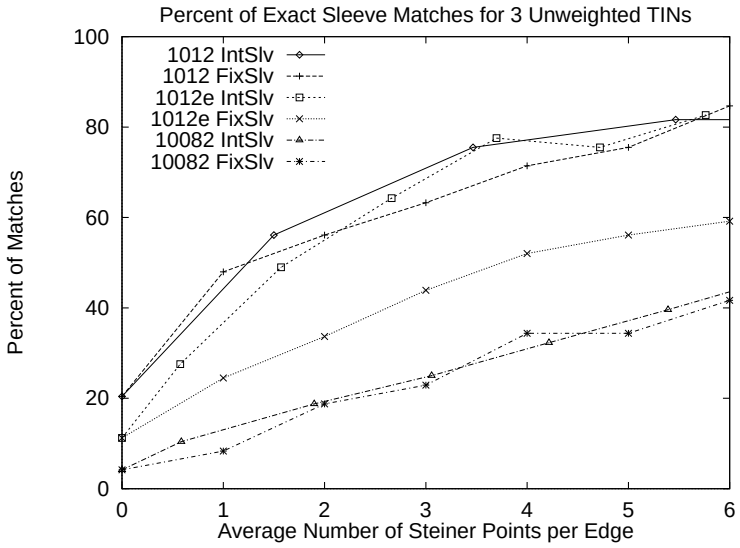

Figure 2.15: Graph showing the percentage of exact edge sequence matches for a 1012 face terrain, a 1012 face stretched terrain and a 10082 face terrain.

six Steiner points per edge, an exact shortest path is obtained between 40% to 80% of the time.

We have proven that in the worst case the Euclidean approximation for the fixed and interval schemes achieves an additive factor of $W|L|$. The analysis made the pessimistic assumption that each edge crossed by a shortest path was of length $|L|$ and that all faces had maximum weight. We made the claim that terrains typically have many edges which are shorter than L and hence our worst case analysis is an over estimate (see also Figure 2.14). Just after the proof of Theorem 2.2, we have shown that this bound can be written in terms of the average length of edges through which $\Pi(s,t)$ passes as follows: $\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + \frac{W|L_\Pi|}{|E_\Pi|}$. The graphs of Figure 2.16 compare this improved worst case theoretical accuracy with that of the produced accuracy for tests on the 10,082 face terrain using the fixed and interval schemes. The left graph depicts the maximum (i.e., worst case) error obtained from the 100 paths

tested; whereas the right graph depicts the average error obtained. The worst case and average case theoretical bounds using the computation of $\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + W|L|$ are not shown on the graph, but are 3.220407 and 1.445150, respectively. The results confirm that the algorithm performs much better in practice than predicted by the theory.

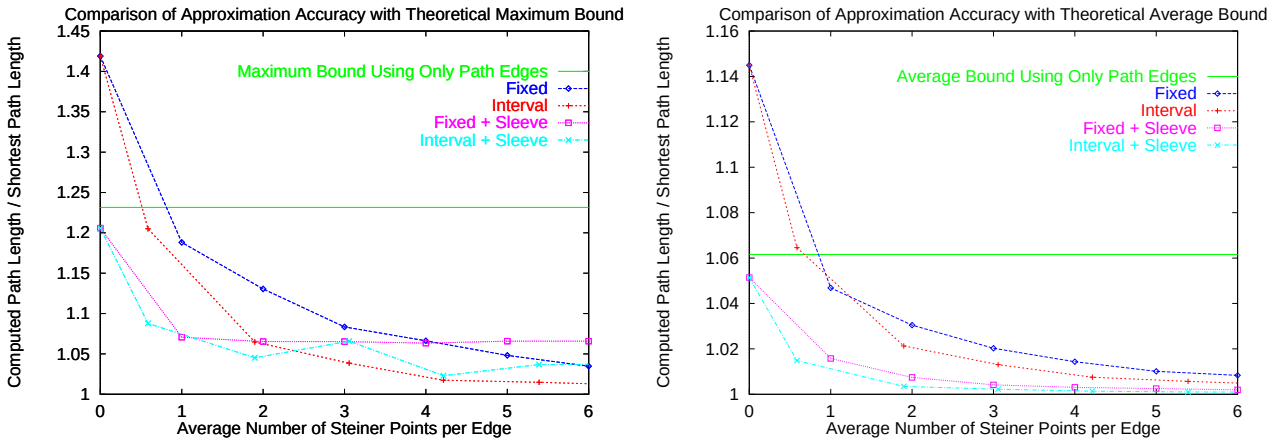

Figure 2.16: Graphs comparing the worst case (theoretical) accuracy with that of the produced accuracy for a 10,082 face terrain using the fixed and interval schemes (left: maximum error; right: average error).

2.2.3.3 Additional Terrains

In order to verify that the algorithm would perform well on a variety of terrain data, we ran additional tests on 10 terrains which were constructed from DEM data from various parts of the world as shown in Table 2.1. The results are depicted in Figure 2.17 and they verify that all terrains tested had similar accuracy and convergence behavior. The path converges quickly to near-optimal after only six Steiner points are added.

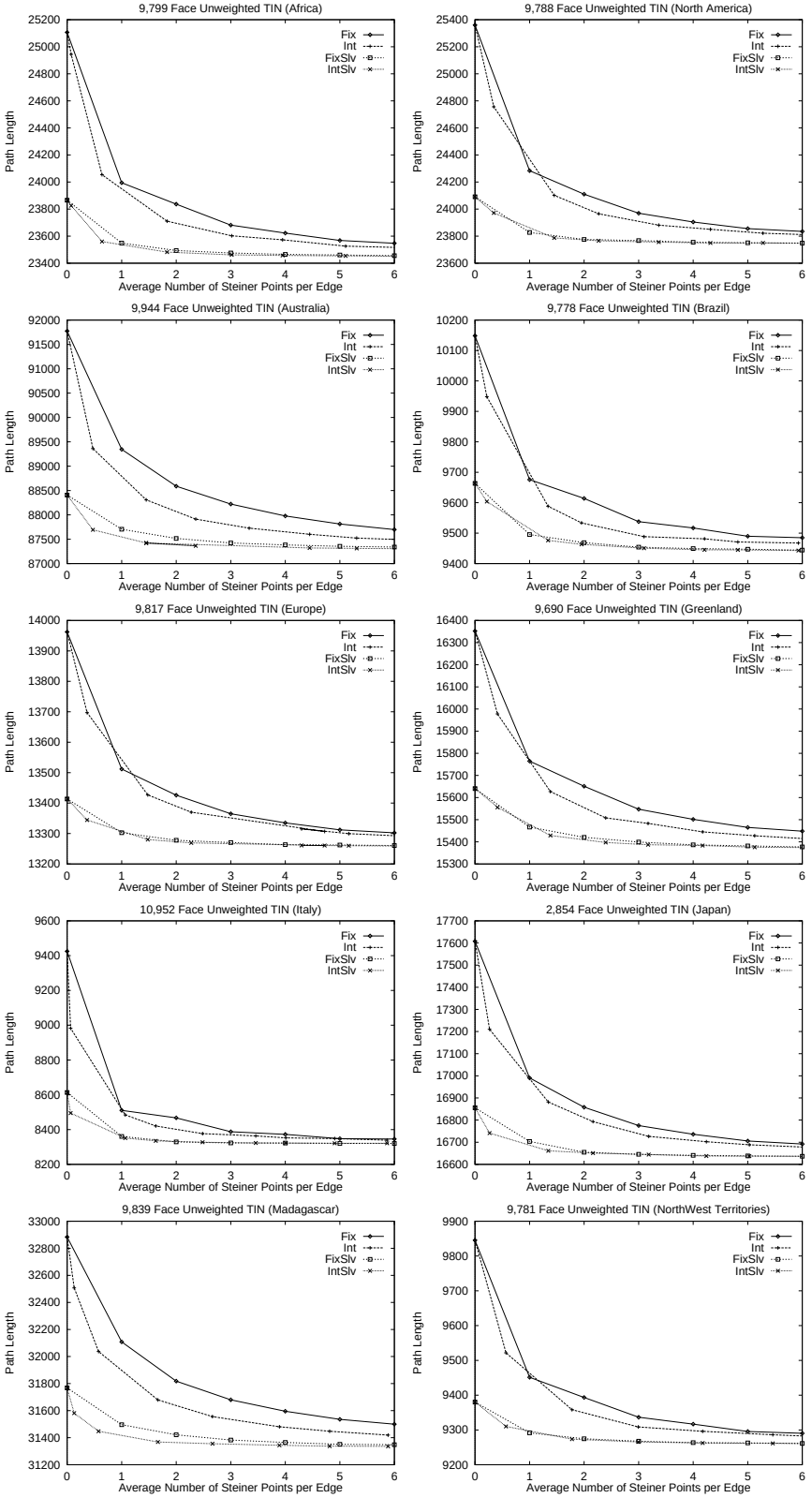

Figure 2.17: Graphs showing average path length for ten real-data terrains.

2.2.3.4 Spanners

To determine the effects of using graph spanners on the path accuracy we ran several experiments whose set-up is described next. In each test, we varied the degree of the graph spanner cones from 1° to 40° degrees in increments of five. The 1° test essentially represents the graph without spanners allowing therefore a comparison of accuracy for schemes with and without spanners. Figure 2.18 depicts the results of computing shortest Euclidean paths on the 10,082 face terrain using the fixed and interval schemes [1]. The graphs show the loss in path accuracy with increasing cone angle. As in the non-spanner schemes, the interval placement scheme converges quicker than the fixed placement scheme.

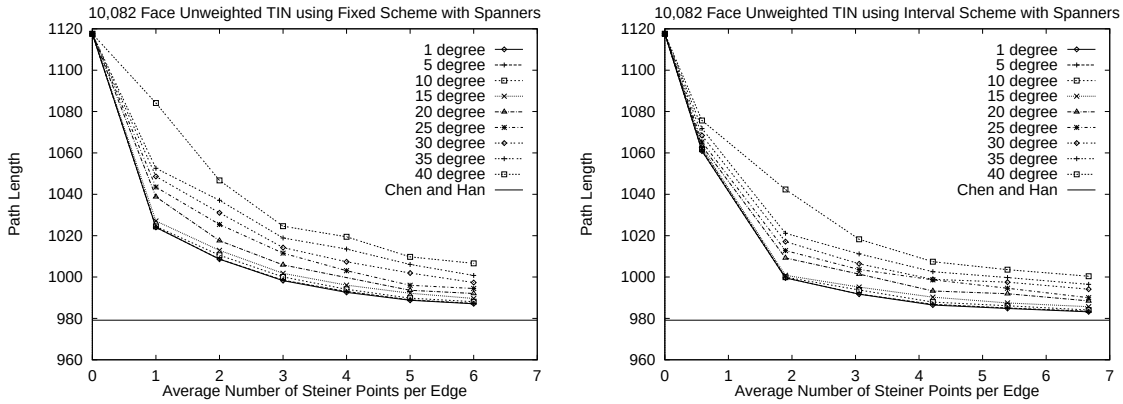

Figure 2.18: Graphs showing path accuracy obtained for a 10,082 face terrain using the fixed and interval schemes for a variety of spanner angles.

2.2.4 Computation Time

We attempted to make a fair comparison between our schemes and Chen and Han's algorithm (for the unweighted case). For this, we used the same geometric primitives

[1] The 100 (random) vertex-pairs considered here are different than those in Figure 2.13.

wherever possible.

In general, our algorithms' running times depend on the number of Steiner edges in G because we are invoking Theorem 1.2. Since we are adding only a constant number of Steiner points on average per edge, the total number of edges is linear, and thus the running time of our algorithms becomes $O(n \log n)$.

Figure 2.19 depicts the actual run-time results of the Euclidean shortest path tests for variations of our approximation schemes on our original six terrains. As we ran tests for many pairs of points, we precomputed the graph G instead of building it every time on the fly. Then we measured the time it took to compute an approximate path for a query pair (source, destination). We can see that our algorithms are substantially faster than that of Chen and Han[21]. The main reason is that our algorithms do not require any complex data structures. Also, with the exception of our sleeve computation, our algorithms do not perform expensive computations (such as 3D rotation and unfolding). Due to the scale of the graph (resulting from the large time difference to Chen and Han), we cannot distinguish between the characteristics of the fixed and interval schemes. Figure 2.20 depicts a graph showing the typical experimental run-time results obtained from our tests. Shown here are two sets of results, one for a 1,012 face TIN and one for the same TIN stretched by a factor of five. From the graph, we can see that the time required for the additional sleeve computation is negligible. In addition, there is very little difference between the fixed and interval schemes. This is mainly due to the fact that we are looking at the average number of Steiner points per edge. If we had chosen the X-axis of the graph to be the maximum number of Steiner points per edge then we would see that the fixed scheme had a much slower running time than the interval scheme. This however, would have been an unfair comparison. Note as well, that the computation time increases for the stretched terrain. The timing results for the 10 additional terrains is shown in Figure

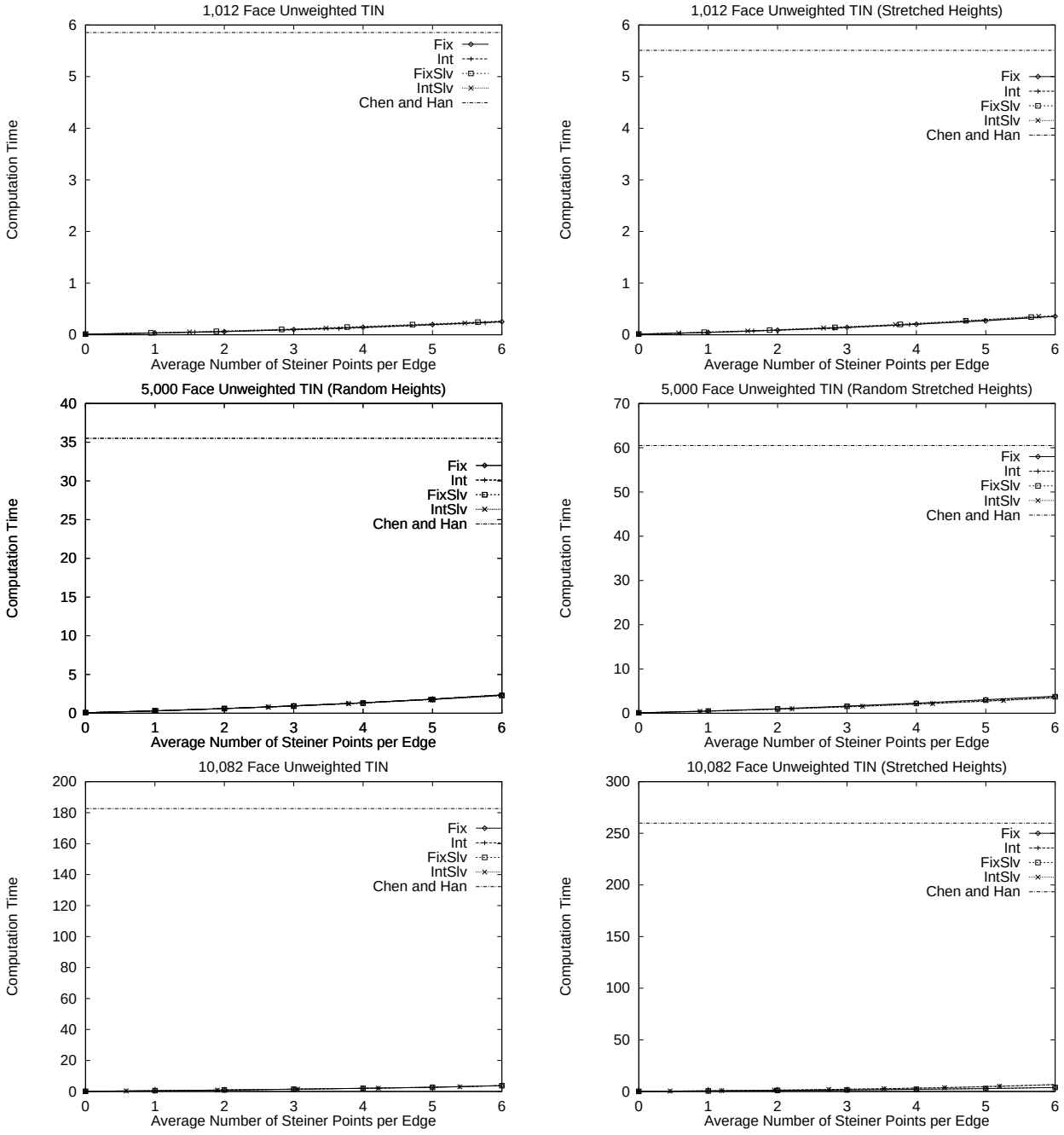

Figure 2.19: Graphs showing average computation time (in seconds) for six selected terrains.

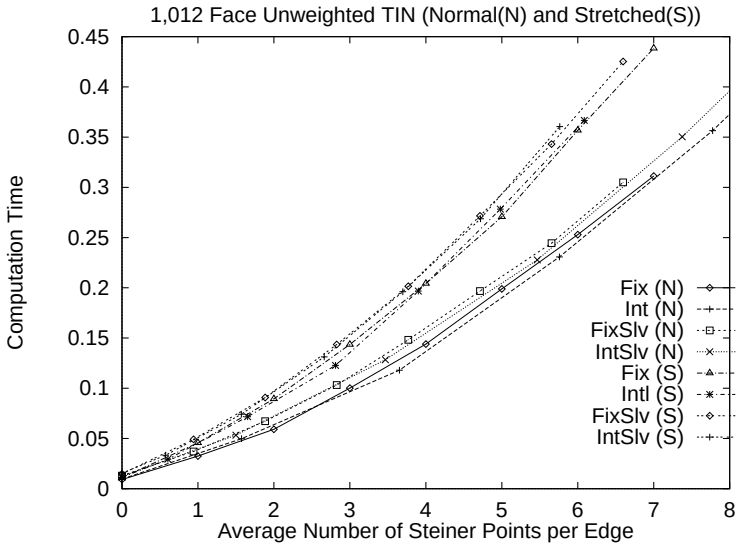

Figure 2.20: Graph showing the typical running time characteristics for the tested terrains using the fixed and interval schemes for normal(N) and stretched(S) terrains.

2.21; notice the similarly computed running time.

2.2.4.1 Graph Spanners

Getting back to the spanner schemes, we can now examine their computation time. Although the path accuracy is reduced when the sparse spanner graphs are used, the running time also decreases since there are less edges in the graph. Figure 2.22 shows the running time for the tests corresponding to the graphs of Figure 2.18. We can see that the spanners using the larger cone angle have better running time that those with smaller cone angles. In fact, we see that the graph shape goes from "quadratic" to "linear" since the number of graph edges becomes linear (times a constant) when spanners are used. We also see that there is little difference again between the fixed and interval schemes.

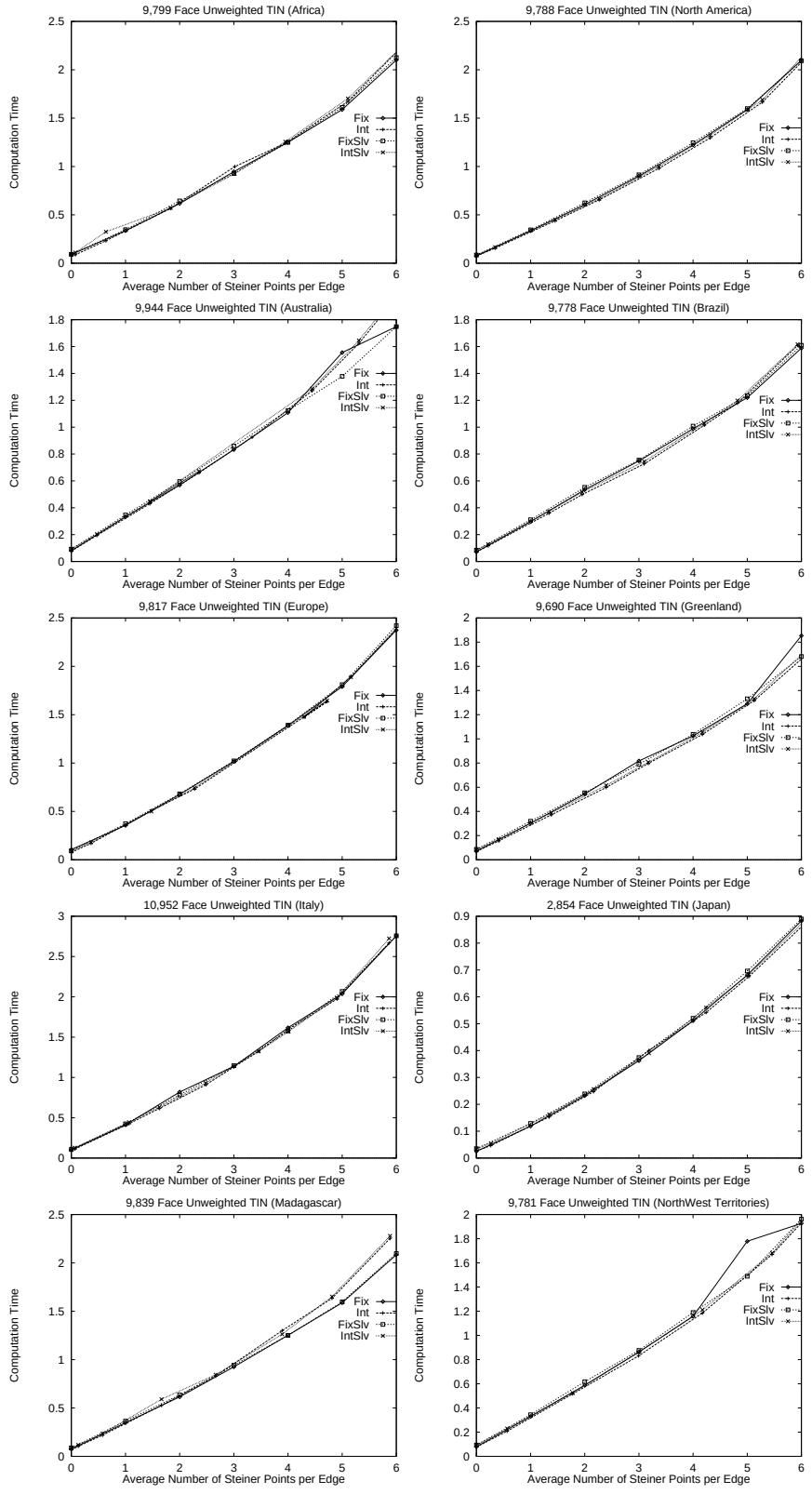

Figure 2.21: Graphs showing avg. computation time (secs.) for ten real-data terrains.

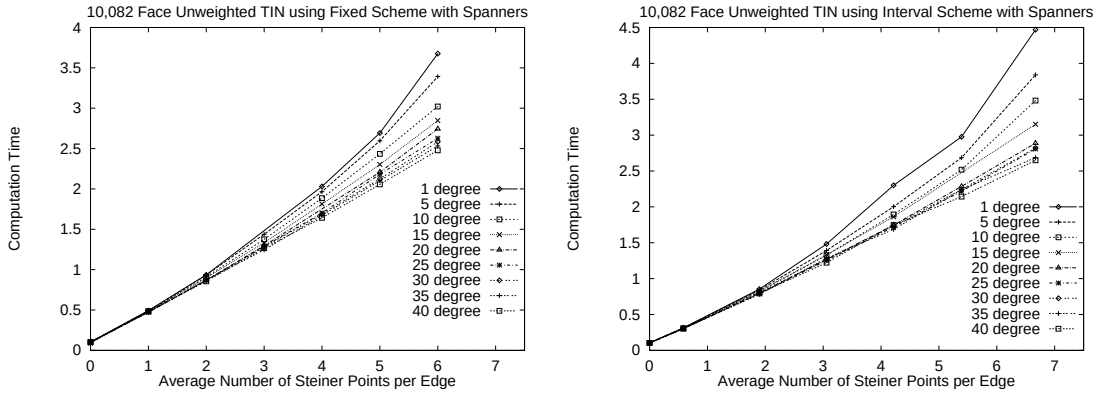

Figure 2.22: Graphs showing computation time obtained for a 10,082 face terrain using the fixed and interval schemes for a variety of spanner angles.

Clearly, the spanners allow a tradeoff between path accuracy and running time. From looking at these graphs, it is not immediately clear how the tradeoff can help make a decision as to whether or not to use a spanner and if so, what cone size to choose. The graphs of Figure 2.23 help determine the feasibility of the spanner scheme. The graphs show the path accuracy vs. computation time for the same data as Figure 2.18 and Figure 2.22. Here we can see which cone size provides the best path accuracy, when given a certain computation time. For instance, if a path is required in 2 seconds, one can see that the 1 degree spanner provides the best accuracy for that amount of time and the 40 degree spanner provides the worst accuracy.

Examining the graphs more closely, we notice that the 5 and 10 degree spanners provide essentially the same accuracy as the 1 degree spanner. The 15 degree spanner also has similar accuracy. Figure 2.22 illustrates that the 10 and 15 degree spanner results are approximately 20% to 30% faster than the 1 degree spanner when six Steiner points are used. One could therefore conclude that it may be worthwhile to implement the 5, 10 or 15 degree spanner since nearly the same accuracy can be obtained in less time. The graphs also indicate that if the allowable computation

time is small, the more sparse spanner schemes do not perform well and should not be used. If the graphs could be extrapolated for larger run times, it is likely that at some point, the spanner schemes will provide a better accuracy vs. run time tradeoff. However, since good path accuracy is obtained in practice after only a few (constant) Steiner points are added per edge, the more sparse spanners become impractical.

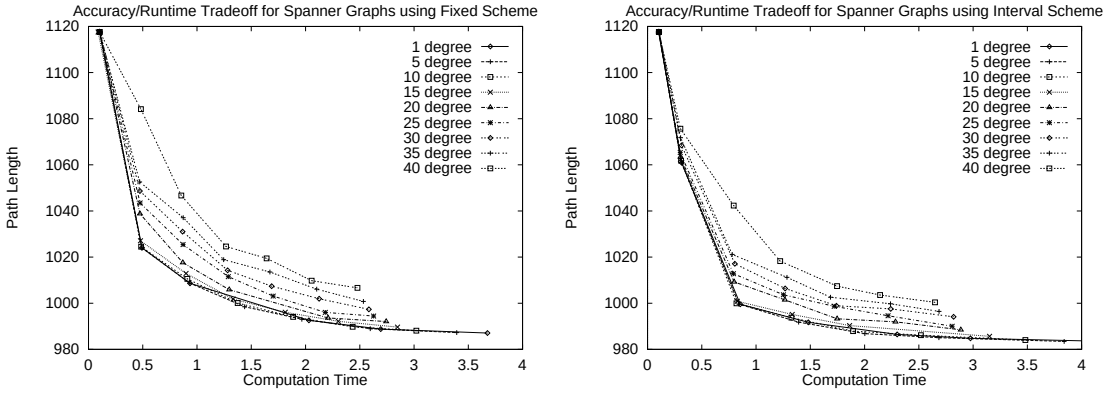

Figure 2.23: Graphs showing path accuracy vs. computation time obtained for a 10,082 face terrain using the fixed and interval schemes for a variety of spanner angles.

2.2.5 Weighted Paths

For the weighted scenario, there is no known algorithm that determines a (true) shortest weighted path. This poses a problem when determining the accuracy of approximation. Another problem arises when attempting to refine weighted paths by adding Steiner points. How many Steiner points are to be used during the second stage of approximation ?

For our initial experiments, we used the fixed scheme with 20 Steiner points per edge in this second stage. However, with the interval scheme, even though the average number of Steiner points per edge is small (e.g., six) there may be many more than

six Steiner points placed on longer edges. Therefore, the approximation accuracy may get worse during the second stage if we only allow a maximum of 20 Steiner points per edge.

Figure 2.24 shows the accuracy obtained through experimentation on weighted terrains with and without the second stage approximation (using 20 Steiner points per edge as mentioned). Like for the unweighted scenario, the path costs converge after only a few Steiner points have been added on each edge. Since the convergence is similar to that of the unweighted case, it is natural to conjecture that the cost of the paths converges to the actual weighted path cost. The second approximation based on the buffer technique provides an increase in accuracy, similarly. Since we use the same algorithm for unweighted and weighted scenarios, we obtained almost identical running time as shown in Figure 2.25. The second stage of approximation resulted in a significant increase in computation time. This is mainly due to the construction of a newly refined graph which is necessary for each query.

2.2.5.1 Time Independence from Weight Assignment

The results just mentioned are based on terrains in which weights were assigned to each face based on slope. To show that this assignment of weights does not bias the results, we ran additional tests in which the weights were chosen at random for each face. [2] The additional tests were performed on the normal and stretched versions of the 5000 face terrain with random heights and face weights. For these tests, we also changed the number of Steiner points used in the second stage of approximation. For the second stage approximation using the interval scheme, we increased the number of Steiner points per edge to produce intervals of approximately half of the size from the first stage. The fixed scheme tests were carried out as before with 20 Steiner

[2]The standard gnu-c function drand48() was used to generate the random weights.

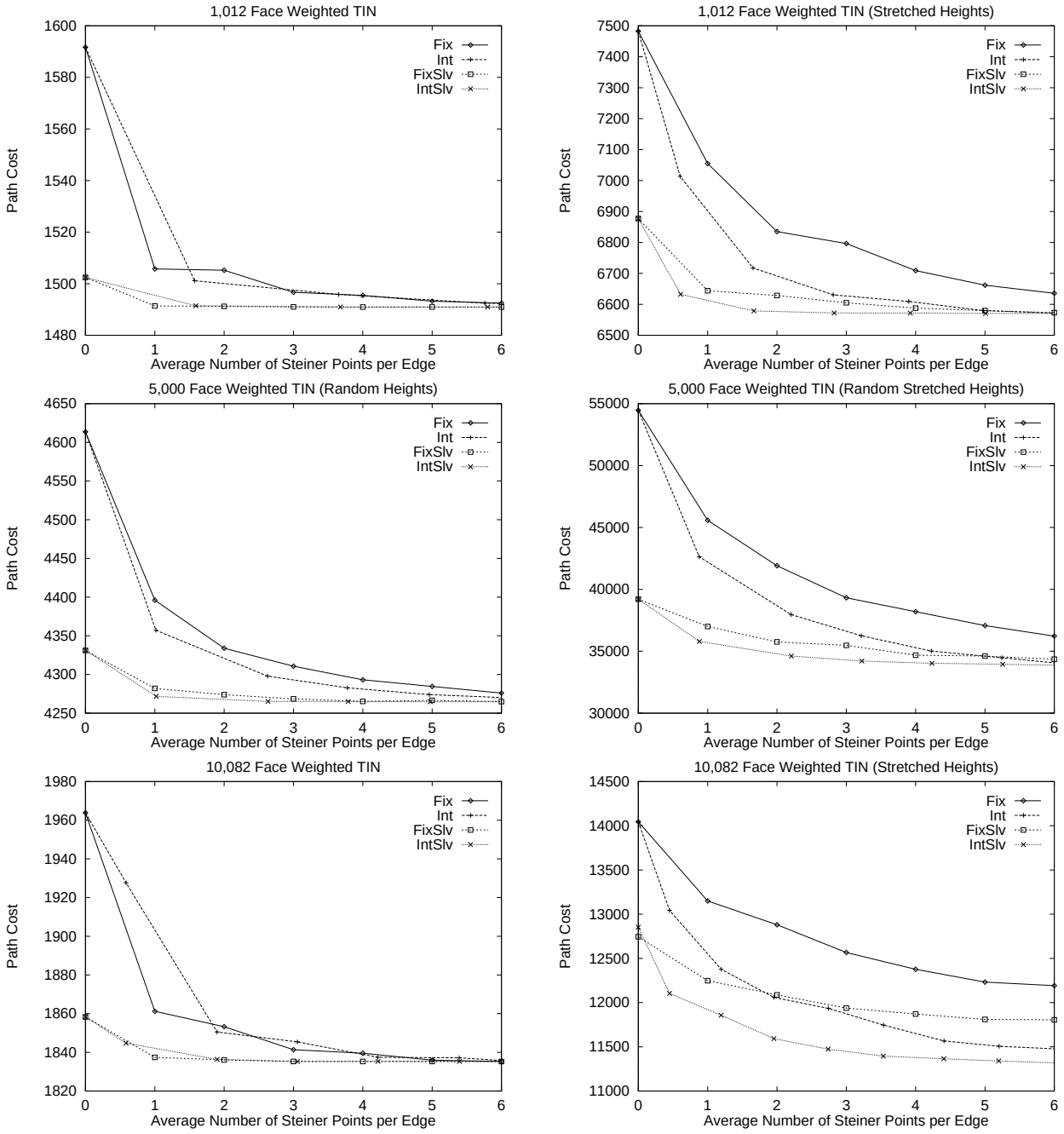

Figure 2.24: Graphs showing average path cost for six selected weighted terrains.

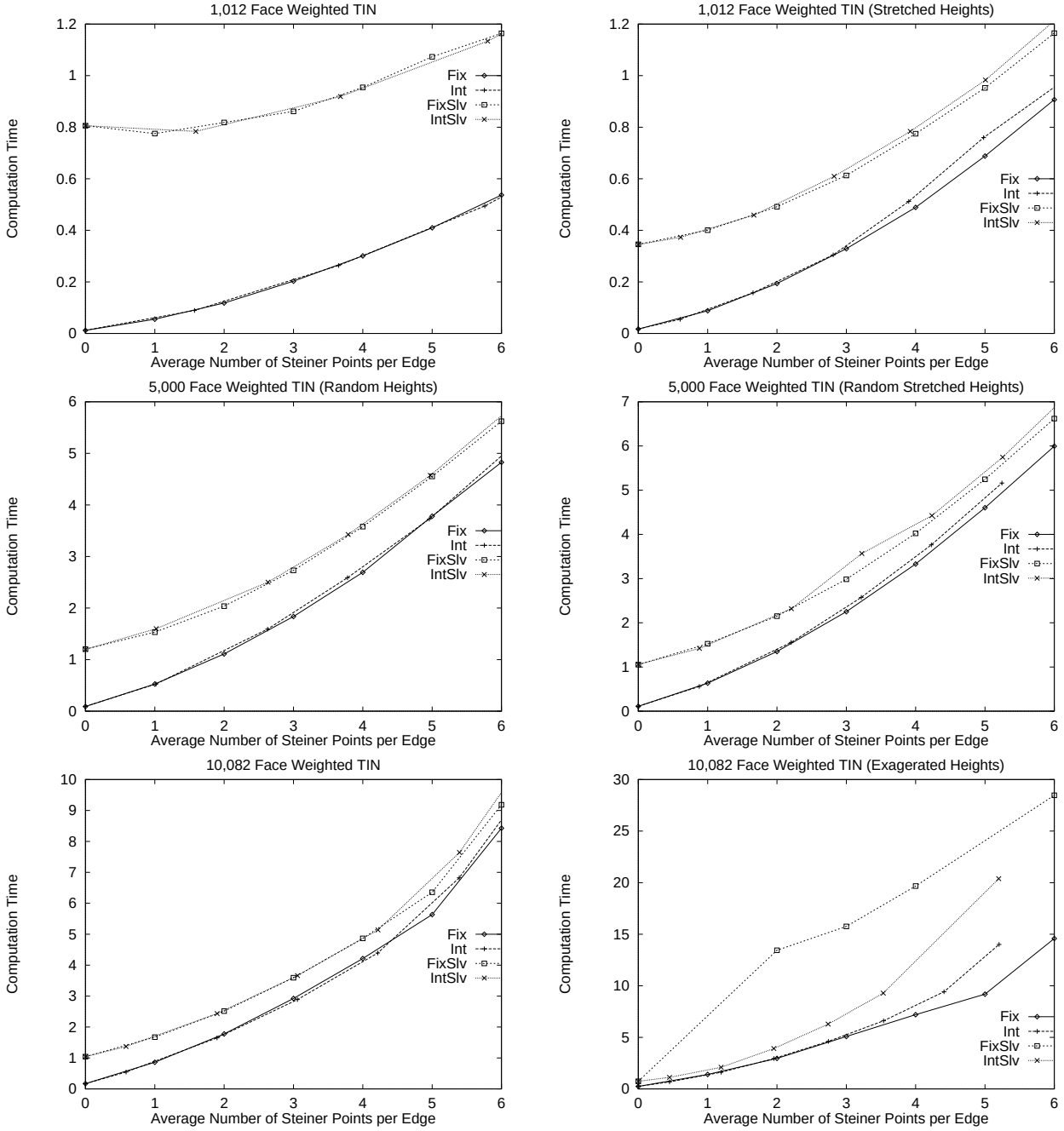

Figure 2.25: Graphs showing average computation time for six selected weighted terrains.

points per edge.

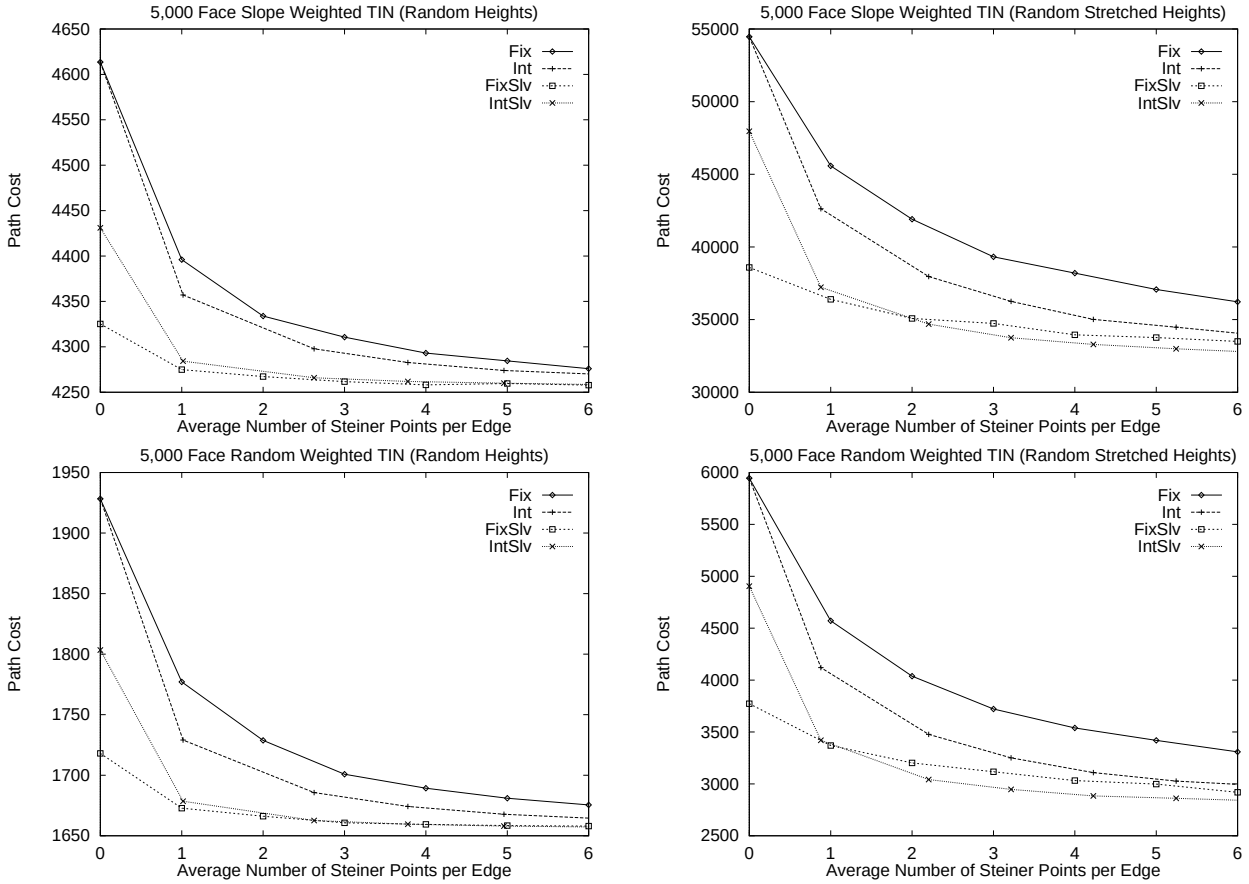

Figure 2.26: Graphs comparing the accuracy of weighted approximations for 2 weight scenarios: slope weights and random weights.

Figure 2.26 compares the weighted path costs between terrains with slope weights and random weights. The slope weight tests were re-done here with the new calculation for the second stage approximation. As can be seen by the graphs, there is very little difference between the shape and convergence behavior between the two weighted scenarios. The timing for these additional tests was similar to that of the original weighted path tests of Figure 2.25, and is therefore not shown here. The similar characteristic shape and scale of the graphs as compared to the unweighted

case imply that our algorithm is not sensitive to the weights of the faces.

2.3 Extensions

In this section, we describe two extensions to our algorithms. The first is that of computing paths between two arbitrary query points on the polyhedral surface. The second is that of producing ϵ-approximations under certain constraints.

2.3.1 Shortest Path Queries

In this section, we describe algorithms for computing paths between two arbitrary query points on the polyhedral surface. We consider two variations of the shortest path query problem: 1) single source, and 2) two-point queries. The preprocessing time involves running our algorithm from the previous section and building a structure for point location.

2.3.1.1 Single Source

Let us consider a fixed source (without loss of generality assumed to be a vertex) and allow arbitrary destination queries. Since the source is fixed, we can run Theorem 1.2 as a preprocessing step. If the destination query point is a vertex or a Steiner point, the cost from s to t is precomputed and thus can be reported in constant time. The path can be reported in time proportional to the number of its segments. If the destination t is not a vertex, the face f_t which contains t must first be located. Then the pre-determined shortest paths to the vertices and/or Steiner points of f_t are used. An approximation for $\Pi'(s,t)$ can be constructed in one of two ways through concatenation of:

1. $\{\Pi'(s, q_t), \overline{q_t t}\}$, where q_t is a Steiner point of f_t, or
2. $\{\Pi'(s, v_t), \overline{v_t t}\}$, where v_t is a vertex of f_t.

The choice of going through a vertex or a Steiner point allows for a tradeoff between path accuracy and query time. The following theorem applies to the two construction strategies above:

Theorem 2.4 *A polyhedron $\mathcal{P}$ can be preprocessed, for a given source vertex s such that a path $\Pi'(s,t)$ can be computed to a query point t on $\mathcal{P}$ in*
1) $O(m + \log n)$ time such that $\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + (1 + \frac{1}{m+1})W|L|$
2) $O(\log n)$ time such that $\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + (1 + \frac{2}{\sqrt{3}})W|L|$.

Proof: The proof is given separately for the two construction methods:
1) During preprocessing, we can precompute $O(m)$ paths from s to the Steiner points of f_t. The query time follows from Observation 1.1 applied to n faces with an additional $O(m)$ time for computing the minimum of the $O(m)$ precomputed paths. Consider the accuracy of the path so obtained. Let b be the point at which $\Pi(s,t)$ enters f_t and let b' be the Steiner point, closest to b, at which $\Pi'(s,t)$ enters f_t (as per our scheme). Using Theorem 2.2 we can derive the following bound on the cost of $\Pi'(s,t)$ to be $\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + (1 + \frac{1}{m+1})W|L|$.

2) For the analysis, we will assume that v_t is the vertex of f_t that is closest to t and let L be the longest edge among the edges of f_t. It can be shown that $|\overline{v_t t}| \leq \frac{|L|}{\sqrt{3}}$, since the distance $|v_t t|$ is maximized when f_t is an equilateral triangle and t is the point of intersection of the perpendicular bisectors of each of the sides. From Theorem 2.2 it follows that $\|\Pi'(s, v_t)\| \leq \|\Pi(s, v_t)\| + W|L|$ Applying this to the constructed path $\Pi'(s,t)$ we obtain: $\|\Pi'(s,t)\| = \|\Pi'(s,v_t)\| + \|\overline{v_t t}\| \leq \|\Pi(s,t)\| + \|\overline{v_t t}\| + W|L| + \|\overline{v_t t}\| \leq \|\Pi(s,t)\| + (1 + \frac{2}{\sqrt{3}})W|L|$. □

2.3.2 Two-Point Queries

Consider the case in which both s and t are arbitrary query points. Preprocess the polyhedron by computing a shortest path from each vertex of $\mathcal{P}$ to all Steiner points. We can then construct a path in one of two ways as for the fixed source query problem. Let f_s and f_t be the faces containing s and t, respectively. Let q_s (respectively q_t) be a Steiner point of f_s and let v_s (respectively v_t) be a vertex of f_s (respectively f_t). We can construct $\Pi'(s,t)$ in one of the following ways: (A) $\overline{sq_s}, \Pi'(q_s,q_t), \overline{q_t t}$ (B) $\overline{sv_s}, \Pi'(v_s,q_t), \overline{q_t t}$ (C) $\overline{sq_s}, \Pi'(q_s,v_t), \overline{v_t t}$ (D) $\overline{sv_s}, \Pi'(v_s,v_t), \overline{v_t t}$.

In order to make the best choice of q_s, q_t, v_s and/or v_t, we compute the path for all possible pairs of Steiner points and/or vertices of f_s and f_t. In the first case, this takes $O(m^2)$ time as there are $O(m)$ possibilities for both q_s and q_t. For the second (respectively third) case, we need to check $O(m)$ possibilities for q_t (respectively q_s) and three possibilities for v_s (respectively v_t). This takes $O(m)$ time. Since there are at most nine combinations to check, the last case can be carried out in constant time.

Theorem 2.5 *A given polyhedron $\mathcal{P}$ can be preprocessed such that a path $\Pi'(s,t)$ can be computed between two query points s and t on $\mathcal{P}$ in*
1) $O(\log n + m^2)$ time such that $\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + (1 + \frac{2}{m+1})W|L|$,
2) $O(\log n + m)$ time such that $\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + (1 + \frac{2}{\sqrt{3}} + \frac{1}{m+1})W|L|$,
3) $O(\log n)$ time such that $\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + (1 + \frac{4}{\sqrt{3}})W|L|$.

Proof: The proof is similar to Theorem 2.4 where 1) uses construction method (A), 2) uses construction method (B) or (C), and 3) uses construction method (D). □

2.3.3 ϵ-Approximations

All our algorithms can be expressed as ϵ-approximations algorithm if s and t are vertices of $\mathcal{P}$. In this case, we can bound the minimum possible length of $\Pi(s,t)$ since $\|\Pi(s,t)\| \geq |l|$, where l is the shortest edge in $\mathcal{P}$.

Theorem 2.6 *Using the complete interconnection strategy, we can compute an approximation $\Pi'(s,t)$ of a weighted shortest path $\Pi(s,t)$ between two vertices s and t of G such that $\|\Pi'(s,t)\| \leq (1+\epsilon)\|\Pi(s,t)\|$, where $0 < \epsilon < 1$. Moreover, we can compute this path in $O(n^3 K \log nK + n^5 K^2)$ time where $K = \frac{W|L|}{\epsilon|l|}$.*

Proof: Using Lemma 2.3, we can express our approximation accuracy in terms of the longest and shortest edge lengths of $\mathcal{P}$ as follows:

$$\begin{aligned}
\|\Pi'(s,t)\| &\leq \|\Pi(s,t)\| + \frac{kW|L|}{m+1} \\
&= \|\Pi(s,t)\| + \frac{kW|L||l|}{(m+1)|l|} \\
&\leq \|\Pi(s,t)\| + \frac{kW|L|}{(m+1)|l|}\|\Pi(s,t)\| \\
&= (1 + \frac{kW|L|}{(m+1)|l|})\|\Pi(s,t)\|
\end{aligned}$$

From this result we have shown that an ϵ-approximation is achievable where $\epsilon = \frac{kW|L|}{(m+1)|l|}$. In order to obtain this accuracy we must choose $m \geq \frac{kW|L|}{\epsilon|l|} - 1$ Steiner points per edge. (Note that by setting $m > \frac{kW|L|}{|l|} - 1$, then $\epsilon \leq 1$). Since $k = \Theta(n^2)$, we have $m = O(\frac{n^2 W|L|}{\epsilon|l|})$. The running time using Theorem 1.2 is $O(nm \log nm + nm^2)$. If we let $K = \frac{W|L|}{\epsilon|l|}$ then the running time is $O(n^3 K \log nK + n^5 K^2)$.

□

Corollary 2.2 *Using the complete interconnection strategy, we can compute an approximation $\pi'(s,t)$ of a Euclidean shortest path $\pi(s,t)$ between two vertices s and t*

of G such that $|\pi'(s,t)| \leq (1+\epsilon)|\pi(s,t)|$, where $0 < \epsilon < 1$. Moreover, we can compute this path in $O(n^2 K \log nK + n^3 K^2)$ time where $K = \frac{|L|}{\epsilon |l|}$.

Theorem 2.7 *Using the β-spanner interconnection strategy, we can compute an approximation $\Pi'(s,t)$ of a weighted shortest path $\Pi(s,t)$ between two vertices s and t of G such that $\|\Pi'(s,t)\| \leq (1+\epsilon)\|\Pi(s,t)\|$, where $0 < \epsilon < 1$. Moreover, we can compute this path in $O(n^3 K \log nK)$ time where $K = \frac{\beta W |L|}{(1+\epsilon-\beta)|l|}$.*

Proof: Using Lemma 2.3, we can express our approximation accuracy in terms of the longest and shortest edge lengths of $\mathcal{P}$ as follows:

$$\begin{aligned} \|\Pi'(s,t)\| &\leq \beta \left(\|\Pi(s,t)\| + \frac{kW|L|}{m+1} \right) \\ &\leq \beta \left(1 + \frac{kW|L|}{(m+1)|l|} \right) \|\Pi(s,t)\| \end{aligned}$$

From this result we have shown that an ϵ-approximation is achievable where $1 + \epsilon = \beta + \beta \frac{kW|L|}{(m+1)|l|}$. Hence, $\epsilon = \beta - 1 + \frac{k\beta W|L|}{(m+1)|l|}$. To obtain this accuracy, we need to place $m = \frac{k\beta W|L|}{(1+\epsilon-\beta)|l|} - 1$ Steiner points on each edge. Since $k = \Theta(n^2)$, we have $m = O(\frac{n^2 \beta W|L|}{(1+\epsilon-\beta)|l|})$. Letting $K = \frac{\beta W|L|}{(1+\epsilon-\beta)|l|}$ then the running time using Theorem 1.2 is $O(n^3 K \log nK)$.

□

Corollary 2.3 *Using the β-spanner interconnection strategy, we can compute an approximation $\pi'(s,t)$ of a Euclidean shortest path $\pi(s,t)$ between two vertices s and t of G such that $|\pi'(s,t)| \leq (1+\epsilon)|\pi(s,t)|$, where $0 < \epsilon < 1$. Moreover, we can compute this path in $O(n^2 K \log nK)$ time where $K = \frac{\beta |L|}{(\epsilon+1-\beta)|l|}$.*

We can also apply different constraints such as making the assumption that $\|\Pi(s,t)\| \geq \text{dist}(s,t)$. We can obtain similar bounds which would contain the term

$w \cdot \text{dist}(s,t)$ in place of $|l|$. The drawback of this approach is that it requires s and t to be known ahead of time. This would prevent efficient queries to be answered since the building of G would have to be done during pre-processing and at that point, we do not know $\text{dist}(s,t)$ and hence do not know how many Steiner points are required to achieve the ϵ-approximation.

The number of Steiner points required to obtain this ϵ-approximation bound, makes the algorithm impractical from a theoretical point of view. However, we have shown that the algorithm is of high practical value since near-optimal paths are obtained after only a few Steiner points are added per edge.

2.4 Other Approximation Schemes

All of the schemes presented here are based on the discretization of $\mathcal{P}$ by constructing a graph in which Steiner points have been added to the edges of $\mathcal{P}$ and then interconnected. One question that immediately arises is whether or not it would be advantageous to place Steiner points in the interior of the face as well. A natural method of placing Steiner points interior to a face is by decomposing the face recursively into subfaces. We call this method the *face decomposition scheme* and it is obtained by constructing a graph G_i for each face f_i as follows. Add Steiner points to the midpoint of each edge of f_i. The vertices of G_i are the vertices of f_i along with the Steiner points just added. An edge is added to G_i connecting each vertex to the Steiner points lying on their incident edges of f_i. Also, add edges connecting all three Steiner points. The result is a decomposition of f_i into four smaller triangles called subfaces. We repeat this decomposition recursively on the 4 subfaces until the desired maximal edge length is obtained. Let h denote the number of levels of recursive decomposition, where $h = 0$ implies no decomposition. Figure 2.27 depicts

such a construction where $h = 2$. This concludes our construction of G_i. A graph G is then computed as before by forming the union of all $G_i, 1 \leq i \leq n$.

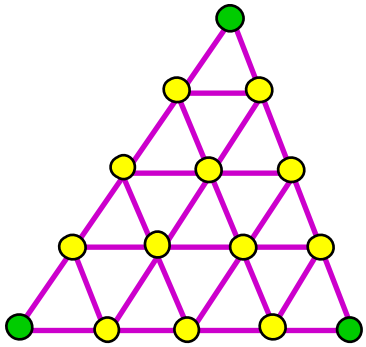

Figure 2.27: Recursively dividing a face into 4 subfaces by joining the midpoints of its edges.

At first glance, this approximation scheme appears to improve upon the accuracy of the approximation since we allow the path to cross faces of $\mathcal{P}$ instead of restricting it to travel only along edges. However, the portions of the approximated path that pass through faces have an undesirable "zig-zag" characteristic. Since this path does not satisfy Property 1.10, this characteristic will result in unnecessary errors in the approximation. In addition, this scheme requires adding Steiner points to each edge in powers of 2 which reduces its scalability.

It can be shown that this scheme provides no accuracy improvement whatsoever, regardless in the number of levels of decomposition when s and t are vertices of $\mathcal{P}$. This was confirmed experimentally. The addition of Steiner points internal to the faces did not provide an improvement in accuracy and in fact, produced poor approximations. This however, does not imply that the addition of any Steiner points to the interior of each face will result in a poor approximation. It would be nice to determine some "rules of thumb" that would give some indication as to which schemes

would perform better than others.

An even more interesting problem is to determine the optimal placement of these Steiner points and edges such that each G_i is planar. It is not difficult to see that no planar scheme can improve upon the accuracy of the complete interconnection scheme if both schemes have the same number of Steiner points along the edges. Hence given a fixed number of Steiner points on the edges of each face, adding Steiner points inside the face can never improve the overall accuracy of the approximation. However, it is possible that by adding points inside the face, one can produce a graph on the outer Steiner points with fewer edges and better characteristics (such as planarity). The accuracy obtained from such a graph may be close to that of the fixed scheme and the running time would be improved upon. In addition, by using planar graphs on each face, this may allow the application of more efficient search algorithms. Chew [24] for example, showed that there are some planar graphs which nearly approximate the complete graph. The planar graphs with these properties are variations of Delauney triangulations. For these reasons, we ran some experiments on two planar schemes as shown in Figure 2.28.

The *star* approximation scheme was created as an attempt to reduce the number of edges added to each face from $O(m^2)$ to $O(m)$ while ensuring planarity. Here we merely placed a Steiner point in the center of each triangle and connect it to the outer edge Steiner points. In essence, the scheme "forces" the approximated path to either pass through a vertex or center of a face. The *leaf* approximation scheme was created as an attempt to improve the star scheme and still allow only $O(m)$ edges per face. Once again, the graph produced on each face is planar. From the results of the fixed scheme (with complete interconnection), we knew that the biggest improvement on accuracy was the addition of the first Steiner point on each edge of the face. The leaf scheme attempts to keep the connections of the complete scheme (with exactly

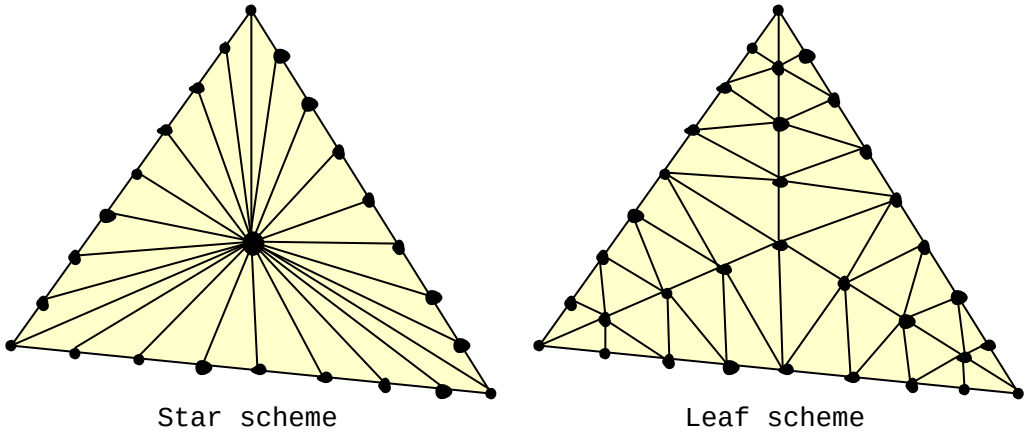

Figure 2.28: The star and leaf approximation schemes.

one Steiner point per edge), while adding internal vertices to make the graph planar. Then, approximately $\frac{m}{2}$ additional Steiner points are added to 3 of the internal edges and these points are connected to the outer Steiner points so that the resulting graph is a triangulation (although a triangulation is not required by the algorithm).

The graphs of Figure 2.29 compare the accuracy of the paths obtained by the star and leaf schemes with the accuracy of the fixed and interval schemes (with complete interconnection) for our 10,082 face TIN.

The graphs confirm that the star and leaf schemes produce worse accuracy than the fixed and interval schemes with and without the additional sleeve computation. The erratic nature of the star scheme indicates that the scheme is susceptible to small changes in Steiner point placement along the edges of the face. More importantly, the star scheme does not improve much as more Steiner points are added per edge. The second graph confirms that the sleeve computation provides a substantial improvement in the accuracy, but still, the resulting path accuracy never reaches that of the fixed and interval schemes. This implies that the schemes were unable to converge to

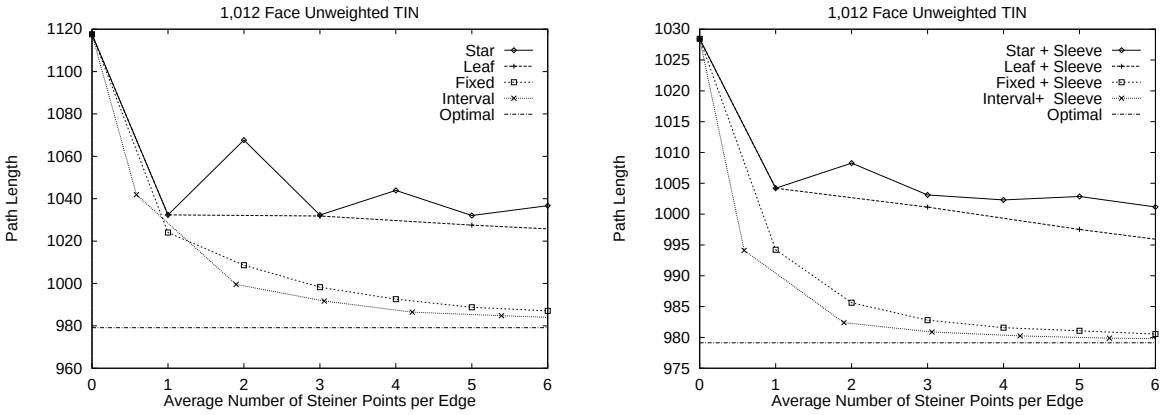

Figure 2.29: Graphs comparing the accuracy of the star and leaf schemes with that of the fixed and interval schemes. The two graphs depict the results with and without the additional sleeve computation.

a proper shortest path edge sequence. However, it is observed that the leaf scheme is less sensitive to Steiner point placement and does converge slightly better than the star scheme.

Getting back to the reason for creating these schemes, we see the running time results for the tests in Figure 2.30. Here we see the linear behaviour of these new schemes as compared to the quadratic behaviour of the fixed and interval schemes. Clearly, the star scheme has the best running time. Since we have only added a few Steiner points per edge, the leaf scheme is shown to have a higher running time than the fixed and interval schemes. If enough points were added, then one would see that the fixed/interval scheme running time would surpass that of the leaf scheme.

Although we have only experimented with two additional schemes, further investigation into other schemes remains as future work. For example, one may like to try schemes such as those depicted in Figure 2.31. The scheme of Figure 2.31a) is similar to the star scheme except that the "ears" of the face are sectioned off and triangulated separately. This may prove more useful than the star scheme since there

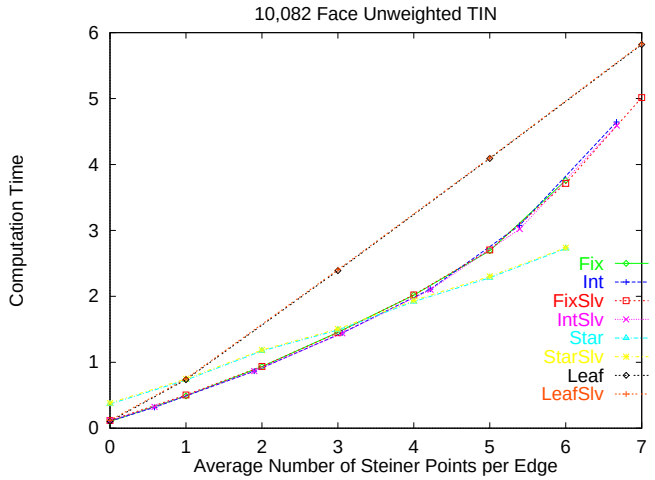

Figure 2.30: Graph comparing the running time of the star and leaf schemes with that of the fixed and interval schemes.

are less long edges close together. Another possibility is that of perhaps merging the star and face decomposition schemes as shown in Figure 2.31b). This would provide additional edge orientations for the face decomposition scheme (as opposed to its original 6) which would decrease the amount of zig-zagging and actually improve the accuracy of the face decomposition scheme. Figure 2.31c) would provide another method in increasing the orientations of the face decomposition scheme. It represents a 12-neighbourhood graph where each Steiner point is connected to its (up to) 12 closest neighbours. The resulting graph allows 12 orientations from each node and hence should improve upon the face decomposition scheme. However, the graph is no longer planar. Finally, Figure 2.31d) represents a layered scheme where a layer of Steiner points is placed inside the face forming a similar but smaller triangle. We add half the number of border Steiner points inside the face to form an inner triangle and connect them using the fixed scheme. The outer and inner triangles are then connected using some kind of triangulation strategy. Although this is described as a two layer approach, the layering can be iterated to allow many layers with a final

small triangle (perhaps even a point) in the center of the layers. This particular scheme spawned from some ideas we had regarding the placement of Steiner points based on using the "spring" graph drawing algorithm. The idea was to determine a "good" placement of Steiner points interior to the face. Intuitively, we felt that by applying the spring algorithm in which the Steiner points along the outer edges of the face were fixed and the Steiner points on the inner layer were allowed to move, then we would get an indication as to where the Steiner points should be placed. After applying the spring algorithm on several triangles, we found that the Steiner points near the middle of the inner layer edges seemed to cave inwards slightly. We felt that the effects of the spring algorithm did not provide a significant change in the placement of the inner Steiner points to warrant their usefulness.

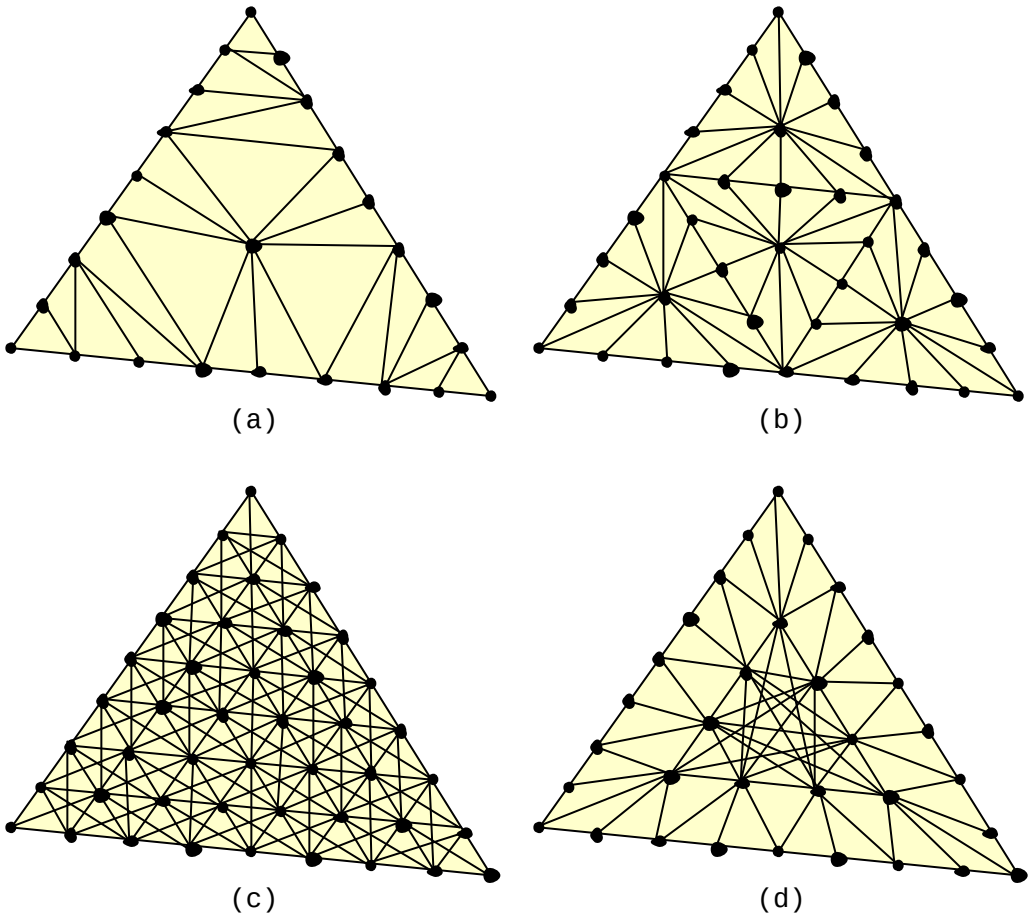

Figure 2.31: a) an ear-clipped star scheme, b) a combined face decomposition and star scheme, c) a 12 neighbourhood scheme and d) a layered scheme.

Chapter 3

An ϵ - Approximation Algorithm

It is often desirable to compute approximations of shortest paths that have a cost which is arbitrarily close to the exact shortest path cost. Our work described in Chapter 2 shows that very accurate paths can be obtained in practice, although the theoretical accuracy bound was not very attractive. That is, to obtain high accuracy in theory, a high number of Steiner points needs to be placed on each edge of the polyhedron. We also showed that an ϵ-approximation can be obtained for some special cases in which constraints are placed on the problem (such as allowing only queries between vertices or computing paths that have length at least dist(s,t)).

In this chapter, we focus on the problem of computing ϵ-approximations on weighted polyhedral surfaces. In addition, the algorithm presented in this chapter improves upon previous research by reducing the dependency on n in the running time, since in many applications n is quite large (larger than 10^5). This smaller dependency on n improves upon the work of Papadimitriou [100], Choi et al. [26], Clarkson [29], Mitchell and Papadimitriou [90] as well as Mata and Mitchell [86]. Table 3.1 compares the running times for the ϵ-approximation algorithms developed by [100], [26],

[29], [90], [86] and the one presented here. The comparisons are done as functions of n (the number of triangular faces of the polyhedron $\mathcal{P}$) and ϵ. From the table we can clearly see that our algorithm improves substantially the dependence on n. Although the objective of Chapter 2 was different, the schemes can also be formulated as ϵ-approximations (provided that additional constraints are met) in which the dependence on n becomes comparable to [86].

Reference	Running Time	Metric
Papadimitriou [100], Choi et al. [26]	$O(\frac{n^4}{\epsilon}\log^2\frac{n}{\epsilon}+\frac{n^4}{\epsilon^3}\log n)$	Euclidean
Clarkson [29]	$O(\frac{n^2}{\epsilon^4}\log\frac{n}{\epsilon}+n^2\log^2 n)$	Euclidean
Mitchell and Papadimitriou [90]	$O(n^8\log\frac{1}{\epsilon})$	weighted
Mata and Mitchell [86]	$O(\frac{n^3}{\epsilon})$	weighted
Results here	$O(\frac{n}{\epsilon}\log\frac{1}{\epsilon}\left(\frac{1}{\epsilon}\log\frac{1}{\epsilon}+\log n\right))$	weighted

Table 3.1: Comparison of Euclidean and weighted shortest path algorithms in terms of n and ϵ.

The chapter is organized as follows: Section 3.1 describes our approach as it differs from the previous work pertaining to ϵ-approximation algorithms and explains our results. Section 3.2 then presents the algorithm for the case in which both the source s and destination t are vertices of $\mathcal{P}$. The algorithm is described and its runtime bounds are analyzed. Section 3.3 then describes how the bounds change for the case in which s and t are arbitrary points.

3.1 Overview of Our Approach

Our approach to solving the problem is to again discretize the polyhedron in a natural way, by placing Steiner points along the edges of the polyhedron as in our edge subdivision approach of Chapter 2. In this work, a graph G is also constructed

containing the Steiner points as vertices and edges as those interconnections between Steiner points that correspond to segments which lie completely in the triangular faces of the polyhedron. The graph is then searched to obtain a shortest path. In the case where s and t are vertices of G, the ϵ-approximation path that we report will be a path in G. If s and t are not vertices of G, then the reported path will consist of a segment from s to some vertex $v \in G$, then a path in G from v to some other vertex $v' \in G$, and finally a segment from v' to t.

One of the differences to the work in Chapter 2 and to other somewhat related work (e.g., [33, 72]) lies in the placement of Steiner points. Here, we introduce a logarithmic number of Steiner points along each edge of $\mathcal{P}$, and these points are placed in a geometric progression along an edge. They are chosen with respect to the vertex joining two edges of a face such that the distance between any two adjacent points on an edge is at most ϵ times the shortest possible path segment that can cross that face between those two points. (This also differs from the approaches of Papadimitriou [100] as well as Choi et al. [27] which use distances from the starting point to place vertices thereby preventing their schemes from being extendible to queries with unknown starting point).

A problem arises when placing these Steiner points near vertices of the face since the shortest possible segment becomes infinitesimal in length as the segment gets closer to the vertex. A similar issue was encountered by Kenyon and Kenyon [72] and Das and Narasimhan [33] during their work on rectifiable curves on the plane and in 3-space, respectively. The problem arises since the distance between adjacent Steiner points, in the near vicinity of a vertex, would have to be infinitesimal requiring an infinite number of Steiner points. We address this problem by constructing spheres around the vertices which have a very small radius (at most ϵ times the shortest distance from the vertex to an edge that is not incident to the vertex). Figure 3.1

shows a vertex v of a face f and the constructed sphere (shown as circle) around this vertex with radius r_v. In the diagram, e is an edge of f incident to v and q_1 is the Steiner point on e that is closest to v. The distance between the next Steiner point along the edge (q_2) is chosen as $\epsilon|x|$, where x is as shown. We can see that as r_v decreases, x gets infinitely close to v and $|x| \to 0$ which implies that $|\overline{q_1 q_2}| \to 0$ as well. Hence, as r_v decreases to be infinitesimal, an infinite number of Steiner points need to be placed on e.

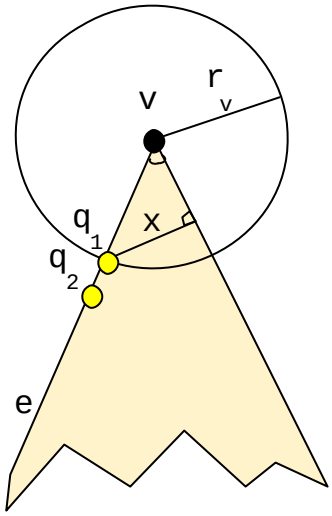

Figure 3.1: A sphere of radius r_v is placed around vertex v in order to make $|\overline{q_i q_{i+1}}|$ finite.

The idea behind the spheres is that they provide a way to ensure that only a finite number of Steiner points are added to each edge of $\mathcal{P}$. The spheres allow us to put a lower bound on the length of the smallest possible edge that passes between two adjacent Steiner points. The graph construction procedure never adds Steiner points within these spheres. Any shortest path segments inside the sphere can be bounded with respect to the portions of the path that are outside the spheres.

We can show that for any given shortest path, there exists a path in this graph with cost that is within $(1+\epsilon)$ times the shortest path cost. For the purpose of simplifying the proofs, we actually show that the approximation is within the bound of $(1+f(\epsilon))$ times the shortest path costs where $f(\epsilon)$ varies according to the type of query and the weight metric being used (i.e., Euclidean or weighted). Our bounds make an assumption that $0 < \epsilon < \frac{1}{2}$ and also include a term $\frac{W}{w}$ which represents the largest to smallest weight ratio of the faces of $\mathcal{P}$.

For the case in which both s and t are vertices of $\mathcal{P}$ we show a bound of $(1+(\frac{3-2\epsilon}{1-2\epsilon})\epsilon)$ times the shortest path length in the unweighted scenario and within the bound of $(1+(2+\frac{2W}{(1-2\epsilon)w})\epsilon)$ times the shortest path cost in the weighted scenario. The desired ϵ-approximation is achieved by dividing ϵ by $\frac{3-2\epsilon}{1-2\epsilon}$ or $(2+\frac{2W}{(1-2\epsilon)w})$ for the unweighted and weighted case, respectively. When either or both of s and t is not a vertex of $\mathcal{P}$, we give slightly worse bounds. As will be seen, the bounds appear somewhat complicated. We can simplify the bounds of our algorithm when $\epsilon \leq 1/6$ and also by taking the limit as $\epsilon \to 0$ (see Table 3.2 for the bounds of three different types of query problems in the unweighted and weighted cases. Note that "V to V" represents paths between vertices of $\mathcal{P}$, "P to P" represents paths between arbitrary points of $\mathcal{P}$ and "V to P" represents a path between a vertex and an arbitrary point).

Query Type	Unweighted $\epsilon \leq 1/6$	Unweighted $\epsilon \to 0$	Weighted $\epsilon \leq 1/6$	Weighted $\epsilon \to 0$
V to V	$1+4\epsilon$	$1+3\epsilon$	$1+(2+3W/w)\epsilon$	$1+(2+2W/w)\epsilon$
V to P	$1+8\epsilon$	$1+5\epsilon$	$1+(4+5W/w)\epsilon$	$1+(3+3W/w)\epsilon$
P to P	$1+13\epsilon$	$1+9\epsilon$	$1+(6+10W/w)\epsilon$	$1+(4+6W/w)\epsilon$

Table 3.2: The resulting bounds obtained by our algorithm when $\epsilon \leq 1/6$ and when $\epsilon \to 0$ for three different types of queries.

The running time of our algorithm is the cost for computing the graph G plus that

of running a shortest path algorithm in G. The graph consists of $|V| = nm$ vertices and $|E| = nm^2$ edges where $m = O(\log_\delta(|L|/r))$, $|L|$ is the length of the longest edge, r is ϵ times the minimum distance from any vertex to the boundary of the union of its incident faces (denoted as minimum height h of any face), and $\delta \geq 1 + \epsilon \sin\theta$, where θ is the minimum angle between any two adjacent edges of $\mathcal{P}$.

Our analysis reveals the exact relationship between the geometric parameters and the algorithm's running time. The dependence on geometric parameters is an interesting feature of several approximation geometric algorithms. Many researchers have advocated the use of geometric parameters in analyzing the performance of geometric algorithms, and our result indicates that if the geometric parameters are "well-behaved" then the asymptotic complexity of our algorithms is several orders of magnitude better than existing ones. One of the conclusions from our study is that while studying the performance of geometric algorithms, geometric parameters (e.g., fatness, density, aspect ratio, longest, closest) should not be ignored, and in fact it could potentially be very useful to express the performance that includes the relevant geometric parameters. [1]

3.2 An ϵ-Approximation Scheme Between Vertices

In this section we present an approximation scheme for computing a shortest path in the case in which s and t are vertices of $\mathcal{P}$. We first describe the discretization of the problem by constructing a graph G which depends on the choice of ϵ. We assume that $0 < \epsilon < \frac{1}{2}$ and is chosen before the graph is constructed. The ϵ-approximation path $\pi'(s,t)$ (or $\Pi'(s,t)$) that we report will be a shortest path in G between s and t. Lastly, we analyze the bounds for the unweighted and weighted cases, respectively.

[1] De Berg et al. [36] provide a discussion on the relations between the properties of various data models and their use with different algorithms.

3.2.1 Constructing the Graph

For each vertex v of face f_i we do the following: Let e_q and e_p be the edges of f_i incident to v. First, place Steiner points on edges e_q and e_p at distance r_v from v; call them q_1 and p_1, respectively. By definition, $|\overline{vq_1}| = |\overline{vp_1}| = r_v$. Define $\delta = (1 + \epsilon \sin\theta_v)$ if $\theta_v < \frac{\pi}{2}$, otherwise set $\delta = (1 + \epsilon)$. We now add Steiner points $q_2, q_3, ..., q_{\mu_q - 1}$ along e_q such that $|\overline{vq_j}| = r_v \delta^{j-1}$ where $\mu_q = \log_\delta(|e_q|/r_v)$. Similarly, add Steiner points $p_2, p_3, ..., p_{\mu_p - 1}$ along e_p, where $\mu_p = \log_\delta(|e_p|/r_v)$. This strategy creates sets of Steiner points along edges e_q and e_p (see Figure 3.2a). The following claim bounds the distance between adjacent Steiner points on e_q. Note that a similar claim applies to the Steiner points on e_p.

Claim 3.1 $|\overline{q_i q_{i+1}}| = \epsilon \cdot dist(q_i, e_p)$ *where* $0 < i < \mu_q$.

Proof: Assume first that $\theta_v < \frac{\pi}{2}$. We can manipulate $|\overline{q_i q_{i+1}}|$ as follows:

$$\begin{aligned} |\overline{q_i q_{i+1}}| &= |\overline{vq_{i+1}}| - |\overline{vq_i}| \\ &= r_v\delta^i - r_v\delta^{i-1} \\ &= r_v\delta^{i-1}(\delta - 1) \\ &= |\overline{vq_i}|(\delta - 1) \end{aligned}$$

By construction, $\delta = (1 + \epsilon \sin\theta_v)$, and so

$$\begin{aligned} |\overline{q_i q_{i+1}}| &= |\overline{vq_i}|(1 + \epsilon \sin\theta_v - 1) \\ &= \epsilon \sin\theta_v |\overline{vq_i}| \end{aligned}$$

Since $\sin\theta_v = \frac{\text{dist}(q_i, e_p)}{|\overline{vq_i}|}$ by definition, then $|\overline{q_i q_{i+1}}| = \epsilon \cdot \text{dist}(q_i, e_p)$.

If $\theta_v \geq \frac{\pi}{2}$, then by definition, $\delta = (1 + \epsilon)$, and so $|\overline{q_i q_{i+1}}| = |\overline{vq_i}|(1 + \epsilon - 1) = \epsilon|\overline{vq_i}|$.

Since $\text{dist}(q_i, e_p) = |\overline{vq_i}|$ by definition when $\theta_v \geq \frac{\pi}{2}$, then $|\overline{q_i q_{i+1}}| = \epsilon \cdot \text{dist}(q_i, e_p)$. □

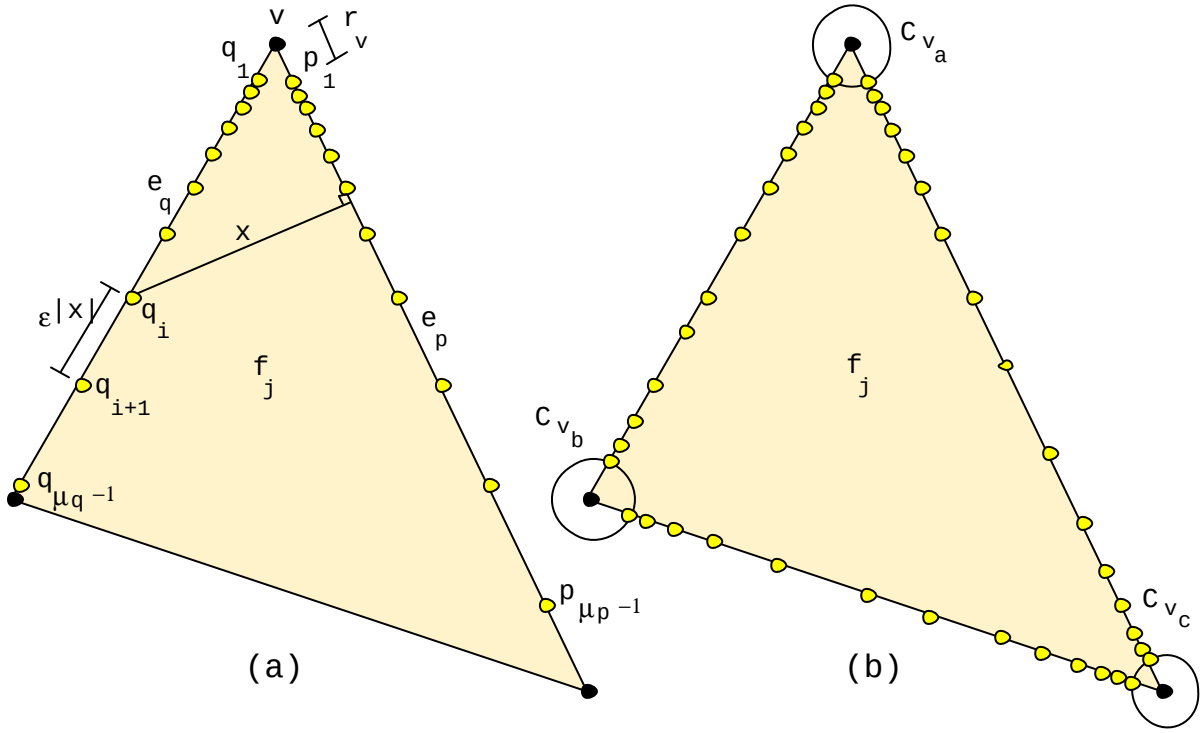

Figure 3.2: a) Placement of Steiner points on the edges of f_i that are incident to vertex v. b) Results of merging Steiner points along edges.

Claim 3.2 $|\overline{q_i q_{i+1}}| \leq (1+\epsilon)|\overline{q_{i-1} q_i}|$ *where* $1 < i < \mu_q$.

Proof: Similar to Claim 3.1, we can show that $|\overline{q_{i-1} q_i}| = |\overline{vq_i}| - |\overline{vq_{i-1}}| = |\overline{vq_{i-1}}|(\delta - 1)$ and $|\overline{q_i q_{i+1}}| = |\overline{vq_{i+1}}| - |\overline{vq_i}| = |\overline{vq_i}|(\delta - 1)$. Hence,

$$\begin{aligned}
|\overline{q_i q_{i+1}}| &= \left(|\overline{q_{i-1} q_i}| + |\overline{vq_{i-1}}|\right)(\delta - 1) \\
&= \left(|\overline{q_{i-1} q_i}| + \frac{|\overline{vq_{i-1}}|}{\delta - 1}\right)(\delta - 1) \\
&= ((\delta - 1) + 1)|\overline{q_{i-1} q_i}| \\
&= \delta |\overline{q_{i-1} q_i}| \\
&\leq (1+\epsilon)|\overline{q_{i-1} q_i}|
\end{aligned}$$

□

Since we have added Steiner points based on the minimum angle θ_v about v, we obtain "concentric parallel wavefronts" centered at v consisting of Steiner point layers along the incident edges of v. Since this construction is made for each vertex of a face f_i, there will be two overlapping sets of Steiner points on each edge of f_i. To eliminate this overlap, we reduce the number of Steiner points on each edge. If two sets of Steiner points on an edge originate from the endpoints of an edge e, we determine the point on e where the interval sizes from each set are equal and eliminate all larger intervals. Intuitively, intervals are eliminated from one set if there are small intervals in the other set that overlap with it (see Figure 3.2b). The example of Figure 3.3 shows the effect of applying this to each face of $\mathcal{P}$. Through shading, the example shows the association between the Steiner points and the vertex that introduced them. The vertices of G_i will be Steiner points as well as the vertices of

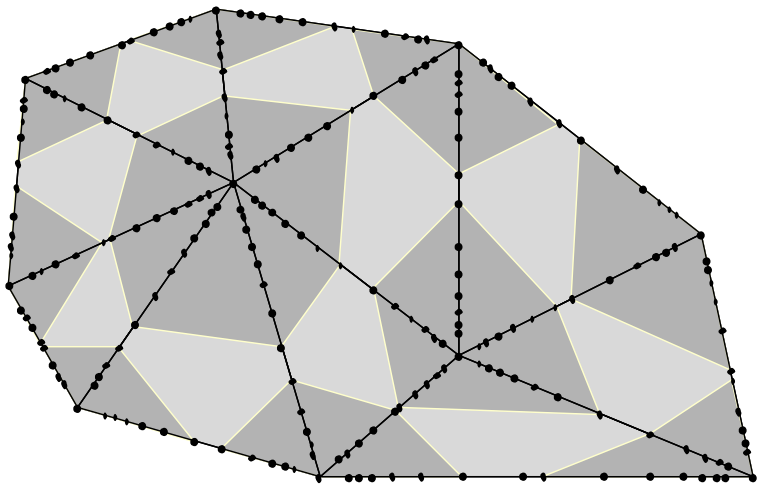

Figure 3.3: Example showing the association between the Steiner points and the vertex that introduced them.

$\mathcal{P}$ defining f_i. The edges of G_i form a complete graph on its vertices. The graph G

is defined to be the union $G_1 \cup G_2 \cup ... \cup G_n$. Observe that G is connected.

Claim 3.3 *At most* $m \leq 2(1 + \log_\delta(|L|/r))$ *Steiner points are added to each edge of* f_i, *for* $1 \leq i \leq n$.

Proof: Examining the case in which m is maximized, it is easily seen that this would happen when the edge has length $|L|$ and it is joining two vertices which have the minimum radius r. The bound on m follows from construction of G_i, in which we choose these worst-case parameters. The merging step from the edge vertices results in an additional factor of 2. Note that most edges of $\mathcal{P}$ will not have length L nor, in general, will its vertices have sphere radii of r. Hence in general, edges will have a smaller value of m that that stated here.

□

Claim 3.4 G *has* $O(n \log_\delta(|L|/r))$ *vertices and* $O(n(\log_\delta(|L|/r))^2)$ *edges.*

Proof: Since there are $O(n)$ edges of $\mathcal{P}$, Claim 3.3 implies that G contains $O(nm) = O(n \log_\delta(|L|/r))$ vertices. The number of edges in G is n times the number of edges in each G_i. Since each G_i is a complete graph on $O(m)$ vertices, it has $O(m^2)$ edges and therefore there are $O(n(\log_\delta(|L|/r))^2)$ edges in G.

□

3.2.2 Accuracy Bound of the Approximation

This section provides an analysis of the approximated path accuracy for both unweighted and weighted paths produced for different placements of query vertices s and t. We present a construction to show that there exists a path between s and t, with cost at most $f(\epsilon)$ times the length of the actual shortest path cost. We will show

that $f(\epsilon)$ varies depending on the distance between s and t and also on the metric used (i.e., unweighted vs. weighted). Before beginning the proofs we establish several properties of this graph and some additional definitions.

Consider a subgraph G_j, $1 \leq j \leq n$, as defined above. Let v be a vertex of a face f_j with edges e_p and e_q incident to v. We need the following technical lemma.

Lemma 3.1 *Let s_i be the smallest segment contained within f_j such that one endpoint of s_i intersects e_q between q_i and q_{i+1} and the other endpoint intersects e_p. It holds that $|\overline{q_i q_{i+1}}| \leq \epsilon |s_i|$. Furthermore, if $\theta_v < \frac{\pi}{2}$ then s_i is a perpendicular to e_p and if $\theta_v \geq \frac{\pi}{2}$ then $|s_i| \geq |\overline{vq_i}|$.*

Proof: Consider the case in which $\theta_v < \frac{\pi}{2}$. Clearly, the smallest segment starting in the interval $[q_i, q_{i+1}]$ on e_q is a perpendicular to line e_p since e_q and e_p are not parallel. Furthermore, the smallest such perpendicular must start at q_i. Therefore $|s_i| \geq \text{dist}(q_i, e_p)$. Since Claim 3.1 ensures that $|\overline{q_i q_{i+1}}| = \epsilon \cdot \text{dist}(q_i, e_p)$, then $|\overline{q_i q_{i+1}}| \leq \epsilon |s_i|$. If $\theta_v \geq \frac{\pi}{2}$ then smallest segment starting in the interval $[q_i, q_{i+1}]$ on e_q is $\overline{vq_i}$. Therefore $|s_i| \geq |\overline{vq_i}| = \text{dist}(q_i, e_p)$. Once again, Claim 3.1 is used to show that $|\overline{q_i q_{i+1}}| \leq \epsilon |s_i|$.

□

Let s_i be a segment of $\pi(s,t)$ (or $\Pi(s,t)$) crossing face f_i. Each s_i, must be of one of the following types of segments:

Definition 3.1 Outside-sphere: *A segment such that $s_i \cap C_v = \emptyset$.*

Definition 3.2 Overlapping-sphere: *A segment such that $s_i \cap C_v =$ subsegment of s_i.*

Definition 3.3 Inside-sphere: *A segment such that $s_i \cap C_v = s_i$.*

Let $C_{\sigma_1}, C_{\sigma_2}, ..., C_{\sigma_\kappa}$ be a sequence of spheres (listed in order from s to t) intersected by overlapping-sphere segments of $\pi(s,t)$ such that $C_{\sigma_j} \neq C_{\sigma_{j+1}}$. Define subpaths of $\pi(s,t)$ (and $\Pi(s,t)$) as being one of two kinds:

Definition 3.4 Between-sphere subpath: *A path consisting of an overlapping-sphere segment followed by zero or more consecutive outside-sphere segments followed by an overlapping-sphere segment. These subpaths will be denoted as $\pi(\sigma_j, \sigma_{j+1})$ (denoted as $\Pi(\sigma_j, \sigma_{j+1})$ for weighted case) whose first and last segments intersect C_{σ_j} and $C_{\sigma_{j+1}}$, respectively. We will also consider paths that begin or/and end at a vertex to be a degenerate case of this type of path containing only outside-sphere segments.*

Definition 3.5 Inside-sphere subpath: *A path consisting of one or more consecutive inside-sphere segments all lying within the same C_{σ_j}; these are denoted as $\pi(\sigma_j)$ (denoted as $\Pi(\sigma_j)$ for weighted case).*

Note that inside-sphere subpaths of $\pi(s,t)$ (and $\Pi(s,t)$) always lie between two between-sphere subpaths. That is, $\pi(\sigma_j)$ lies between $\pi(\sigma_{j-1}, \sigma_j)$ and $\pi(\sigma_j, \sigma_{j+1})$.

Claim 3.5 *Let s_i be an outside-sphere segment with one endpoint between Steiner points q_j and q_{j+1} on edge e_q of a face f_i and the other endpoint between Steiner points p_k and p_{k+1} on edge e_p of f_i.*
Then $\max(\min(|\overline{q_j p_k}|, |\overline{q_j p_{k+1}}|), \min(|\overline{q_{j+1} p_k}|, |\overline{q_{j+1} p_{k+1}}|)) \leq (1+\epsilon)|s_i|$.

Proof: Let v be the vertex shared by e_q and e_p and the angle between e_p and e_q be θ_v. Define x_q (x_p) to be the closest point on e_p (e_q) from q_j (p_k). When $\theta_v < \frac{\pi}{2}$, there are five sub-cases corresponding to where x_p and x_q lie with respect to intervals $[q_j, q_{j+1}]$ and $[p_k, p_{k+1}]$, respectively as shown in Figure 3.4a) to e). When $\theta_v \geq \frac{\pi}{2}$ there is only one case as shown in Figure 3.4f. It is easily seen that cases d) and e)

are symmetrical to cases b) and c), respectively, and so the proof for cases d) and e) has been omitted. In each case the shortest possible s_i (called s_{min}) is used. In cases a, b, c and f it is shown that both $|\overline{q_j p_k}|$, $|\overline{q_{j+1} p_k}| \leq (1+\epsilon)|s_{min}|$ and in cases d) and e) it is shown that $|\overline{q_j p_{k+1}}|$, $|\overline{q_{j+1} p_{k+1}}| \leq (1+\epsilon)|s_{min}|$. Since $|s_i| \geq |s_{min}|$, the claim will hold.

Case a: Here, $s_{min} = \overline{q_j p_k}$. By construction, $|\overline{q_j q_{j+1}}| \leq \epsilon|\overline{q_j x_q}| \leq \epsilon|\overline{q_j p_k}|$. By triangle inequality $|\overline{q_{j+1} p_k}| \leq (1+\epsilon)|\overline{q_j p_k}| = (1+\epsilon)|s_{min}|$.

Cases b: Here, $s_{min} = \overline{p_k x_p}$. By construction, $|\overline{q_j q_{j+1}}| \leq \epsilon|\overline{q_j x_q}| \leq \epsilon|\overline{x_p p_k}|$. Furthermore, the triangle inequality ensures that $|\overline{q_{j+1} p_k}| < |\overline{q_{j+1} x_p}| + |\overline{x_p p_k}| \leq (1+\epsilon)|\overline{x_p p_k}|$. Similarly, $|\overline{q_j x_p}| < \epsilon|\overline{x_p p_k}|$. Therefore $|\overline{q_j p_k}| < (1+\epsilon)|\overline{p_k x_p}| = (1+\epsilon)|s_{min}|$.

Cases c: Here, $s_{min} = \overline{q_{j+1} p_k}$. Once again, using the triangle inequality we know that $|\overline{q_j p_k}| < |\overline{q_j q_{j+1}}| + |\overline{q_{j+1} p_k}|$. As before, $|\overline{q_j q_{j+1}}| \leq \epsilon|\overline{q_j x_q}| \leq \epsilon|\overline{q_{j+1} p_k}|$. Therefore $|\overline{q_j p_k}| < (1+\epsilon)|\overline{q_{j+1} p_k}| = (1+\epsilon)|s_{min}|$.

Case f: It is obvious that for any choice of p_k, $s_{min} = \overline{q_j p_k}$. By Claim 3.1, $|\overline{q_j q_{j+1}}| \leq \epsilon|\overline{v q_j}| \leq \epsilon|\overline{q_j p_k}| \leq \epsilon|\overline{q_{j+1} p_k}|$. Again from triangle inequality, it can be shown that $|\overline{q_{j+1} p_k}| \leq |\overline{q_j q_{j+1}}| + |\overline{q_j p_k}| \leq (1+\epsilon)|s_{min}|$.

□

Claim 3.6 *Let s_i be an overlapping-sphere segment crossing edge e_q of f_i between Steiner points q_j and q_{j+1} and crossing e_p between v and Steiner point p_1, where $j \geq 1$ and v is the vertex common to e_q and e_p. Then $|\overline{q_1 q_j}|$ and $|\overline{q_1 q_{j+1}}|$ are less than $(1+\epsilon)|s_i|$.*

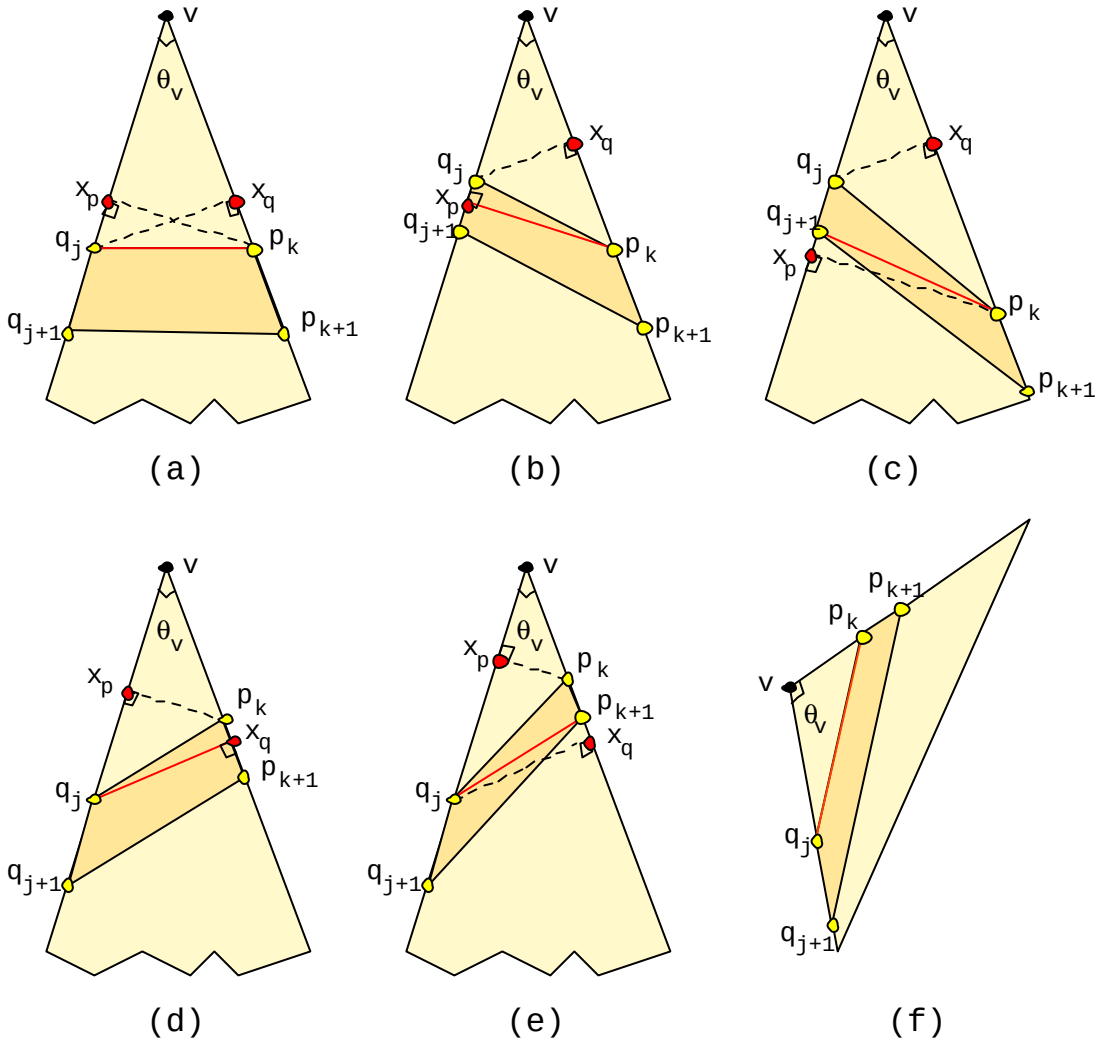

Figure 3.4: Six possible combinations of intervals containing s.

Proof: $|\overline{vq_1}| = |\overline{vp_1}|$ and hence $\phi = \angle q_j q_1 p_1 > \frac{\pi}{2}$ (see Figure 3.5). Since $\phi > \frac{\pi}{2}$, then $|\overline{q_1 q_j}| < |s_i|$. The construction of G_i ensures that, $|\overline{q_1 q_{j+1}}| < (1 + \epsilon \sin\theta_v)|\overline{q_1 q_j}|$ if $\theta_v < \frac{\pi}{2}$. If $\theta_v \geq \frac{\pi}{2}$ then $|\overline{q_1 q_{j+1}}| < (1+\epsilon)|\overline{q_1 q_j}|$. Thus, $|\overline{q_1 q_{j+1}}| < (1+\epsilon)|s_i|$. □

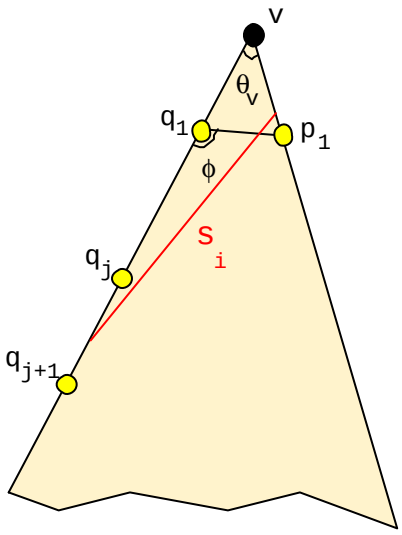

Figure 3.5: A segment s_i passing through intervals defined by $[q_j, q_{j+1}]$ and $[v, p_1]$.

Consider a shortest path $\pi(s,t)$ in $\mathcal{P}$. It is composed of several segments which go through faces, and/or along edges, and/or through spheres around vertices. For segments of $\pi(s,t)$ that are not completely contained inside spheres, we show that there exists an appropriate edge in the graph with cost at most $1 + f(\epsilon)$ times this segment. In the unweighted scenario, we show that $f(\epsilon) = (\frac{3-2\epsilon}{1-2\epsilon})\epsilon$ and in the weighted scenario, $f(\epsilon) = (2 + \frac{2W}{(1-2\epsilon)w})\epsilon$. For segments that are lying inside a sphere, we use a "charging" argument where the cost of the approximated segment is "charged" to the cost of the preceding between-sphere subpath in $\pi'(s,t)$.

3.2.2.1 Constructing an Unweighted Path for Analysis

In this section, we describe the construction of an approximate path $\pi'(s,t)$ in G, given a shortest Euclidean path $\pi(s,t)$ in $\mathcal{P}$. The algorithm will either compute this path or a path with equal or better cost. Clearly, if s and t are endpoints of the same edge, then $\pi'(s,t) = \overline{st}$ and the approximation is exact. We will therefore assume that s and t do not satisfy this condition and hence there exists at least one between-sphere subpath in $\pi(s,t)$. More specifically, we will assume that t lies outside of τ_s.

Consider a between-sphere subpath $\pi(\sigma_j, \sigma_{j+1})$, which consists only of outside-sphere and overlapping-sphere segments types. First examine an outside-sphere segment s_i of $\pi(\sigma_j, \sigma_{j+1})$ that passes through edges e_q and e_p of face f_i. Assume s_i intersects e_q between Steiner points q_j and q_{j+1} and also intersects e_p between Steiner points p_k and p_{k+1}, where $j, k \geq 1$. The approximated path is chosen such that it enters face f_i through Steiner point q_j or q_{j+1}. Without loss of generality assume that the approximated path enters f_i at q_j. Choose s'_i to be the shortest of $\overline{q_j p_k}$ and $\overline{q_j p_{k+1}}$. It is easily seen that $\pi'(\sigma_j, \sigma_{j+1})$ is connected since adjacent segments s'_{i-1} and s'_{i+1} share an endpoint (i.e., a Steiner point).

Now examine an overlapping-sphere segment of $\pi(\sigma_j, \sigma_{j+1})$; this can appear as the first or last segment. Without loss of generality assume that it is the last segment. Let this segment enter f_i between Steiner points q_j and q_{j+1} and exit between vertex $v_{\sigma_{j+1}}$ and Steiner point p_1 on e_p. Let $s'_i = \overline{q_j q_1}$ (if s_i is the first segment, then we either choose s'_i to be $\overline{p_j p_1}$ or $\overline{p_{j+1} p_1}$ depending on at which Steiner point the approximated path enters f_{i+1}). It is easily seen that the combination of these approximated segments forms a connected chain of edges in G which we will call $\pi'(\sigma_j, \sigma_{j+1})$. One crucial property of $\pi'(\sigma_j, \sigma_{j+1})$ is that it begins at a point where C_{σ_j} intersects an edge of $\mathcal{P}$ and ends at a point where $C_{\sigma_{j+1}}$ intersects an edge of $\mathcal{P}$.

Consider now two consecutive between-sphere subpaths of $\pi(s,t)$, say $\pi'(\sigma_{j-1},\sigma_j)$ and $\pi'(\sigma_j,\sigma_{j+1})$. They are disjoint from one another, however, the first path ends at sphere C_{σ_j} and the second path starts at C_{σ_j}. Join the end of $\pi'(\sigma_{j-1},\sigma_j)$ and the start of $\pi'(\sigma_j,\sigma_{j+1})$ to vertex v_{σ_j} by two segments (which are edges of G). These two segments together form an inside-sphere subpath and will be denoted as $\pi'(\sigma_j)$. This step is repeated for each consecutive pair of between-sphere subpaths so that all subpaths are joined to form $\pi'(s,t)$. (The example of Figure 3.6 shows how between-sphere subpaths are connected to inside-sphere subpaths.) Constructing a path in this manner results in a continuous path that lies on the surface of $\mathcal{P}$.

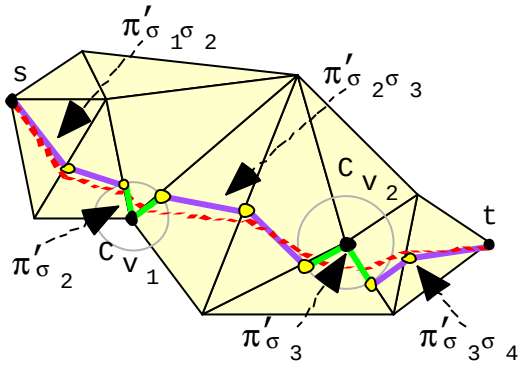

Figure 3.6: An example showing the between-sphere and inside-sphere subpaths that connect to form the approximated path $\pi(s,t)$.

3.2.2.2 Bounding the Unweighted Path

We now bound the path that was chosen in the previous section. Once again, it should be pointed out that since we are bounding a particular path which we have chosen, any path chosen by Dijkstra's algorithm will have cost at most equal to this one.

Claim 3.7 *Let $\pi'(\sigma_{j-1}, \sigma_j)$ be a between-sphere subpath of $\pi'(s,t)$ corresponding to an approximation of $\pi(\sigma_{j-1}, \sigma_j)$. Then $|\pi'(\sigma_{j-1}, \sigma_j)| \leq (1+\epsilon)|\pi(\sigma_{j-1}, \sigma_j)|$, where $0 < \epsilon < 1$.*

Proof: The proof follows from Claim 3.5 and Claim 3.6 since segments of $\pi'(\sigma_{j-1}, \sigma_j)$ are either outside-sphere or overlapping-sphere only.

□

Claim 3.8 *Let $\pi'(\sigma_{j-1}, \sigma_j)$ be a between-sphere subpath of $\pi'(s,t)$ corresponding to an approximation of $\pi(\sigma_{j-1}, \sigma_j)$ where $1 < j \leq \kappa$. Then $|\pi'(\sigma_j)| \leq \frac{2\epsilon}{1-2\epsilon}|\pi(\sigma_{j-1}, \sigma_j)|$, where $0 < \epsilon < \frac{1}{2}$.*

Proof: From Property 1.3 it follows that the distance between $C_{\sigma_{j-1}}$ and C_{σ_j} must be at least $(1-2\epsilon)\max(h_{v_{\sigma_j}}, h_{v_{\sigma_{j-1}}})$ which is at least $(1-2\epsilon)h_{v_{\sigma_j}}$. Since $\pi(\sigma_{j-1}, \sigma_j)$ is a between-sphere subpath, it intersects both $C_{\sigma_{j-1}}$ and C_{σ_j}. Thus $|\pi(\sigma_{j-1}, \sigma_j)| \geq (1-2\epsilon)h_{v_{\sigma_j}}$. By definition, $\pi'(\sigma_j)$ consists of exactly two segments which together have length satisfying $|\pi'(\sigma_j)| = 2r_{v_{\sigma_j}} = 2\epsilon h_{v_{\sigma_j}}$. Thus, $|\pi(\sigma_{j-1}, \sigma_j)| \geq \left(\frac{1-2\epsilon}{2\epsilon}\right)|\pi'(\sigma_j)|$ which can be re-written as $|\pi'(\sigma_j)| \leq \frac{2\epsilon}{1-2\epsilon}|\pi(\sigma_{j-1}, \sigma_j)|$.

□

Lemma 3.2 *Let s be a vertex of $\mathcal{P}$ and p be a Steiner point on $\mathcal{P}$. Let s' and p' be the vertices in G corresponding to s and p, respectively. If $\pi(s,p)$ is a shortest path in $\mathcal{P}$ then there exists an approximated path $\pi'(s',p') \in G$ for which $|\pi'(s',p')| \leq \left(1 + \frac{3-2\epsilon}{1-2\epsilon}\epsilon\right)|\pi(s,p)|$, where $0 < \epsilon < \frac{1}{2}$.*

Proof: Using the results of Claim 3.7 and Claim 3.8, we can "charge" the cost of each inside-sphere subpath $\pi'(\sigma_j)$ to the between-sphere subpath $\pi'(\sigma_{j-1}, \sigma_j)$ as follows:

$$\begin{aligned} |\pi'(\sigma_{j-1}, \sigma_j)| + |\pi'(\sigma_j)| &\leq (1+\epsilon)|\pi(\sigma_{j-1}, \sigma_j)| + \left(\frac{2\epsilon}{1-2\epsilon}\right)|\pi(\sigma_{j-1}, \sigma_j)| \\ &= \left(1 + \left(\frac{3-2\epsilon}{1-2\epsilon}\right)\epsilon\right)|\pi(\sigma_{j-1}, \sigma_j)| \end{aligned}$$

The union of all subpaths $\pi'(\sigma_{j-1}, \sigma_j)$ and $\pi'(\sigma_j)$ form $\pi'(s', p')$ where $2 \leq j \leq \kappa$. Hence, we have bounded $|\pi'(s', p')|$ with respect to the between-sphere subpaths of $\pi(s, p)$. Therefore

$$|\pi'(s', p')| \leq \left(1 + \left(\frac{3 - 2\epsilon}{1 - 2\epsilon}\right)\epsilon\right) \sum_{j=2}^{\kappa} |\pi(\sigma_{j-1}, \sigma_j)| \leq \left(1 + \left(\frac{3 - 2\epsilon}{1 - 2\epsilon}\right)\epsilon\right) |\pi(s, p)|$$

□

Corollary 3.1 *If $\pi(s, t)$ is a shortest path in $\mathcal{P}$, where s and t are vertices of $\mathcal{P}$ then there exists an approximated path $\pi'(s, t) \in G$ for which $|\pi'(s, t)| \leq \left(1 + \left(\frac{3-2\epsilon}{1-2\epsilon}\right)\epsilon\right) |\pi(s, t)|$, where $0 < \epsilon < \frac{1}{2}$.*

Proof: The proof follows from Lemma 3.2 since t is also a vertex of G.

□

3.2.2.3 Constructing a Weighted Path for Analysis

Some additional shortest path properties arise in the weighted scenario. If s_i is an edge-using segment with s_{i-1} and s_{i+1} being the preceding and succeeding segments of s_i in $\Pi(s, t)$, respectively, then the following properties hold:

Property 3.1 *Segments s_{i-1} and s_{i+1} must be face-crossing.*

Property 3.2 *If s_i is an outside-sphere segment, then s_{i-1} and s_{i+1} cannot be inside-sphere segments.*

Given a shortest path $\Pi(s, t)$, we construct a path $\Pi'(s, t)$ in a similar manner as in the unweighted scenario. However, we must consider the approximation of edge-using segments since they may no longer span the full length of the edge which they are

using. Consider an edge-using segment s_i of $\Pi(s,t)$ on edge e_p of $\mathcal{P}$ with endpoints lying in Steiner point intervals $[p_y, p_{y+1}]$ and $[p_{u-1}, p_u]$ along e_p, where $y < u$. Let s_{i-1} and s_{i+1}, respectively, be the two crossing segments representing the predecessor and successor of s_i in the sequence of segments in $\Pi(s,t)$ (see Figure 3.7a)). Without loss of generality, we will assume that two such edges exist although it is possible that s_{i-1} and s_i meet at a vertex of $\mathcal{P}$; which can easily be handled as well. We choose an approximation s'_i of s_i to be either $\overline{p_y p_{u-1}}$ or $\overline{p_{y+1} p_{u-1}}$ depending on whether s'_{i-1} intersects e_p at p_y or p_{y+1}, respectively (see Figure 3.7 b) and c)). Note that we make sure to choose s'_i so that it is connected to s'_{i-1}. Of course, s'_{i+1} will also be chosen to ensure connectivity with s'_i. In the degenerate case where $u = y + 1$, there is no approximation for s_i. Instead, s'_{i-1} is connected directly to s'_{i+1} (see Figure 3.8). It should be noted that Dijkstra's algorithm will never choose such a subpath since it does not make use of e_p. That is, it would be more cost-effective to take a shortcut directly between the two "non-shared" endpoints of s_{i-1} and s_{i+1} (see Figure 3.9).

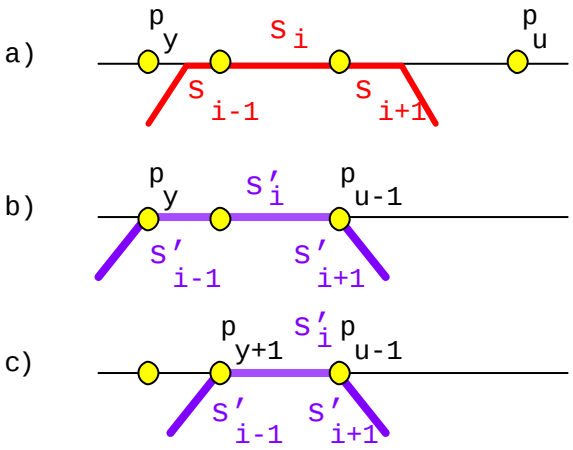

Figure 3.7: a) The edge-using segment s_i and face crossing segments s_{i-1}, s_{i+1} and b),c) the two possible choices of approximating it with s'_i.

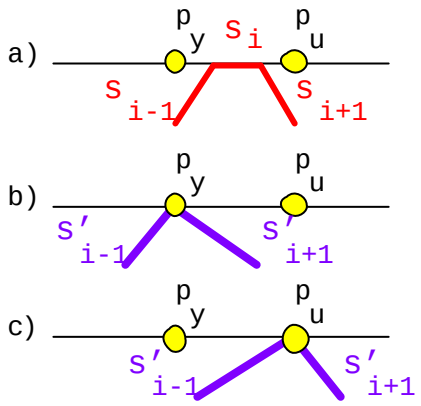

Figure 3.8: a) The degenerate edge-using segment s_i and face crossing segments s_{i-1}, s_{i+1} and b),c) the two possible choices of connecting s'_{i-1} with s'_{i+1}.

3.2.2.4 Bounding the Weighted Path

As in the unweighted case, we bound the chosen approximation path.

Claim 3.9 *Each face-crossing segment s_i of $\Pi(\sigma_j, \sigma_{j+1})$ is approximated by a segment s'_i of $\Pi'(\sigma_j, \sigma_{j+1})$ for which $\|s'_i\| \leq (1+\epsilon)\|s_i\|$.*

Proof: First, consider s_i to be an outside-sphere edge of $\Pi(\sigma_j, \sigma_{j+1})$. Claim 3.5 has shown that there exists an edge s'_i of G_i that has length at most $(1+\epsilon)|s_i|$. Since both s_i and s'_i cross the face f_i, the same weight is applied to each segment and hence $\|s'_i\| = w_i|s'_i| \leq w_i(1+\epsilon)|s_i| = (1+\epsilon)\|s_i\|$. Assume that s_i is an overlapping-sphere segment of $\Pi(\sigma_j, \sigma_{j+1})$. Claim 3.6 guarantees the existence of an edge-using segment s'_i of G_i of length at most $(1+\epsilon)|s_i|$. Although s'_i is edge-using and s_i is face-crossing, Property 1.12 ensures that the weight of s'_i will never be greater than the weight of s_i. Hence $\|s'_i\| = w_i|s'_i| \leq w_i(1+\epsilon)|s_i| = (1+\epsilon)\|s_i\|$.

□

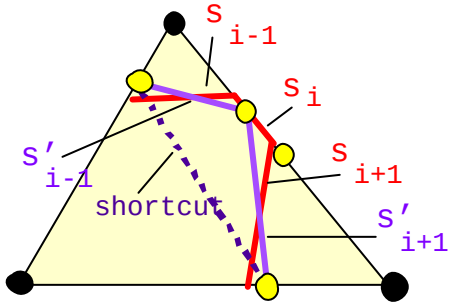

Figure 3.9: The shortcut that would be taken by our algorithm to join the "non-shared" endpoints of s_{i-1} and s_{i+1} in the degenerate case where s_{i-1} and s_{i+1} share an endpoint.

Claim 3.10 *Let s_i be an edge-using segment of a between-sphere subpath $\Pi(\sigma_j, \sigma_{j+1})$ and let s_{i-1} be the segment of $\Pi(\sigma_j, \sigma_{j+1})$ preceding s_i. There exists a segment s'_i of $\Pi'(\sigma_j, \sigma_{j+1})$ for which $\|s'_i\| \leq \|s_i\| + \epsilon\|s_{i-1}\|$.*

Proof: We will assume that s_i is an outside-sphere segment since it is in a between-sphere subpath and is preceded by a segment s_{i-1} (which itself may be an outside-sphere or overlapping-sphere segment). The proof can be easily modified for the degenerate case where s_i is an overlapping-sphere segment (i.e., first or/and last segment of $\Pi(\sigma_j, \sigma_{j+1})$) such that the claim also holds. First consider the case when s_i lies between Steiner points p_y and p_u on edge e_p of face f_i, where $y+1 < u-1$. We will approximate s_i with s'_i as depicted in one of the two cases of Figure 3.7. Clearly, the portion of s'_i lying between p_{y+1} and p_{u-1} has length at most $|s_i|$. Since s_{i-1} has an endpoint between p_y and p_{y+1}, Lemma 3.1 ensures that $|p_y p_{y+1}| \leq \epsilon|s_{i-1}|$. Since our construction of $\Pi(s,t)$ allows us to approximate s_i by either $\overline{p_y p_{u-1}}$ or $\overline{p_{y+1} p_{u-1}}$ we obtain that $|s'_i| \leq |p_y p_{y+1}| + |p_{y+1} p_{u-1}|$. Hence, $|s'_i| \leq \epsilon|s_{i-1}| + |s_i|$. In the case where s_i lies between Steiner points p_y and p_{y+1} then again Lemma 3.1 implies that $|s'_i| \leq \epsilon|s_{i-1}|$ which certainly satisfies $|s'_i| \leq \epsilon|s_{i-1}| + |s_i|$. Again from Property 1.12

it follows that $\|s'_i\| \leq \|s_i\| + \epsilon\|s_{i-1}\|$.

□

Lemma 3.3 *If $\Pi'(\sigma_{j-1}, \sigma_j)$ is a between-sphere subpath of $\Pi'(s,t)$ corresponding to an approximation of $\Pi(\sigma_{j-1}, \sigma_j)$ then $\|\Pi'(\sigma_{j-1}, \sigma_j)\| \leq (1+2\epsilon)\|\Pi(\sigma_{j-1}, \sigma_j)\|$.*

Proof: If s'_i is an approximation of a face-crossing segment s_i, then by Claim 3.9 $\|s'_i\| \leq (1+\epsilon)\|s_i\|$. If s'_i is an edge-using segment then from Claim 3.10 follows that $\|s'_i\| \leq \|s_i\| + \epsilon\|s_{i-1}\|$, where s_{i-1} is a face-crossing segment. We "charge" the portion $\epsilon\|s_{i-1}\|$ of $\|s'_i\|$ to the cost of s'_{i-1} so that $\|s'_{i-1}\| \leq (1+2\epsilon)\|s_{i-1}\|$. The remaining portion indicates that $\|s'_i\| \leq \|s_i\|$. Hence, each segment s'_i of $\Pi'(\sigma_{j-1}, \sigma_j)$ has cost at most $(1+2\epsilon)\|s_i\|$ and $\|\Pi'(\sigma_{j-1}, \sigma_j)\| \leq (1+2\epsilon)\|\Pi(\sigma_{j-1}, \sigma_j)\|$.

□

Claim 3.11 *Let $\Pi'(\sigma_{j-1}, \sigma_j)$ be a between-sphere subpath of $\Pi'(s,t)$ corresponding to an approximation of $\Pi(\sigma_{j-1}, \sigma_j)$ then $\|\Pi'(\sigma_j)\| \leq \frac{2\epsilon W}{(1-2\epsilon)w}\|\Pi(\sigma_{j-1}, \sigma_j)\|$ where $0 < \epsilon < \frac{1}{2}$.*

Proof: Using a similar proof as for Claim 3.7, it can be shown that $|\Pi'(\sigma_j)| \leq \frac{2\epsilon}{1-2\epsilon}|\Pi(\sigma_{j-1}, \sigma_j)|$. The two edges of $\Pi'(\sigma_j)$ are chosen so as to pass along two particular edges of $\mathcal{P}$, depending on the face containing the overlapping-sphere edges of $\Pi(\sigma_{j-1}, \sigma_j)$ and $\Pi(\sigma_j, \sigma_{j+1})$. These chosen edges may have a high weight associated with them, say W (i.e., the maximum face weight). Although the paths $\Pi(\sigma_{j-1}, \sigma_j)$ and $\Pi(\sigma_j, \sigma_{j+1})$ will have segments that pass through the same expensive faces, these segments may be arbitrarily short in length. Therefore $|\Pi'(\sigma_j)|$ may be small with respect to $|\Pi(\sigma_{j-1}, \sigma_j)|$, but it may have much higher weight and greater cost. Hence, in worst case $\|\Pi'(\sigma_j)\| = W|\Pi'(\sigma_j)|$ and $\|\Pi(\sigma_{j-1}, \sigma_j)\| = w|\Pi(\sigma_{j-1}, \sigma_j)|$. Therefore, it is only safe to say that $\|\Pi'(\sigma_j)\| \leq \frac{2\epsilon W}{(1-2\epsilon)w}\|\Pi(\sigma_{j-1}, \sigma_j)\|$.

□

We have made the assumption that $\Pi'(\sigma_j)$ consists of segments passing through faces that have weight W. Although this may be true in the worst case, we could use the maximum weight of any face adjacent to v_{σ_j}, which typically would be smaller than W. In addition, we have assumed that $\Pi'(\sigma_{j-1}, \sigma_j)$ traveled through faces with minimum weight. We could determine the smallest weight of any face through which $\Pi'(\sigma_{j-1}, \sigma_j)$ passes and use that in place of w. This would lead to a better bound.

Lemma 3.4 *If $\Pi(s,p)$ is a shortest weighted path in $\mathcal{P}$, where s is a vertex of $\mathcal{P}$ and p is a vertex of G then there exists an approximated path $\Pi'(s,p) \in G$ such that $\|\Pi'(s,p)\| \leq (1 + (2 + \frac{2W}{(1-2\epsilon)w})\epsilon)\|\Pi(s,p)\|$ where $0 < \epsilon < \frac{1}{2}$.*

Proof: Using the results of Claim 3.11 and Lemma 3.3, it can be shown that

$$\|\Pi'(\sigma_{j-1}, \sigma_j)\| + \|\Pi'(\sigma_j)\| \leq (1 + (2 + \frac{2W}{(1-2\epsilon)w})\epsilon)\|\Pi(\sigma_{j-1}, \sigma_j)\|.$$

This essentially "charges" the length of an inside-sphere subpath to a between-sphere subpath. The union of all such subpaths form $\Pi'(s,p)$. This allows us to approximate $\Pi'(s,p)$ within the bound of $1 + (2 + \frac{2W}{(1-2\epsilon)w})\epsilon$ times the total cost of all the between-sphere subpaths of $\Pi(s,p)$. Since $\Pi(s,p)$ has cost at least that of its between-sphere subpaths, $\|\Pi'(s,p)\| \leq (1 + (2 + \frac{2W}{(1-2\epsilon)w})\epsilon)\|\Pi(s,p)\|$.

□

Corollary 3.2 *If $\Pi(s,t)$ is a shortest weighted path in $\mathcal{P}$,where s and t are vertices of $\mathcal{P}$ then there exists an approximated path $\Pi'(s,t) \in G$ such that $\|\Pi'(s,t)\| \leq (1 + (2 + \frac{2W}{(1-2\epsilon)w})\epsilon)\|\Pi(s,t)\|$ where $0 < \epsilon < \frac{1}{2}$.*

Proof: The proof follows from Lemma 3.4 since t is also a vertex of G.

□

Theorem 3.1 *Let $0 < \epsilon < \frac{1}{2}$. Let $\mathcal{P}$ be a simple polyhedron with n faces and let s and t be two of its vertices. An approximation $\pi'(s,t)$ of a Euclidean shortest path $\pi(s,t)$ between s and t can be computed such that $|\pi'(s,t)| \leq (1 + \frac{3-2\epsilon}{1-2\epsilon}\epsilon)|\pi(s,t)|$. An approximation $\Pi'(s,t)$ of a weighted shortest path $\Pi(s,t)$ between s and t can be computed such that $\|\Pi'(s,t)\| \leq (1+(2+\frac{2W}{(1-2\epsilon)w})\epsilon)\|\Pi(s,t)\|$. The approximations can be computed in $O(mn \log mn + nm^2)$ time where $m = \log_\delta \frac{|L|}{r}$, and $\delta = (1 + \epsilon \sin\theta)$.*

Proof: For both cases, we showed that there exists a path in G that satisfies the claimed bounds using Corollary 3.1 and Corollary 3.2, respectively. Dijkstra's algorithm will either compute this path or a path with equal or better cost, and therefore the path computed by Dijkstra's algorithm as well satisfies the claimed approximation bounds. The running time of the algorithm follows from the size of the graph as stated in Claim 3.4. The variant of Dijkstra's algorithm (i.e., Theorem 1.2) which employs Fibonacci heaps [45] is employed to compute the path in the stated time bounds.

□

3.3 Modifying the Bounds For Arbitrary Query Points

Until now, the analysis of our ϵ-approximation algorithm has only considered the case in which both s and t are vertices of $\mathcal{P}$ (and hence vertices of G). This section explains how the approximation scheme can be extended to allow s and t to be arbitrarily chosen points on $\mathcal{P}$. We consider first the case in which s is a vertex of $\mathcal{P}$ and t lies arbitrarily on $\mathcal{P}$ (this correlates to the scenario of fixed source with queried destination) and then consider when s and t both lie arbitrarily on $\mathcal{P}$. We once again assume that t lies outside τ_s and show that both of these query-type scenarios will

result in different bounds on the approximated path produced. Of course, as stated earlier, we can always vary the number of vertices of G to obtain paths satisfying: $\|\Pi'(s,t)\| \leq (1+\epsilon)\|\Pi(s,t)\|$. For both of these query problems, we begin by giving the bound on the weighted approximation produced by the algorithm and then give a corollary that bounds the unweighted approximation.

3.3.1 Vertex to Arbitrary Point Approximations

Consider paths in which s is a vertex of $\mathcal{P}$ and t is any point on $\mathcal{P}$. Let f_t be the face containing t with weight W_{f_t} and let x be the point in which $\Pi(s,t)$ enters f_t (i.e., x lies on an edge e of f_t). Without loss of generality, assume that x is not a vertex or Steiner point of $\mathcal{P}$; the bound would be better in that case. If x lies within some sphere C_v around a vertex v of $\mathcal{P}$ then let $p = v$. Otherwise, let p be the Steiner point on e that is closest to x. We construct the approximation $\Pi'(s,t)$ to be the concatenation of $\{\Pi'(s,p), \overline{pt}\}$. It is this path which we give a bound for, keeping in mind that Dijkstra's algorithm may produce a better path.

Lemma 3.5 *Given a vertex s and an arbitrary point t on $\mathcal{P}$, there exists a path $\Pi'(s,t)$ on $\mathcal{P}$ and a positive $\epsilon < \frac{1}{2}$ such that*

$$\|\Pi'(s,t)\| \leq \left(1 + \left(\frac{3(1+\frac{W}{w})-2(3+\frac{W}{w})\epsilon}{(1-\epsilon)(1-2\epsilon)}\right)\epsilon\right)\|\Pi(s,t)\| \leq \left(1 + \frac{3+3\frac{W}{w}}{1-3\epsilon}\epsilon\right)\|\Pi(s,t)\|.$$

Proof: Using Lemma 3.4 and Property 1.16 and letting $f(\epsilon) = \left(2 + \frac{2W}{(1-2\epsilon)w}\right)\epsilon$ we have:

$$\begin{aligned}
\|\Pi'(s,t)\| &= \|\Pi'(s,p)\| + \|\overline{pt}\| \\
&\leq (1+f(\epsilon))\|\Pi(s,p)\| + W_{f_t}|\overline{pt}| \\
&\leq (1+f(\epsilon))(\|\Pi(s,x)\| + \|\overline{xp}\|) + W_{f_t}(|\overline{px}| + |\overline{xt}|) \\
&= \|\Pi(s,t)\| + f(\epsilon)\|\Pi(s,x)\| + (1+f(\epsilon))\|\overline{xp}\| + W_{f_t}|\overline{px}|
\end{aligned}$$

$$\leq \quad (1+f(\epsilon))\|\Pi(s,t)\| + (1+f(\epsilon))\|\overline{px}\| + W_{f_t}|\overline{px}| \tag{3.1}$$

If p is a Steiner point, then since t lies outside τ_s and the second last edge s_{k-1} of $\Pi(s,t)$ must cross a face adjacent to f_t. We can therefore use Lemma 3.1 to show that $|\overline{px}| \leq \epsilon|s_{k-1}| \leq \epsilon|\Pi(s,x)| \leq \epsilon|\Pi(s,t)|$. Using this result and Property 1.12 we have $\|\overline{px}\| \leq \epsilon\|s_{k-1}\| \leq \epsilon\|\Pi(s,t)\|$. Also, if we use the weight of W_{f_t} for $\|\overline{px}\|$ then we have $W_{f_t}|\overline{px}| \leq \epsilon W_{f_t}|\Pi(s,t)| \leq \epsilon\frac{W}{w}\|\Pi(s,t)\|$. Substitution into equation (3.1) yields:

$$\|\Pi'(s,t)\| \quad \leq \quad ((1+\epsilon)(1+f(\epsilon)) + \epsilon\frac{W}{w})\|\Pi(s,t)\|. \tag{3.2}$$

Now consider the case when p is a vertex of $\mathcal{P}$ (i.e., x lies inside a sphere C_v). Since x lies within C_v, then $\Pi(s,x)$ intersects C_v and so $|\Pi(s,x)| \geq |\Pi(s,C_v)|$. We can make use of Property 1.2 to show that $|\Pi(s,C_v)| \geq (1-\epsilon)h_v = \frac{(1-\epsilon)}{\epsilon}r_v \geq \frac{(1-\epsilon)}{\epsilon}|\overline{px}|$. Therefore $|\overline{px}| \leq \frac{\epsilon}{(1-\epsilon)}|\Pi(s,x)| \leq \frac{\epsilon}{(1-\epsilon)}|\Pi(s,t)|$. Making use of Property 1.12 and substituting into equation (3.1) as above results in:

$$\|\Pi'(s,t)\| \quad \leq \quad \left((1+\frac{\epsilon}{1-\epsilon})(1+f(\epsilon)) + \frac{\epsilon W}{(1-\epsilon)w}\right)\|\Pi(s,t)\|. \tag{3.3}$$

This bound in equation (3.3) is slightly worse than that of equation (3.2). Substituting $f(\epsilon)$ into equation (3.3) yields

$$\|\Pi'(s,t)\| \quad \leq \quad \left(1+\left(\frac{3(1+\frac{W}{w})-2(3+\frac{W}{w})\epsilon}{(1-\epsilon)(1-2\epsilon)}\right)\epsilon\right)\|\Pi(s,t)\|.$$

□

Corollary 3.3 *Given a vertex s and an arbitrary point t on $\mathcal{P}$, there exists a path $\pi'(s,t)$ on $\mathcal{P}$ and a positive $\epsilon < \frac{1}{2}$ such that $|\pi'(s,t)| \leq \left(1+\left(\frac{5-6\epsilon}{(1-\epsilon)(1-2\epsilon)}\right)\epsilon\right)|\pi(s,t)|$.*

Proof: The proof is similar to that of Lemma 3.5 where $\frac{W}{w} = 1$ and $f(\epsilon) = \left(\frac{3-2\epsilon}{1-2\epsilon}\right)\epsilon$ as obtained from Lemma 3.2.

□

3.3.2 Approximations Between Arbitrary Points

Consider now paths in which both s and t are arbitrary points on the surface of $\mathcal{P}$. Let f_s be the face containing s and let x be the point in which $\Pi(s,t)$ exits f_s (i.e., x lies on an edge e_x of f_s). Similarly define f_t, y and e_y for point t. Again, without loss of generality assume that x and y are not vertices of $\mathcal{P}$. If x (respectively y) lies within some sphere C_{v_x} (respectively C_{v_y}) around a vertex v_x (respectively v_y) of $\mathcal{P}$ then let $p = v_x$ (respectively $q = v_y$). Otherwise, let p (respectively q) be the Steiner point on e_x (respectively e_y) that is closest to s (respectively t). We analyze the path $\Pi'(s,t)$ which is constructed as the concatenation of $\{\overline{sp}, \Pi'(p,q), \overline{qt}\}$. To begin, we will analyze the accuracy of $\Pi'(p,q)$. This is done similarly to Lemma 3.4, but it must consider Steiner points that are close together (i.e., t lies inside τ_s). As will be seen, the bound is similar.

Lemma 3.6 *Let p and q be two vertices of G that do not lie on edges which are incident to the same vertex. There exists an approximated path $\Pi'(p,q)$ in G and a positive $\epsilon < \frac{1}{2}$ for which*

$$\|\Pi'(p,q)\| \leq \left(1 + \left(\frac{2(1+2\frac{W}{w})-6(1+\frac{W}{w})\epsilon+4\epsilon^2}{(1-\epsilon)(1-2\epsilon)}\right)\epsilon\right)\|\Pi(p,q)\|.$$

Proof: If at least one of p and q are vertices of $\mathcal{P}$, then we can use Lemma 3.4 to bound $\|\Pi'(p,q)\|$ with a better bound than given here. Assume therefore that p and q are both Steiner points. In addition, assume that $\Pi(p,q)$ passes through at least one sphere, otherwise all segments of $\Pi'(p,q)$ are either outside-sphere or overlapping-sphere segments and we can use Lemma 3.3 to bound them such that $\|\Pi'(p,q)\| \leq (1+2\epsilon)\|\Pi(p,q)\|$. A problem arises in the proof of Claim 3.11 and Lemma 3.4 in which we "charge" the cost of inside-sphere subpaths to their preceding between-sphere subpaths. More specifically, the "charging" problem occurs only for the first sphere C_{σ_1} that is intersected by $\Pi(p,q)$ since it is possible that $\|\Pi(p,\sigma_1)\| <$

$\|\Pi'(\sigma_1)\|$ (see Figure 3.10). Since p and q do not lie on edges incident to the same vertex, then Property 1.2 ensures that either $|\Pi(p, C_{\sigma_1})| \geq (1-\epsilon)h_{\sigma_1}$ or $|\Pi(q, C_{\sigma_1})| \geq (1-\epsilon)h_{\sigma_1}$. Also, since $\Pi(p,q)$ passes through C_{σ_1} by assumption, then $|\Pi(p,q)| \geq |\Pi(p, C_{\sigma_1})|$ and $|\Pi(p,q)| \geq |\Pi(q, C_{\sigma_1})|$. Thus $|\Pi(p,q)| \geq (1-\epsilon)h_{\sigma_1} \geq \frac{(1-\epsilon)}{2\epsilon}|\Pi'(\sigma_1)|$. This implies that $|\Pi'(\sigma_1)| \leq \frac{2\epsilon}{1-\epsilon}|\Pi(p,q)|$. Using a proof similar to that of Lemma 3.4, we bound all subpaths of $\Pi'(p,q)$ except $\Pi'(\sigma_1)$ within the bound of $(1+(2+\frac{2W}{(1-2\epsilon)w})\epsilon)\|\Pi(p,q)\|$. The additional cost of $\|\Pi'(\sigma_1)\|$ is at most $\frac{2\epsilon W}{(1-\epsilon)w}\|\Pi(p,q)\|$ and so $\|\Pi'(p,q)\| \leq \left(1+\left(\frac{2(1+2\frac{W}{w})-6(1+\frac{W}{w})\epsilon+4\epsilon^2}{(1-\epsilon)(1-2\epsilon)}\right)\epsilon\right)\|\Pi(p,q)\|$.

□

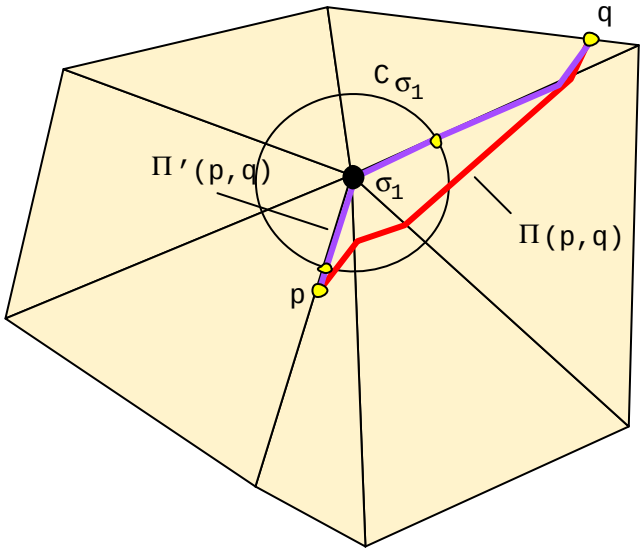

Figure 3.10: An example in which $\|\Pi(p,\sigma_1)\| < \|\Pi'(\sigma_1)\|$.

Corollary 3.4 *Let p and q be two vertices of G that do not lie on edges which are incident to the same vertex. There exists an approximated path $\pi'(p,q)$ in G and a positive $\epsilon < \frac{1}{2}$ for which $|\pi'(p,q)| \leq (1+(\frac{5-9\epsilon+2\epsilon^2}{(1-\epsilon)(1-2\epsilon)})\epsilon)|\pi(p,q)|$.*

Proof: The proof is similar to that of Lemma 3.6 where $\frac{W}{w} = 1$ and a proof similar to Lemma 3.2 is used in place of Lemma 3.4.

□

Theorem 3.2 *Given two points s and t on $\mathcal{P}$ such that t lies outside τ_s, there exists a path $\Pi'(s,t)$ on $\mathcal{P}$ and a positive $\epsilon < \frac{1}{2}$ such that*

$$\|\Pi'(s,t)\| \leq \left(1 + \left(\frac{4(1+\frac{W}{w}) - 2(3+\frac{W}{w})\epsilon - 4\epsilon^2 + \max(0, 2\frac{W}{w} - 2(1+\frac{W}{w})\epsilon - 8\frac{W}{w}\epsilon^2 + 8\epsilon^3)}{(1-\epsilon)(1-2\epsilon)}\right)\epsilon\right)\|\Pi(s,t)\|$$
$$\leq \left(1 + \epsilon\left(\frac{4+6\frac{W}{w}}{1-3\epsilon}\right)\right)\|\Pi(s,t)\|.$$

Proof: Since t lies outside τ_s, then p and q cannot lie on edges that are incident to the same vertex. First consider the case where neither p nor q are vertices of $\mathcal{P}$. We can make use of Lemma 3.6 to bound $\|\Pi'(s,t)\|$ as follows:

$$\begin{aligned}\|\Pi'(s,t)\| &= \|\overline{sp}\| + \|\Pi'(p,q)\| + \|\overline{qt}\| \\ &= \|\overline{sp}\| + (1+f(\epsilon))\|\Pi(p,q)\| + \|\overline{qt}\|\end{aligned}$$

where $f(\epsilon) = \left(\frac{2(1+2\frac{W}{w}) - 6(1+\frac{W}{w})\epsilon + 4\epsilon^2}{(1-\epsilon)(1-2\epsilon)}\right)\epsilon$. Recall the definitions of x and y (see beginning of this section). Using Property 1.16 we can express $\|\Pi'(s,t)\|$ as:

$$\begin{aligned}\|\Pi'(s,t)\| &\leq W_{f_s}(|\overline{sx}| + |\overline{xp}|) + (1+f(\epsilon))(\|\overline{px}\| + \|\Pi(x,y)\| + \|\overline{yq}\|) + W_{f_t}(|\overline{qy}| + |\overline{yt}|) \\ &= \|\overline{sx}\| + (1+f(\epsilon))(\|\overline{px}\| + \|\Pi(x,y)\| + \|\overline{qy}\|) + \|\overline{yt}\| + W_{f_s}|\overline{px}| + W_{f_t}|\overline{qy}| \\ &= \|\Pi(s,t)\| + f(\epsilon)\|\Pi(x,y)\| + (1+f(\epsilon))(\|\overline{px}\| + \|\overline{qy}\|) + W_{f_s}|\overline{px}| + W_{f_t}|\overline{qy}| \\ &\leq (1+f(\epsilon))\|\Pi(s,t)\| + (1+f(\epsilon))(\|\overline{px}\| + \|\overline{qy}\|) + W_{f_s}|\overline{px}| + W_{f_t}|\overline{qy}| \quad (3.4)\end{aligned}$$

We can now bound $\|\overline{px}\|$ (resp. $\|\overline{qy}\|$) with respect to the length of the second segment s_2 (resp. second last segment s_{k-1}) of $\Pi(s,t)$ (see Figure 3.11). This is accomplished by using Lemma 3.1 and Property 1.12 as was done in Lemma 3.5. As a result, we obtain $\|\overline{px}\| \leq \epsilon\|\Pi(s,t)\|$ and $\|\overline{qy}\| \leq \epsilon\|\Pi(s,t)\|$. In addition, $W_{f_s}|\overline{px}| \leq \epsilon\frac{W}{w}\|\Pi(s,t)\|$

and $W_{f_t}|\overline{qy}| \leq \epsilon\frac{W}{w}\|\Pi(s,t)\|$. Making these substitutions into equation (3.4) results in:

$$\begin{aligned}\|\Pi'(s,t)\| &\leq (1+f(\epsilon))\|\Pi(s,t)\| + 2(1+f(\epsilon))\epsilon\|\Pi(s,t)\| + \frac{2W\epsilon}{w}\|\Pi(s,t)\| \\ &= ((1+2\epsilon)(1+f(\epsilon)) + \frac{2W\epsilon}{w})\|\Pi(s,t)\| \\ &= (1+\left(\frac{(4+\frac{6W}{w}) - (8+\frac{4W}{w})\epsilon - (4+\frac{8W}{w})\epsilon^2 + 8\epsilon^3}{(1-\epsilon)(1-2\epsilon)}\right)\epsilon)\|\Pi(s,t)\| \quad (3.5)\end{aligned}$$

In the case where at least one of p or q is a vertex of $\mathcal{P}$, we can again make use of equation (3.4). However, we can now use the results of Lemma 3.4 and use $f(\epsilon) = (2+\frac{2W}{(1-2\epsilon)w})\epsilon$. Using a similar argument as in Lemma 3.5, we can easily show that $\|\overline{px}\| \leq \frac{\epsilon}{(1-\epsilon)}\|\Pi(s,t)\|$, $\|\overline{qy}\| \leq \frac{\epsilon}{(1-\epsilon)}\|\Pi(s,t)\|$, $W_{f_s}|\overline{px}| \leq \frac{\epsilon W}{(1-\epsilon)w}\|\Pi(s,t)\|$, and $W_{f_t}|\overline{qy}| \leq \frac{\epsilon W}{(1-\epsilon)w}\|\Pi(s,t)\|$. Making the substitution into 3.4 results in:

$$\begin{aligned}\|\Pi'(s,t)\| &\leq \left((1+\frac{2\epsilon}{(1-\epsilon)})(1+f(\epsilon)) + \frac{2W\epsilon}{(1-\epsilon)w}\right)\|\Pi(s,t)\| \\ &= \left(1+\left(\frac{4(1+\frac{W}{w}) - 2(3+\frac{W}{w})\epsilon - 4\epsilon^2}{(1-\epsilon)(1-2\epsilon)}\right)\epsilon\right)\|\Pi(s,t)\| \quad (3.6)\end{aligned}$$

By extracting the common terms from equations (3.5) and (3.6), we obtain the bound stated in the Lemma.

□

Corollary 3.5 *Given two points s and t on $\mathcal{P}$ such that t lies outside τ_s, there exists a path $\pi'(s,t)$ on $\mathcal{P}$ and a positive $\epsilon < \frac{1}{2}$ such that*

$$|\pi'(s,t)| \leq \left(1+\left(\frac{7-7\epsilon-2\epsilon^2+\max(0,2-4\epsilon-6\epsilon^2+4\epsilon^3)}{(1-\epsilon)(1-2\epsilon)}\right)\epsilon\right)|\pi(s,t)| \leq \left(1+\frac{9\epsilon}{1-3\epsilon}\right)|\pi(s,t)|.$$

Proof: The proof is similar to that of Theorem 3.2 where $\frac{W}{w} = 1$. When neither p nor q are vertices of $\mathcal{P}$, then $f(\epsilon) = \left(\frac{5-9\epsilon+2\epsilon^2}{(1-\epsilon)(1-2\epsilon)}\right)\epsilon$ is obtained from Corollary 3.4 instead of from Lemma 3.6 and the bound is $\|\Pi'(s,t)\| \leq \left(1+\left(\frac{9-11\epsilon-8\epsilon^2+4\epsilon^3}{(1-\epsilon)(1-2\epsilon)}\right)\epsilon\right)\|\Pi(s,t)\|$.

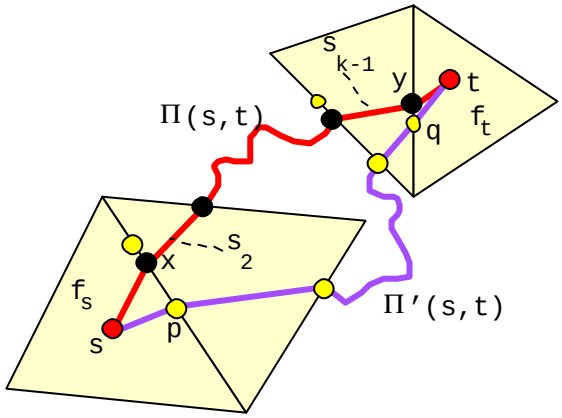

Figure 3.11: The second and second last segments of a path $\Pi(s,t)$.

When one or both of p and q are vertices of $\mathcal{P}$, then $f(\epsilon) = \left(\frac{3-2\epsilon}{1-2\epsilon}\right)\epsilon$ is obtained from Lemma 3.2 and the bound is $\|\Pi'(s,t)\| \leq \left(1 + \left(\frac{7-7\epsilon-2\epsilon^2}{(1-\epsilon)(1-2\epsilon)}\right)\epsilon\right)\|\Pi(s,t)\|$. By extracting the common terms from these two bounds we obtain the stated bound.

□

Chapter 4

Approximating Minimal Energy Paths

Euclidean shortest paths on terrain surfaces may not generally provide optimal time or energy paths since they ignore the terrain attributes. Weighted shortest paths may produce shortest energy paths since they can incorporate some terrain attributes such as variable costs for different regions, face slopes and/or frictional coefficients. However, these different weight costs are constant for each region (face) and therefore they ignore the issue of travel direction. The direction of travel plays a large role in determining the physical effects incurred on a vehicle (i.e., car, truck, robot etc.) traveling along a terrain surface. It is for this reason that we investigate shortest *anisotropic* paths (i.e., paths that take into account the direction of travel as well as length and weight). Through anisotropism, we can also identify certain directions of travel that represent inclines that are too steep to ascend or unsafe to travel due to possible dangers such as overturning, sliding or wheel slippage. We address the problem of determining a path for a vehicle between two points, s and t, on a terrain $\mathcal{P}$ such that the path minimizes energy consumption and does not travel on "dangerous"

slopes.

The chapter is organized as follows. In Section 4.1 we introduce the physical model that is used to determine the cost of an anisotropic path. Section 4.2 then presents some of the properties of shortest anisotropic paths such as path characteristics and bounds on the number of segments. In Section 4.3 we then explain how the anisotropic paths can be approximated in a straight forward and practical manner and then give bounds on the approximation produced. Section 4.4 then describes an ϵ-approximation algorithm to solve the same problem but with a better theoretical worst-case bound. Lastly, in Section 4.5 we present some experimental results from our tests using an implementation of our more practical algorithm of Section 4.3.

4.1 The Physical Model

We have chosen the model used by Rowe and Ross [108] for solving our minimal energy path problem. The model allows two main forces to act against the propulsion of the vehicle, namely friction and gravity. It is assumed that the vehicle has no net acceleration over the path from s to t and that the cost of turning is insignificant with respect to energy loss. Also, the model ignores the cost and feasibility of making turns.

We assume that the terrain surface $\mathcal{P}$ is composed of n triangular faces, each face $f_j, 1 \leq j \leq n$ having a homogeneous weight (cost) μ_j pertaining to the coefficient of kinetic friction (between 0 and 1) for that face with respect to the moving vehicle. Examples of some coefficients are shown in Table 4.1. We will assume in our model that only one coefficient is assigned to each face. The rubber on concrete, for example, may be used to represent vehicle movement on roadways. In some scenarios though, it may be desirable to have more than one coefficient of friction assigned to each face.

For example, modeling a person climbing up a hill (face) may require a coefficient similar to "rubber shoes on snow" and that person coming down the hill may use a snowboard which would require a different coefficient such as "waxed wood on dry snow".

Scenario	μ_j
rubber on concrete	0.8
wood on wood	0.2
waxed wood on dry snow	0.04
ice on ice	0.03

Table 4.1: Some known coefficients of friction.

4.1.1 Basic Model (Weight Metric)

Let mg be the weight of the vehicle where m is the mass of the vehicle and g is the coefficient of gravity. Consider a segment s_i of a shortest path which crosses a face f_j of $\mathcal{P}$. Let ϕ_j be the inclination angle (gradient) of f_j and let φ_i be the inclination angle of s_i with respect to the XY plane (see Figure 4.1). Our model assumes that the terrain has no vertical faces and so it is always the case that $-90 < \phi_j, \varphi_i < 90$. Our model assumes that the cost of travel for s_i is:

$$mg(\mu_j \cos\phi_j + \sin\varphi_i) \cdot |s_i| \tag{4.1}$$

We assume that mg is constant for the problem instance and hence we will ignore it during the computation of the path and multiply the path cost by this factor after it has been computed. The cost due to the force of friction is represented by $\mu_j \cos\phi_j \cdot |s_i|$. Rowe and Ross [108] reason about paths in their 2-D azimuth projections on the plane of the terrain. They ignore the $\cos\phi_j$ factor, stating that it is very close to 1 for most traversable natural terrain. We keep this value since it is easily computed and can be

combined with the μ_j factor to get an overall face weight. Therefore, it is convenient to define $w_j = \mu_j \cos\phi_j$ to be the weight of face f_j. Let W (respectively w) be the maximum (respectively minimum) of all $w_j, 1 \leq j \leq n$.

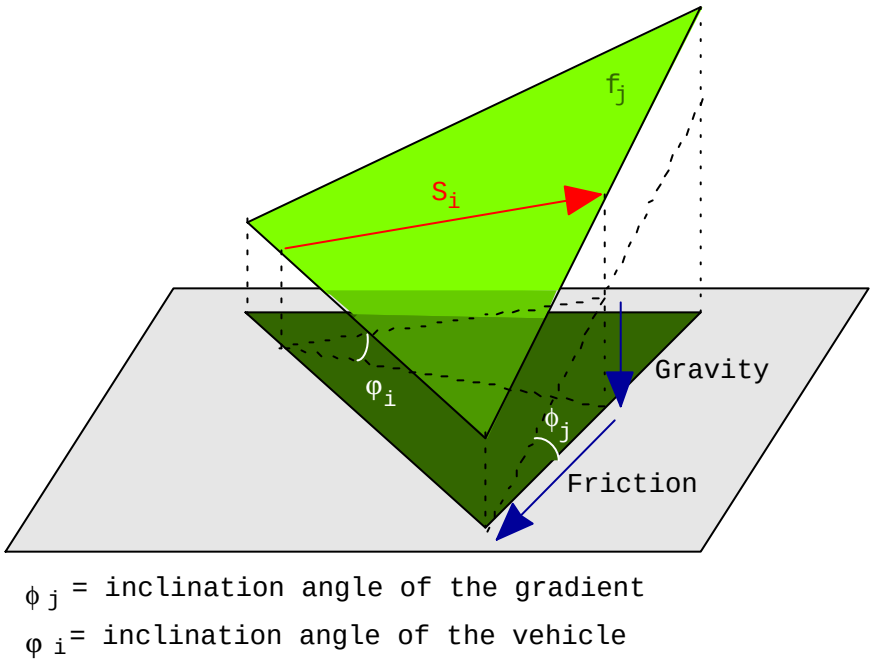

Figure 4.1: The forces of friction and gravity that act against the propulsion of the vehicle.

The cost due to gravity is represented by $|s_i| \sin \varphi_i$ which is the change in elevation of the segment s_i. Hence the work expended against gravity during path traversal is merely the sum of the elevation changes of all the path segments. This sum represents the difference in height between s and t which is independent of the path taken. As a result, we can leave this portion out when computing a shortest cost path and add it after the path has been computed.

Note that this model ignores internal energy losses (such as heat loss in an engine), friction of wheels on axles and wind resistance, and costs of turning. Rowe and Ross

[108] claim that internal energy losses are proportional to external work done against friction and gravity and can be handled by an appropriate multiplication factor. Also, the wheel friction and wind resistance can be modeled proportional to the path distance.

Consequently, the "uphill" anisotropic cost of travel (i.e., when the cost formula of Equation (4.1) is positive) for a segment s_i through a face f_j is $w_j|s_i|$. Notice that this cost is the same as that of the weighted isotropic shortest path cost that was used in our previous work described in Chapters 2 and 3. We will now consider the effects of braking phenomenon and invalid directions (obstacles).

4.1.2 Model With Braking

For inclination angles $\varphi_i < -\arcsin(\mu_j \cos\phi_j)$, the cost formula of Equation (4.1) becomes negative, representing energy gain while going downhill. This however, violates our earlier assumption that there is no net acceleration. We denote the inclination angle $\varphi_i = -\arcsin(\mu_j \cos\phi_j)$ in which this sign change occurs as a *critical braking angle*. To compensate, the cost formula is adjusted so that the energy gained going downhill is exactly compensated by the energy required to brake. So, the vehicle neither accelerates nor does it gain or lose energy when traveling in a braking range. Furthermore, it will be assumed that braking requires negligible energy. The compensation is made possible by replacing the $\mu_j \cos\phi_j$ friction factor by $-\sin\varphi_i$ which cancels out the gravity force resulting in zero cost travel. Notice that although the cost for downhill braking is zero, the negative gravity force has already been extracted from the metric, leaving a cost of $-\sin\varphi_i \cdot |s_i|$ for segment s_i. This cost will always be positive and non-zero for the valid range of downhill angles: $-90 < \varphi_i < 0$ degrees. Such a segment is called a *braking* segment and it is always assigned a positive, non-zero weight.

Property 4.1 *For a braking segment s_i passing through face f_j, $-\sin\varphi_i \geq w_j$.*

Proof: For braking segments, $\varphi_i < -\arcsin(\mu_j \cos\phi_j)$. This can be rewritten as $\varphi_i < -\arcsin(w_j)$. Hence $-\sin\varphi_i \geq w_j$.

□

4.1.3 Model With Anisotropic Obstacles

One last characteristic of the model is that it will prevent travel in "unsafe" directions. Rula and Nuttall [109] indicate that four conditions can give anisotropic slope limitations:

1. Limited expendable force to counteract friction and gravity effects. Here the traveling inclination angle φ_i cannot exceed $\arcsin(\frac{F_{max}}{mg\sqrt{\mu_j^2+1}}) - \arctan(\mu_j)$ where F_{max} is the maximum force that the vehicle can exert.

2. Speed or power limitations of the vehicle. Here the traveling inclination angle φ_i cannot exceed $\arcsin(\frac{P_{max}}{v_{min}mg\sqrt{\mu_j^2+1}}) - \arctan(\mu_j)$ where P_{max} is the maximum power that the vehicle can exert and v_{min} is the minimum (non-zero) speed of the vehicle.

3. Loss of traction danger (slippage). This occurs when the traveling inclination angle φ_i exceeds approximately $\arctan(\mu'_j - \mu_j)$, where μ'_j is the coefficient of static friction of face f_j.

4. Sideslope overturn danger. Here the projection of the center of gravity falls outside the polygon formed by its support points (most often perpendicular to "uphill"). The formula here depends on the vehicle shape (see Figure 4.2).

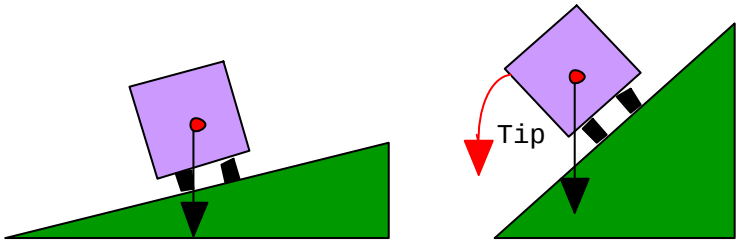

Figure 4.2: The sideslope overturn problem. The vehicle tips as its center of gravity projection falls outside the support polygon defined by the wheels.

The formulae above give rise to possibly three ranges of angles that define directions on a face that are impermissible for traveling. These are denoted as *impermissible ranges*. Together, with the braking range, there are up to four important angular ranges per face as shown in Figure 4.3. The boundary angles of the impermissible ranges are called *critical impermissibility angles*. The boundary angles of the braking range are called *critical braking angles*. For the regular angular ranges (i.e., not impermissible or braking), the range is bounded by critical impermissibility or braking angles. These angles will be called the *critical regular angles* for that regular range. Note that in the case of very steep angles, an impermissible range may arise within the braking range due to dangers of wheel slippage. That is, some ranges of downward traversal may be too steep for safe travel. We will ignore this impermissible range, since the face itself should be deemed unsafe of travel anyway. A path is said to be *valid* if and only if it does not travel in any impermissible directions.

If any of the impermissible ranges overlap, they are combined to make a single impermissible range. In some cases, the impermissible ranges may cover all possible angles. This represents an *isotropic obstacle* which is essentially a face that cannot be traveled on safely. If there exists an isotropic obstacle on $\mathcal{P}$ then it is possible that a

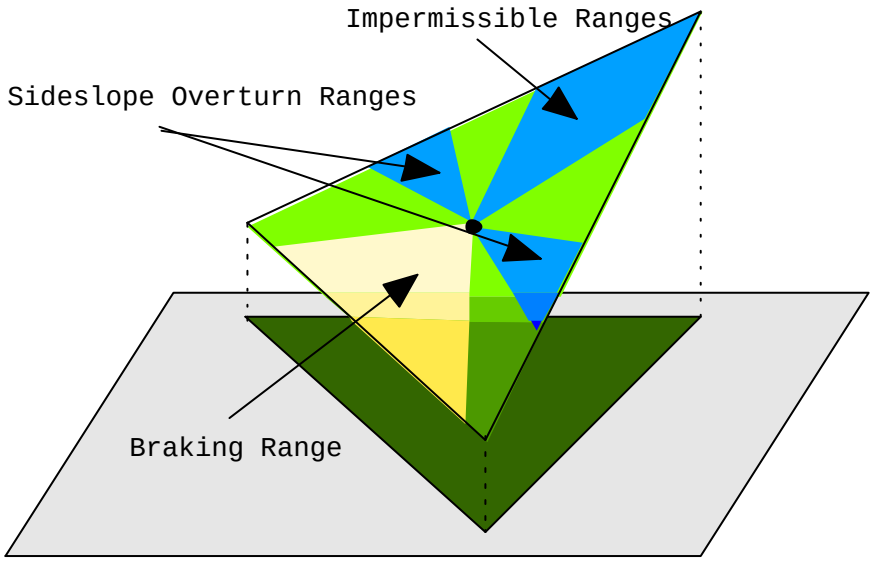

Figure 4.3: The up to three ranges representing impermissible travel and the braking range with respect to the center point of a single face.

valid path between two arbitrary points on $\mathcal{P}$ does not exist. For example, consider the example of Figure 4.4. It shows that there may be no path from s to t due to the steep faces (i.e., isotropic obstacles) that surround the plateau containing t.

Property 4.2 *Given two points s and t on $\mathcal{P}$, there may not be a valid path $\Pi(s,t)$ between them.*

The algorithm description and analysis presented here will assume that there exists at least one valid path, $\Pi(s,t)$ between s and t. Although we make this assumption in the analysis, our algorithm is able to detect the absence of valid paths and report when such a path does not exist. All that remains to be shown is that if a valid path does indeed exist, then our algorithm will always produce a valid path as well. This will be proven in Section 4.3.2.

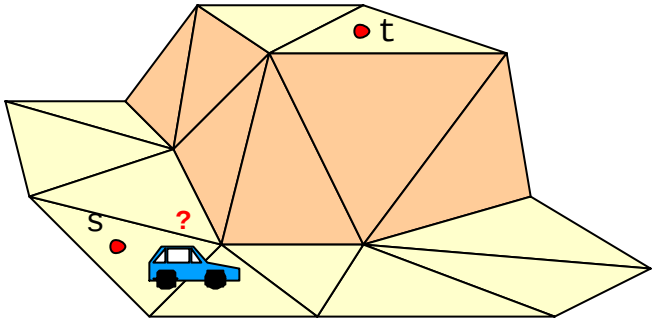

Figure 4.4: An example showing that there may not be a valid path between two points on $\mathcal{P}$ due to isotropic obstacles (shown as steep slopes surrounding the plateau containing t).

Let φ_c be a critical impermissibility angle for one of the critical impermissibility ranges of a face $f_j, 1 < j < n$ of $\mathcal{P}$. Let $\vec{u}$ and $\vec{v}$ be the two unit vectors representing the directions on the boundaries of the range (these are called a *matched pair* of critical angles). Thus, the angle that $\vec{u}$ and $\vec{v}$ make with the horizontal plane is φ_c. Let α_c be the angle between these two vectors when placed end-to-end as shown in Figure 4.5. Let α_j be the minimum of all α_c for the (up to three) impermissible ranges of f_j. Define $\beta_j, 1 < j < n$ to be the angle on f_j between the matched pair of braking angles and $\lambda_j, 1 < j < n$ to be the minimum angle on f_j between a matched pair of regular angles on f_j. Furthermore, for $1 < j < n$ let $\alpha = \min(\alpha_j)$, $\beta = \min(\beta_j)$, and $\lambda = \min(\beta_j, \lambda_j)$.

4.2 Path Types and Properties

On flat terrain, our model behaves like a weighted shortest path as in the weighted region problem of Mitchell and Papadimitriou [90] since there are no braking or impermissible ranges. That is, the path bends at edges of the terrain, obeying Snell's

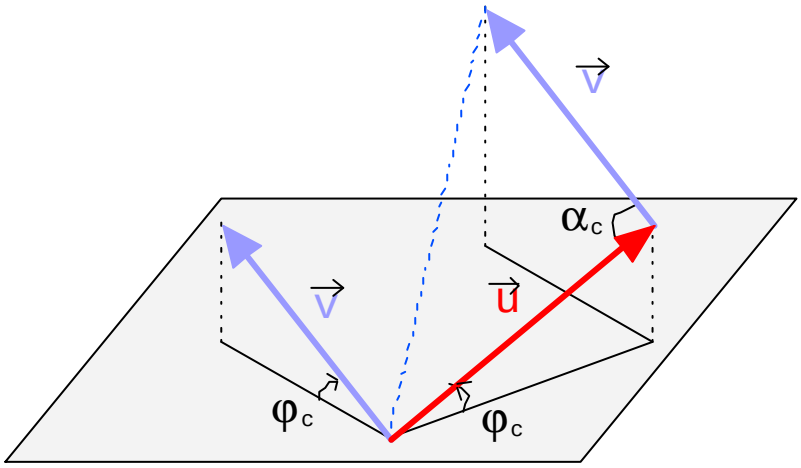

Figure 4.5: The value of α_c with respect to critical angle vectors $\vec{u}$ and $\vec{v}$.

law of refraction. For non-flat terrains with steep slopes, the path characteristics are different. Rowe and Ross [108] show that a shortest anisotropic path can traverse a face in three different ways as shown in Figure 4.6. This corresponds to one of three types of face crossings which we denote as *direction types*:

1. *Regular*: Straight across at a permissible, non-braking heading obeying Snell's law along the face boundaries (may travel along a critical angle).

2. *Switchback*: Given a matched pair of critical impermissibility direction vectors $\vec{u}$ and $\vec{v}$, this path consists of consecutive straight line segments that alternate in directions $\vec{u}$ and $\vec{v}$. Each change in direction is called a *switchback*.

3. *Braking*: Straight across with braking at a non-critical heading.

The algorithm of Rowe and Ross [108] computes a shortest energy path for a vehicle moving in a terrain. Their algorithm computes all possible combinations of

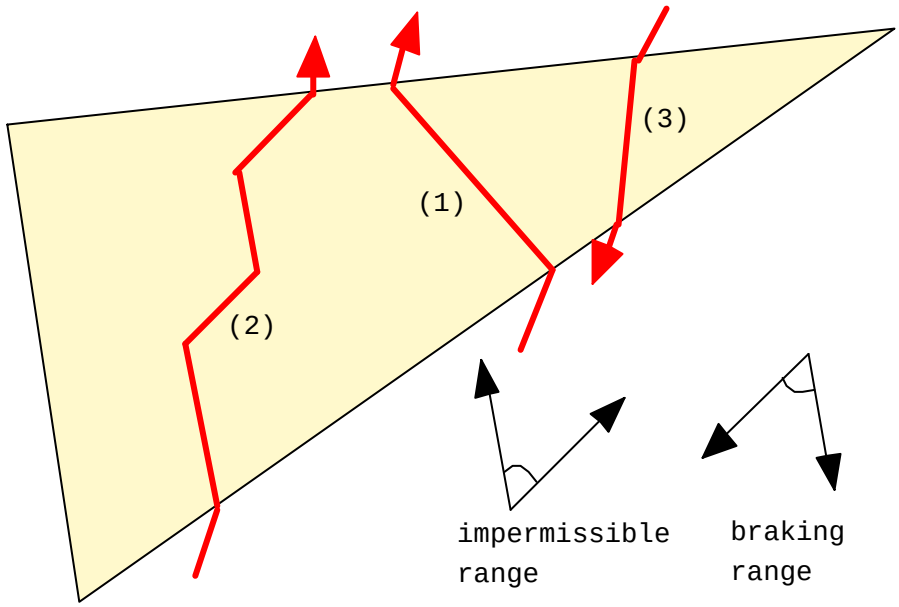

Figure 4.6: The 4 ways in which a shortest anisotropic path can cross a face.

the three traversal types above at each edge encountered during propagation. They exhaust all possibilities through an A^* search (see [56]) resulting in an $O(n^n)$ algorithm in the worst case. They do however, use a set of heuristics and pruning techniques to help alleviate this large time complexity, stating that it performs much better than this pessimistic bound. In their implementation, switchback paths were disallowed.

Let us examine a switchback path that passes through a face f_j. Rowe and Ross [108] show that these switchback paths are contained within f_j. Furthermore, they show that a switchback path consists of a chain of consecutive segments directed in the directions of a matched pair of critical impermissibility angle directions. This matched pair of angle directions, say $\vec{u}$ and $\vec{v}$, correspond to the boundaries of a single impermissibility range.

Property 4.3 *A switchback path between a point x of $\mathcal{P}$ and a vertex v of $\mathcal{P}$ may have an infinite number of segments.*

To understand this property, consider a switchback path from an edge of a face f_j to the vertex v opposite the edge as shown in Figure 4.7. Due to the constraint that the switchback path must lie within f_j, the number of switchbacks required to reach v is infinite. However, as we will see in Claim 4.1, the length of this path is bounded. The problem can only occur when at least one of the switchback path endpoints is a vertex of $\mathcal{P}$. To see this, consider for example a switchback path to v. The path is constrained to remain within the face. As the path gets closer to v, the convergence of the edges incident to v cause the path to get smaller and smaller. Since a direct path to v is not permissible, this convergence will cause the path segments to become infinitesimal in length as they approach v. A switchback path to a non-vertex point x on an edge of f_j does not encounter this problem since there always exists a finite length segment in a critical angle direction that reaches x that is completely contained within f_j.

In our algorithms, we will treat switchback paths as a single segment and assign a weight to it which incorporates the length of the switchback path itself. This allows us to compute paths with a finite number of segments. When reporting the path, in place of this segment we can report the actual switchback path. However, since there are an infinite number of segments in such a switchback path, we must "stop" reporting at some time. We can report the path up to the segment that intersects C_v and then finish off the reported path with a final segment from C_v to v. This final segment will be in an impermissible direction, but it can be made arbitrarily small in length. Note however, that the actual path cost is bounded as we will now show.

Let $\vec{u}$ and $\vec{v}$ be the critical angles that define the directions of the switchback path through a face f_j. The following properties bound the length of this switchback path

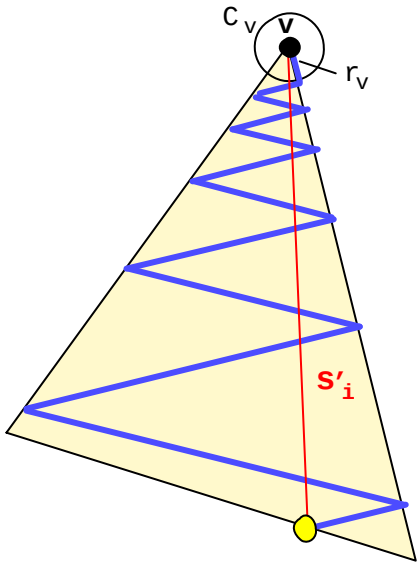

Figure 4.7: A switchback path to a vertex can have an infinite number of segments.

between two given points a and b lying in the plane defined by f_j:

Property 4.4 *There exist real scalars C_1 and C_2 such that $b = a + C_1\vec{u} + C_2\vec{v}$.*

Property 4.5 *There exist real scalars C_1 and C_2 such that $C_1|\vec{u}| + C_2|\vec{v}| \leq \frac{|\overline{ab}|}{\sin(\alpha_j/2)}$.*

Proof: Let c be the point at which $(a + C_1\vec{u})$ and $(b - C_2\vec{v})$ meet. We are sure that such a point exists because of Property 4.4. Thus, a, b and c form a triangle $\triangle acb$, as shown in Figure 4.8. Let $\omega = \angle cba, \psi = \angle bac$ and $\alpha_j = \angle acb$. The sine law ensures that $\sin\omega = \frac{C_1|\vec{u}|\sin\alpha_j}{|\overline{ab}|}$ and that $\sin\psi = \frac{C_2|\vec{v}|\sin\alpha_j}{|\overline{ab}|}$. Thus,

$$C_1|\vec{u}| + C_2|\vec{v}| = \left(\frac{\sin\omega + \sin\psi}{\sin\alpha_j}\right)|\overline{ab}| \tag{4.2}$$

If $\omega \geq \psi$ then $\sin\frac{\omega}{2} \geq \sin\frac{\psi}{2}$ and $\cos\frac{\omega}{2} - \cos\frac{\psi}{2} \leq 0$. Similarly, if $\omega \leq \psi$ then $\sin\frac{\omega}{2} \leq \sin\frac{\psi}{2}$ and $\cos\frac{\omega}{2} - \cos\frac{\psi}{2} \geq 0$. In either case,

$$\sin\frac{\omega}{2}\left(\cos\frac{\omega}{2} - \cos\frac{\psi}{2}\right) \leq \sin\frac{\psi}{2}\left(\cos\frac{\omega}{2} - \cos\frac{\psi}{2}\right)$$

and so

$$\begin{aligned}\sin\frac{\omega}{2}\cos\frac{\omega}{2} + \sin\frac{\psi}{2}\cos\frac{\psi}{2} &\leq \sin\frac{\omega}{2}\cos\frac{\psi}{2} + \sin\frac{\psi}{2}\cos\frac{\omega}{2} \\ \frac{\sin\omega}{2} + \frac{\sin\psi}{2} &\leq \sin\left(\frac{\omega+\psi}{2}\right).\end{aligned}$$

This can be simplified by substituting $\omega + \psi = \pi - \alpha_j$ as follows:

$$\begin{aligned}\sin\omega + \sin\psi &\leq 2\sin\left(\frac{\pi}{2} - \frac{\alpha_j}{2}\right) \\ &\leq 2\cos\frac{\alpha_j}{2} \\ &\leq \frac{2\sin\alpha_j\cos\frac{\alpha_j}{2}}{\sin\alpha_j} \\ &\leq \frac{\sin\alpha_j}{\sin\frac{\alpha_j}{2}}.\end{aligned}$$

Hence,

$$\frac{\sin\omega + \sin\psi}{\sin\alpha_j} \leq \frac{1}{\sin\frac{\alpha_j}{2}}.$$

□

Claim 4.1 *A switchback path between two points a and b on a face f_j which uses directions defined by $\vec{u}$ and $\vec{v}$ has length at most $\frac{|\overline{ab}|}{\sin(\alpha_j/2)}$.*

Proof: Given two points a and b on a plane and two non-parallel vectors $\vec{u}$ and $\vec{v}$, it is easy to see that all paths that join a and b and are composed solely of segments that are directed in directions $\vec{u}$ and $\vec{v}$ have the same length. Since switchback paths for a given impermissibility range travel in at most two non-parallel directions, all switchback paths for a given impermissibility range between a and b have equal length. From Property 4.5, this length is at most $\frac{|\overline{ab}|}{\sin(\alpha_j/2)}|\overline{ab}|$.

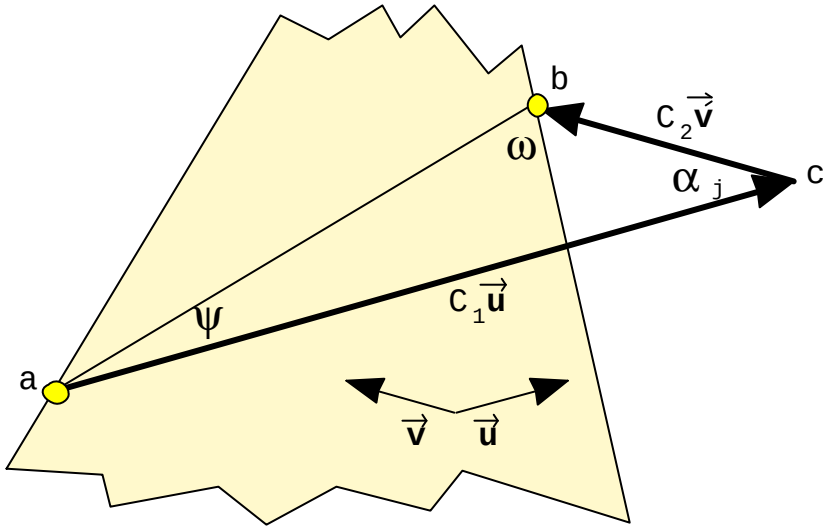

Figure 4.8: The triangle formed by extending $\vec{u}$ and $-\vec{v}$ from points a and b, respectively.

□

Claim 4.2 *A shortest anisotropic path $\Pi(s,t)$ is a simple path.*

Proof: Consider an edge $\overline{ab}$ of $\Pi(s,t)$ that crosses a face f_j of $\mathcal{P}$. Assume that some other edge $\overline{cd}$ of $\Pi(s,t)$ crosses (i.e., intersects) $\overline{ab}$ at some point x on f_j (e.g., see Figure 4.9). Clearly, the subsegment $\overline{ax}$ of $\overline{ab}$ is shorter than $\overline{ab}$ and so it would be cheaper to remain at x instead of traveling from x through b along $\Pi(b,c)$ then along $\overline{cx}$. Hence, $\overline{ax}$ and $\overline{cx}$ cannot be on $\Pi(s,t)$ because it would cause a contradiction of $\Pi(s,t)$ being a shortest path. Note that $\vec{cd}$ is a valid direction and so $\vec{xd}$ is as well. □

Note that Claim 4.2 is true only because there is no turning constraint at x. That is, our model does not consider the cost or feasibility of making turns. If for example, $\angle axd$ represented a turn that was too sharp, it may not be permissible to turn at x

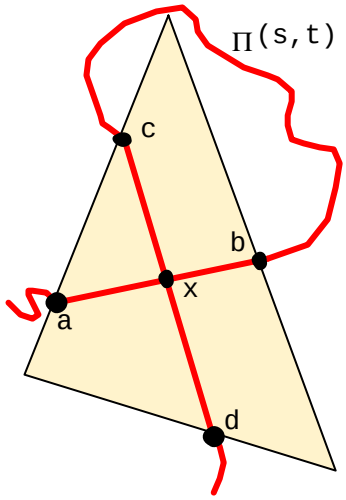

Figure 4.9: The impossible situation in which an edge $\overline{ab}$ of $\Pi(s,t)$ supposedly intersects another edge $\overline{cd}$ of $\Pi(s,t)$.

and perhaps the path through b and c would be a shortest permissible path from a to d.

We will now investigate the number of segments of $\Pi(s,t)$. We will consider a switchback path of $\Pi(s,t)$ to be a single segment. As in the previous chapter, a segment that intersects an edge of $\mathcal{P}$ within some distance r_v of any vertex v is called an *inside-sphere* segment. From here on, we will assume that k is finite and each switchback path is considered to be a single segment. Still, the task of bounding k is non-trivial. Instead, we will give a bound for k^*, which we denote as the number of non inside-sphere segments of $\Pi(s,t)$. The following claim bounds k^*:

Claim 4.3 *If $\Pi(s,t)$ does not have any inside-sphere segments then it may cross an edge of $\mathcal{P}$ at most $O(\log_{\mathcal{F}} \frac{|L|}{r})$ times, where $\mathcal{F} = (1 + \min(w, \sin\frac{\alpha}{2})\sin\theta)$.*

Proof: Consider a single permissible segment $s_i = \overline{ab}$ of $\Pi(s,t)$ that crosses through a face f of $\mathcal{P}$ where $1 < i < k$. If another segment , say $s_{i+j} = \overline{cd}$, of $\Pi(s,t)$ crosses f, then $\angle abc$ and $\angle abd$ must either both be left turns or both be right turns since Claim 4.2 does not permit s_i and s_{i+j} to cross. Without loss of generality, we will assume that $\angle abc$ and $\angle abd$ are left turns. Let x_i be the segment of minimum cost crossing f with one endpoint at a. Without loss of generality assume that x_i is a perpendicular and that it has minimal cost (i.e., smaller of braking or regular cost). Let e be the edge of f on which a lies. We would like to determine a point a' on e such that subsegment $e_a = \overline{aa'}$ of e has cost which is equal to $\|x_i\|$. Clearly s_{i+j} cannot cross e_a (i.e., point c cannot lie on e_a) since it would have been cheaper to travel from a to c along e_a than to travel along s_i then along $\Pi(b,c)$. We therefore must ensure

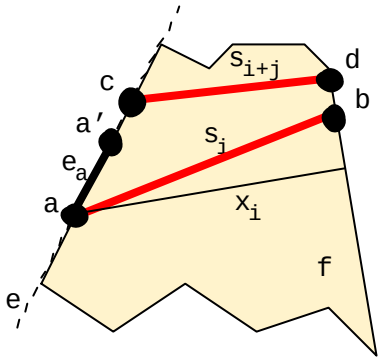

Figure 4.10: A subsegment e_a which can be crossed by only one segment of $\pi(s,t)$.

that $\|\overline{aa'}\| = \|x_i\|$. The determination of a' gives an indication as to how close to a another path segment may cross e. We then set $a = a'$ and repeat the procedure to obtain a new subsegment $e_{a'}$ along e with respect to a'. By continuing in this manner and ensuring that each such subsegment is minimized, we are essentially determining the maximum number of times that $\Pi(s,t)$ can cross edge e (and hence also face f with an additional factor of at most three: pairing of edges).

Due to the different costs that can arise because of the braking and impermissibility ranges we must consider each of the different direction types for x_i and $\overline{aa'}$. For each case we consider the minimum allowable cost of $\overline{aa'}$ in order to get an upper bound on the number of subsegments on e.

Let φ_{e_a} and φ_{x_i} be the inclination angles of $\overline{aa'}$ and x_i, respectively. Iterating through all nine cases in which $\overline{aa'}$ and x_i are regular, braking or switchback, it can easily be shown that the smallest possible value of $|\overline{aa'}|$ such that $|\overline{aa'}|$ is maximized is when $|\overline{aa'}| = F|x_i|$, where

$$F = \min(1, \frac{-\sin\varphi_{x_i}}{w_f}, \frac{1}{\sin\frac{\alpha}{2}}, \frac{w_f}{-\sin\varphi_{e_a}}, \frac{-\sin\varphi_{x_i}}{-\sin\varphi_{e_a}}, \frac{w_f}{-\sin\varphi_{e_a}\sin\frac{\alpha}{2}}, \sin\frac{\alpha}{2}, \frac{-\sin\varphi_{x_i}\sin\frac{\alpha}{2}}{w_f}, 1).$$

Note that Property 4.1 ensures that for all terms in the above expression corresponding to braking angles (i.e., when $-\sin\varphi_{e_a}$ or $-\sin\varphi_{x_i}$ are used) then both $-\sin\varphi_{e_a}$ and $-\sin\varphi_{x_i}$ are greater than w_f. With the additional fact that $1 \leq \frac{1}{\sin\frac{\alpha}{2}}$ then we can simplify the above expression to $F = \min(\frac{w_f}{-\sin\varphi_{e_a}}, \sin\frac{\alpha}{2})$. Therefore, $|\overline{aa'}| = \min(\frac{w_f}{-\sin\varphi_{e_a}}, \sin\frac{\alpha}{2})|x_i|$. We can simplify this further by allowing $|\overline{aa'}|$ to be smaller that what is stated here. Note that this change leads to a slightly worse bound on the number of times the edge may be crossed. The simplification is to replace $-\sin\varphi_{e_a}$ by its highest value of 1:

$$|\overline{aa'}| = \min(w_f, \sin\frac{\alpha}{2})|x_i| \tag{4.3}$$

In order to determine the maximum number of times e may be crossed by $\Pi(s,t)$, we will consider the shortest such subsegments e_a that can be formed along the edge e based on Equation (4.3). Let v be a vertex of e and let p_1 be the intersection point of C_v and e. Let $p_2, p_3, p_4, ..., p_\kappa$ be a set of points along e such that

$$|\overline{p_j p_{j-1}}| = (1 + \min(w_f, \sin\frac{\alpha}{2})\sin\theta_v)|\overline{p_{j-1}p_{j-2}}|$$

where $2 < j \leq \kappa$. The value of κ represents the limit on the number of points that we can place along e in this fashion. The maximum value of κ represents the situation

in which e is crossed the most by segments of $\Pi(s,t)$. With a little algebra, it can be shown that $\kappa = O(\log_{\mathcal{F}} \frac{|e|}{r})$, where $\mathcal{F} = (1 + \min(w_f, \sin\frac{\alpha}{2})\sin\theta_v)$. By substituting the worst case when $w_f = w$, $\theta_v = \theta$ and $|e| = |L|$ we obtain the stated bounds.

□

Lemma 4.1 *A shortest anisotropic path $\Pi(s,t)$ that does not have any inside-sphere segments can have at most $O(n \log_{\mathcal{F}} \frac{|L|}{r})$ segments, where $\mathcal{F} = (1+\min(w, \sin\frac{\alpha}{2})\sin\theta)$ and r defines the radius of the smallest sphere around a vertex of $\mathcal{P}$.*

Proof: Follows from Claim 4.3 applied to the n faces of $\mathcal{P}$.

□

4.3 A Simple Approach to Approximation

Our goal is to improve the large running time of Rowe and Ross [108] through approximation. Our first approach is similar to the *Fixed Scheme* of the Euclidean and weighted cost shortest path approximation techniques used in Chapter 2. Recall that this scheme adds m Steiner points along each edge of the terrain and then connects Steiner points of each face with a complete graph. The global graph is then searched using Dijkstra's algorithm to obtain the approximation. This approach cannot be directly applied to solve the anisotropic shortest path problem since it may produce edges that are impermissible for travel. The problem does not lie in the placement of Steiner points, but rather lies in the direction and weights of the edges connecting them. The simple approach here is to also build a graph; the main difference in this graph lies in the weights that are placed on the graph edges which are now based on direction.

4.3.1 Constructing the Graph

For each face f_j of $\mathcal{P}$, we build a graph G_j. We begin by placing m equally spaced Steiner points along each edge of f_j, for some positive integer m. The vertices of G_i are the vertices of f_j as well as these Steiner points. Connect p and q with two directed edges $\overrightarrow{pq}$ and $\overrightarrow{qp}$, where p and q are Steiner points on different edges of f_j. Connect adjacent Steiner points on the same edge of f_j by two oppositely directed edges as well. Lastly, connect each vertex v of f_j to all Steiner points on the edge of f_j opposite to v with two oppositely directed edges.

We now assign weights to the edges of G_j. For each edge e of G_j that is directed in a heading that is non-braking and non-impermissible, its weight is set to be $w_j|e|$. Edges whose direction falls within the braking range are given weight equal to $-\sin\varphi_i|e|$. This essentially assigns a weight on the edge equal to the change in elevation. Note again that φ_i is a negative angle in this case and the weight will therefore be positive. Edges representing impermissible headings are assigned a weight corresponding to the length of a switchback path between their endpoints. From Claim 4.1 we know that this length is at most $\frac{|e|}{\sin\frac{\alpha_j}{2}}$. However, this is an upper bound on the length. In fact, we can assign the exact length of the path as stated in the proof of Property 4.5. This length is $\left(\frac{\sin\omega+\sin\psi}{\sin\alpha_j}\right)|e|$, where ω and ψ are defined as shown in Figure 4.8. Hence the weight of e is set to $w_j\left(\frac{\sin\omega+\sin\psi}{\sin\alpha_j}\right)|e|$. To simplify the analysis, however, we will be bounding these switchback paths with the cost of $\frac{w_j|e|}{\sin\frac{\alpha_j}{2}}$. This concludes our construction of G_j. A global graph G is then constructed as the union of all $G_j, 1 \leq j \leq n$.

Claim 4.4 *Each edge $e \in G$ represents a path with bounded cost on $\mathcal{P}$.*

Proof: Regular and braking edges of G clearly have bounded cost since their lengths are finite and they correspond to a single-segment on $\mathcal{P}$. Claim 4.1 ensures that

switchback edges of G represent paths that have bounded length and therefore, bounded cost. Note that we are not saying anything about the number of segments in the switchback paths since Property 4.3 indicates this may be unbounded.

□

4.3.2 Choosing an Approximated Path for Analysis

Given a shortest anisotropic path $\Pi(s,t)$ on $\mathcal{P}$, we describe here a particular path $\Pi'(s,t)$ in G which has a bounded cost with respect to $\Pi(s,t)$. Some of the segments of $\Pi'(s,t)$ may be "flagged" as *switchback* indicating that when the approximated path is to be reported, then these segments are to be replaced by a switchback path between their endpoints. A distinction is therefore required between the path in G and the path that will be reported. The reported path will be denoted as $\Pi''(s,t)$. We denote the number of segments of $\Pi(s,t), \Pi'(s,t)$ and $\Pi''(s,t)$ as k, k' and k'', respectively.

We consider the case in which $\Pi'(s,t)$ has segments that pass through the same faces (in the same order) as those of $\Pi(s,t)$. We will consider this path in our analysis of the approximation factor, although it may be the case that the running of Dijkstra's algorithm produces a better path.

Consider a segment s_i of $\Pi(s,t)$ passing through face f_j, where $0 < i < k$. Let p and q be the vertices of G_j corresponding to Steiner points (or vertices) on edges of f_j that are closest (Euclidean distance) to the endpoints of s_i. If $p \neq q$ then let $s'_i = \overline{pq}$ be the edge of $\Pi'(s,t)$ that approximates s_i. If $p = q$ then let $s'_i = \emptyset$ (Note that this situation only occurs when s_i crosses a face close to a vertex or when s_i travels very briefly along an edge of $\mathcal{P}$ (see Figure 4.11)). Once this is done for each segment of $\Pi(s,t)$, we remove all of the $\emptyset$ edges. Note also that if two or more consecutive edges are collinear, then these edges are merged as one. The resulting sequence of edges is

$\Pi'(s,t)$. This construction ensures that $k' \leq k$.

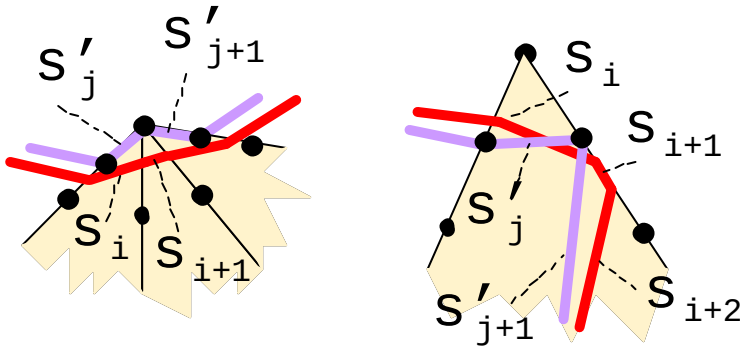

Figure 4.11: When a segment s_i of $\Pi(s,t)$ is approximated near a vertex or briefly along an edge of $\mathcal{P}$, there may be no corresponding segment in $\Pi'(s,t)$.

Claim 4.5 *$\Pi'(s,t)$ is connected.*

Proof: In order to show connectivity of $\Pi'(s,t)$ we need to show that adjacent edges s'_j and s'_{j+1} are connected, where $1 \leq j < k'$. Consider two consecutive edges s_i and s_{i+1} of $\Pi(s,t)$, where $1 \leq i < k$. Since $\Pi(s,t)$ is connected, then s_i and s_{i+1} share a point x on an edge (possible vertex) of $\mathcal{P}$. There are three cases to consider. The first case is when x lies between two Steiner points on an edge of $\mathcal{P}$. In this case, s_i and s_{i+1} are approximated by s'_j and s'_{j+1}, respectively, such that s'_j and s'_{j+1} share an endpoint. Therefore, s'_j and s'_{j+1} are connected. In the second case, x lies between a vertex v of $\mathcal{P}$ and a Steiner point on an edge of $\mathcal{P}$. In this case, s'_j and s'_{j+1} are chosen such that they share the endpoint v, and so they are connected. Recall that the entire subpath of $\Pi(s,t)$ which intersects a sphere of radius r_v around v is approximated by just two segments of $\Pi'(s,t)$ and these segments are connected (see left of Figure 4.11). The final case is when either s_i or s_{i+1} is an edge-using segment (without loss of generality, assume that s_{i+1} is edge-using). If s'_{j+1} is an edge-using segment then s'_j

and s'_{j+1} are connected since they share a Steiner point. If s'_{j+1} is not an edge-using segment then s_{i+1} is not approximated. Instead, s_{i+2} is approximated by s'_{j+1} and they also share a Steiner point (see right of Figure 4.11).

□

If we are merely interested in reporting the cost of the approximated path, then we only need to report $\|\Pi'(s,t)\|$. However, if the path itself is required, it cannot necessarily be reported in $O(k')$ time since switchback segments of $\Pi'(s,t)$ may represent a large chain of segments whose number of links may be infinite and which depends on geometric parameters such as α and θ. We will therefore construct a path $\Pi''(s,t)$ from $\Pi'(s,t)$ in which all of these *switchback* edges are expanded to a finite number of actual segments on $\mathcal{P}$. Thus, it is always the case that $k'' \geq k'$.

To begin, each regular and braking edge s'_i of $\Pi'(s,t)$ has an equivalent (i.e., identical) segment s''_i in $\Pi'(s,t)$. For each switchback path s'_i of $\Pi'(s,t)$, we will construct a finite-segment switchback path z''_i in $\Pi''(s,t)$. To form $\Pi''(s,t)$ we concatenate all s''_i and z''_i in the order from $i = 1$ to k'. Recall that $\|s'_i\|$ encapsulates the length of a two-link switchback path. We cannot merely assign this two-link path as $z''_i = [s''_j, s''_{j+1}]$ since it may not lie completely within a single face. We can however "adjust" this two-link path (by creating more links) such that the newly adjusted path lies within the face (see Figure 4.12). We will set z''_i to be this adjusted path. Claim 4.1 ensures that $\|z''_i\| = \|s'_i\|$.

This creation of z''_i, however, will cause a problem in the case where at least one of the endpoints of s'_i is a vertex, say v. The problem is as described in Property 4.3, in which z''_i has an infinite number of segments. To handle this problem, we merely limit z''_i to contain only one switchback segment that intersects C_v (i.e., the sphere around v). To do this, we can iteratively append consecutive switchback segments to z''_i until we get one that intersects C_v. z''_i is completed with a segment from the last of

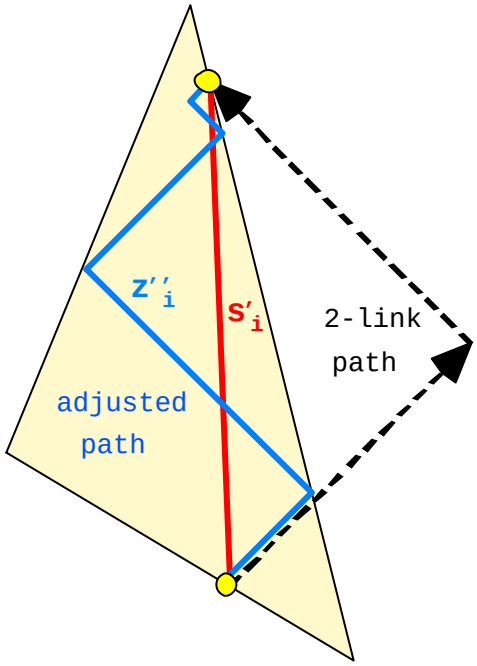

Figure 4.12: Adjusting a 2-link path such that it lies within the face.

these switchback segments to v. Although, this last link will be in an impermissible direction, we can make it arbitrarily small in length and so this should be feasible in practice.

A further problem is that of traveling on an edge in an impermissible direction. Let s'_i be a segment of our approximation that lies along an edge e of $\mathcal{P}$. If s'_i is directed in an impermissible direction, then it is clearly impossible to zig-zag along e. In this case, we construct z''_i as before, but such that the segments of z''_i all lie in one of the faces adjacent to e (i.e., the face with minimum path cost).

Claim 4.6 *If a valid path $\Pi(s,t)$ exists between two vertices s and t of $\mathcal{P}$ then there exists a path $\Pi'(s,t)$ in G and hence a valid path $\Pi''(s,t)$ on $\mathcal{P}$.*

Proof: Since $\Pi(s,t)$ is valid, then by definition, its segments must all be in permissible directions. Each segment s' of $\Pi'(s,t)$ corresponds to a segment s of $\Pi(s,t)$ that passes through the same face. By definition, the faces through which $\Pi(s,t)$ passes cannot be isotropic obstacles and so the faces through which $\Pi'(s,t)$ passes cannot be isotropic obstacles either. Therefore, since s and t are vertices of G, there exists a path in G. Since each edge of G represents either a valid edge or a switchback path of $\Pi''(s,t)$, then $\Pi''(s,t)$ must also exist.

□

4.3.3 Computing a Bound on the Approximated Path

In order to determine the accuracy of $\Pi'(s,t)$, we apply a similar segment-by-segment analysis as with the weighted scheme of Chapter 2. The analysis becomes slightly more complex due to the special cases of braking and switchbacks. That is, we must consider the situation in which a shortest path segment is approximated by a segment that has a different type (e.g., s_i is a braking segment and s'_i is a switchback path). Note that during the analysis, we do not need to consider the number of segments in the switchback paths, only the bound on the length of such paths. The bound of $\|\Pi'(s,t)\|$ will therefore apply to $\|\Pi''(s,t)\|$. Throughout the analysis, we will make use of Lemma 2.2 which states that in the Euclidean scenario,

$$|s'_i| \leq |s_i| + \frac{|L|}{m+1}. \tag{4.4}$$

where L is the longest edge of $\mathcal{P}$ and m is the number of Steiner points placed on each edge of $\mathcal{P}$.

Given a segment s_i (or switchback path z_i) passing through face f_j, we will bound the cost of s'_i. We will consider the nine possible cases which correspond to iterating through the three direction types for each of s_i and s'_i.

Claim 4.7 *If both s_i and s_i' are regular, then* $\|s_i'\| \leq \|s_i\| + \frac{w_j|L|}{m+1}$

Proof: Multiplying Equation (4.4) by w_j, we obtain $w_j|s_i'| \leq w_j|s_i| + w_j\frac{|L|}{m+1}$. Substituting the weighted cost for s_i and s_i', we realize that this is none other than $\|s_i'\| \leq \|s_i\| + \frac{w_j|L|}{m+1}$.

□

Claim 4.8 *If s_i is regular and s_i' is braking, then* $\|s_i'\| \leq -\sin\varphi_i' \left(\frac{\|s_i\|}{w_j} + \frac{|L|}{m+1}\right)$.

Proof: From our weight metric we obtain $\|s_i\| = w_j|s_i|$ and $\|s_i'\| = -\sin\varphi_i'|s_i'|$. Substituting Equation (4.4) for $|s_i'|$ results in:

$$\begin{aligned} \|s_i'\| &= -\sin\varphi_i'|s_i'| \\ &\leq -\sin\varphi_i'(|s_i| + \frac{|L|}{m+1}) \\ &= -\sin\varphi_i'(\frac{1}{w_j}\|s_i\| + \frac{|L|}{m+1}). \end{aligned}$$

□

Claim 4.9 *If s_i is regular and s_i' is switchback, then* $\|s_i'\| \leq \frac{1}{\sin\frac{\alpha_j}{2}} \left(\|s_i\| + \frac{w_j|L|}{m+1}\right)$.

Proof: From our weight metric we have $\|s_i'\| \leq \frac{w_j}{\sin\frac{\alpha_j}{2}}|s_i'|$. Substituting Equation (4.4) for $|s_i'|$ results in:

$$\begin{aligned} \|s_i'\| &\leq \frac{w_j}{\sin\frac{\alpha_j}{2}} \left(|s_i| + \frac{|L|}{m+1}\right) \\ &= \frac{w_j}{\sin\frac{\alpha_j}{2}} \left(\frac{\|s_i\|}{w_j} + \frac{|L|}{m+1}\right) \\ &= \frac{1}{\sin\frac{\alpha_j}{2}} \left(\|s_i\| + \frac{w_j|L|}{m+1}\right). \end{aligned}$$

□

Claim 4.10 *If s_i is braking and s_i' is regular, then $\|s_i'\| \leq \frac{w_j}{-\sin\varphi_i}\|s_i\| + \frac{w_j|L|}{m+1}$.*

Proof: From our weight metric we obtain $\|s_i'\| = w_j|s_i'|$ and $\|s_i\| = -\sin\varphi_i|s_i|$. Substituting the results of Equation (4.4) results in:

$$\begin{aligned}
\|s_i'\| &= w_j|s_i'| \\
&\leq w_j(|s_i| + \frac{|L|}{m+1}) \\
&= w_j(\frac{\|s_i\|}{-\sin\varphi_i} + \frac{|L|}{m+1}) \\
&= \frac{w_j}{-\sin\varphi_i}\|s_i\| + \frac{w_j|L|}{m+1}.
\end{aligned}$$

□

Claim 4.11 *If both s_i and s_i' are braking, then $\|s_i'\| \leq \|s_i\| + \frac{|L|}{m+1}$.*

Proof: From our weight metric we obtain $\|s_i'\| = -\sin\varphi_i'|s_i'|$ and $\|s_i\| = -\sin\varphi_i|s_i|$. Here $\|s_i'\|$ represents the change in height between its endpoints; similarly for $\|s_i\|$. The difference in height between $\|s_i'\|$ and $\|s_i\|$ can be at most $\frac{|L|}{m+1}$ (which would occur if the face was vertical). Therefore, $\|s_i'\| \leq \|s_i\| + \frac{|L|}{m+1}$.

□

Claim 4.12 *If s_i is braking and s_i' is switchback, then $\|s_i'\| \leq \frac{w_j}{\sin\frac{\alpha_j}{2}}\left(\frac{\|s_i\|}{-\sin\varphi_i} + \frac{|L|}{m+1}\right)$.*

Proof: From our weight metric we obtain $\|s_i\| = -\sin\varphi_i|s_i|$ and $\|s_i'\| \leq \frac{w_j}{\sin\frac{\alpha_j}{2}}|s_i'|$. Substituting this into Equation (4.4) results in:

$$\begin{aligned}
\|s_i'\| &\leq \frac{w_j}{\sin\frac{\alpha_j}{2}}\left(|s_i| + \frac{|L|}{m+1}\right) \\
&= \frac{w_j}{\sin\frac{\alpha_j}{2}}\left(\frac{1}{-\sin\varphi_i}\|s_i\| + \frac{|L|}{m+1}\right).
\end{aligned}$$

□

Claim 4.13 *If s_i is switchback and s_i' is regular, then $\|s_i'\| \leq \|s_i\| + \frac{w_j|L|}{m+1}$.*

Proof: From our weight metric we are ensured that $\|s_i'\| = w_j|s_i'|$. Since s_i is switchback, then $\|s_i\| = w_j|z_i| > w_j|s_i|$. Using Equation (4.4) we obtain:

$$\begin{aligned} \|s_i'\| &= w_j|s_i'| \\ &\leq w_j(|s_i| + \frac{|L|}{m+1}) \\ &\leq w_j\left(\frac{\|s_i\|}{w_j} + \frac{|L|}{m+1}\right) \\ &= \|s_i\| + \frac{w_j|L|}{m+1}. \end{aligned}$$

□

Claim 4.14 *If s_i is switchback and s_i' is braking, then $\|s_i'\| \leq -\sin\varphi_i' \left(\frac{\|s_i\|}{w_j} + \frac{|L|}{m+1}\right)$.*

Proof: From our weight metric we have $\|s_i'\| = -\sin\varphi_i'|s_i'|$. Since s_i is switchback, then $\|s_i\| = w_j|z_i| > w_j|s_i|$. Using Equation (4.4) we obtain:

$$\begin{aligned} \|s_i'\| &= -\sin\varphi_i'|s_i'| \\ &\leq -\sin\varphi_i'(|s_i| + \frac{|L|}{m+1}) \\ &\leq -\sin\varphi_i'\left(\frac{\|s_i\|}{w_j} + \frac{|L|}{m+1}\right). \end{aligned}$$

□

Claim 4.15 *If both s_i and s_i' are switchback, then $\|s_i'\| \leq \frac{1}{\sin\frac{\alpha_j}{2}}\left(\|s_i\| + \frac{w_j|L|}{(m+1)}\right)$.*

Proof: From our weight metric we have $\|s_i'\| \leq \frac{w_j}{\sin\frac{\alpha_j}{2}}|s_i'|$. Since s_i is switchback, then $\|s_i\| = w_j|z_i| > w_j|s_i|$. Substituting in Equation (4.4) results in:

$$\begin{aligned}\|s_i'\| &\leq \frac{w_j}{\sin\frac{\alpha_j}{2}}\left(|s_i| + \frac{|L|}{m+1}\right) \\ &\leq \frac{w_j}{\sin\frac{\alpha_j}{2}}\left(\frac{\|s_i\|}{w_j} + \frac{|L|}{m+1}\right) \\ &= \frac{1}{\sin\frac{\alpha_j}{2}}\left(\|s_i\| + \frac{w_j|L|}{(m+1)}\right).\end{aligned}$$

□

Table 4.2 gathers the results of the claims just presented. It shows the bounds for each type of segment s_i' as compared to the type of s_i.

$s_i \quad \backslash \quad s_i'$	Regular	Braking	Switchback
Regular	$\|s_i\| + \frac{w_j\|L\|}{m+1}$	$-\sin\varphi_i'(\frac{\|s_i\|}{w_j} + \frac{\|L\|}{m+1})$	$\frac{1}{\sin\frac{\alpha_j}{2}}\left(\|s_i\| + \frac{w_j\|L\|}{m+1}\right)$
Braking	$\frac{w_j}{-\sin\varphi_i}\|s_i\| + \frac{w_j\|L\|}{m+1}$	$\|s_i\| + \frac{\|L\|}{m+1}$	$\frac{w_j}{\sin\frac{\alpha_j}{2}}\left(\frac{\|s_i\|}{-\sin\varphi_i} + \frac{\|L\|}{m+1}\right)$
Switchback	$\|s_i\| + \frac{w_j\|L\|}{m+1}$	$-\sin\varphi_i'\left(\frac{\|s_i\|}{w_j} + \frac{\|L\|}{m+1}\right)$	$\frac{1}{\sin\frac{\alpha_j}{2}}\left(\|s_i\| + \frac{w_j\|L\|}{(m+1)}\right)$

Table 4.2: The maximum cost of the approximated segment s_i' with respect to the actual shortest path segment cost $\|s_i\|$. Here $w_j = \mu_j \cos\phi_j$.

Lemma 4.2 *Given two vertices s and t of G, there exists a path $\Pi'(s,t)$ in G such that $\|\Pi'(s,t)\| \leq \frac{1}{\sin\frac{\alpha}{2}}\|\Pi(s,t)\| + \frac{W|L|k}{(m+1)\sin\frac{\alpha}{2}}$, where k is the number of segments of $\Pi(s,t)$.*

Proof: Claims 4.7 to 4.15 show that s'_j is bounded such that $\|s'_i\| \leq \frac{1}{\sin\frac{\alpha_j}{2}}\|s_i\| + \frac{w_j|L|}{(m+1)\sin\frac{\alpha_j}{2}}$. To see this, we make use of Property 4.1 which states that $-\sin\varphi_i \geq w_j$. By applying this worst case approximation to all segments of $\Pi'(s,t)$ we have:

$$\sum_{i=1}^{k}\|s'_i\| \leq \sum_{i=1}^{k}\frac{1}{\sin\frac{\alpha_j}{2}}\left(\|s_i\| + \frac{w_j|L|}{m+1}\right).$$

We can now put an upper bound on w_j and a lower bound on α_j:

$$\begin{aligned}\|\Pi'(s,t)\| &\leq \sum_{i=1}^{k}\frac{1}{\sin\frac{\alpha}{2}}\left(\|s_i\| + \frac{W|L|}{m+1}\right)\\ &\leq \frac{1}{\sin\frac{\alpha}{2}}\|\Pi(s,t)\| + \frac{W|L|k}{(m+1)\sin\frac{\alpha}{2}}.\end{aligned}$$

□

Theorem 4.1 *Let $\mathcal{P}$ be a polyhedral surface where h is the minimum distance from a vertex of $\mathcal{P}$ to the boundary of the union of its incident faces. Let $r = \epsilon h$ for some $0 < \epsilon < 1$. We can compute an approximation $\Pi'(s,t)$ of a weighted shortest anisotropic path $\Pi(s,t)$ between two vertices s and t of G such that $\|\Pi'(s,t)\| \leq \frac{1}{\sin\frac{\alpha}{2}}(\|\Pi(s,t)\| + W|L|)$. Moreover, we can compute this path in $O(nm\log nm + nm^2)$ time, where $m = \frac{k^*}{1-2r(k^*+1)}$. Here, $k^* = O(n\log_{\mathcal{F}}\frac{|L|}{r})$ and $\mathcal{F} = (1+\min(w,\sin\frac{\alpha}{2})\sin\theta)$.*

Proof: Consider splitting up $\Pi(s,t)$ into subpaths such that each subpath is a between-sphere subpath or an inside-sphere subpath. Let $k^* < k$ be the number of segments of all between-sphere subpaths. By setting $m = k^* - 1$ for each of these between-sphere subpaths, a proof similar to Lemma 4.2 can be used to bound the paths within the stated bounds of the theorem.

Examine now a particular inside-sphere subpath of $\Pi(s,t)$ that lies completely within some sphere C_v. This subpath is approximated by two segments of $\Pi'(s,t)$ each with length at most r. This two-link path has a cost which is at most $\frac{2Wr}{\sin\frac{\alpha}{2}}$. This

assumes that the two-link path uses the most expensive cost possible. Since inside-sphere subpaths must lie between consecutive between-sphere subpaths, the number of inside-sphere subpaths is at most k^*+1. Therefore, we can deduce that the cost for the approximations of inside-sphere subpaths is at most $\frac{2Wr(k^*+1)}{\sin\frac{\alpha}{2}}$. By setting $m = \frac{k^*|L|}{|L|-2r(k^*+1)} - 1$, and summing the approximate costs for all between-sphere and inside-sphere subpaths, we have:

$$\begin{aligned}
\|\Pi'(s,t)\| &\leq \frac{1}{\sin\frac{\alpha}{2}}\left(\|\Pi(s,t)\| + \frac{Wk^*|L|}{m+1}\right) + \left(\frac{2Wr(k^*+1)}{\sin\frac{\alpha}{2}}\right) \\
&\leq \frac{1}{\sin\frac{\alpha}{2}}\left(\|\Pi(s,t)\| + \frac{Wk^*|L|(|L|-2r(k^*+1))}{k^*|L|} + 2Wr(k^*+1)\right) \\
&\leq \frac{1}{\sin\frac{\alpha}{2}}\left(\|\Pi(s,t)\| + W|L| - 2Wr(k^*+1) + 2Wr(k^*+1)\right) \\
&\leq \frac{1}{\sin\frac{\alpha}{2}}\left(\|\Pi(s,t)\| + W|L|\right)
\end{aligned}$$

As for the time complexity, Dijkstra's algorithm (Theorem 1.2) is applied which runs in $O(nm\log nm + nm^2)$ time where $m = O(\frac{k^*|L|}{|L|-2r(k^*+1)})$. Note that we can increase m to $\frac{k^*}{1-2r(k^*+1)}$ to simplify the bound as long as $|L| > 1$. The value of k^* follows from Claim 4.3.

□

We have just given a bound on the accuracy of the approximated path produced using this simple approach. The approximated path accuracy depends upon geometric parameters of the terrain: L, α, and W. Since the value of α can be arbitrarily small, the path accuracy can be arbitrarily bad. However, in most traversable terrain, there is a limit on reasonable values of α which allows the path accuracy to be reasonably bounded in practice.

During the analysis, we made the assumption that each segment of $\Pi(s,t)$ was approximated in the worst possible manner. That is, we assumed that all segments of $\Pi'(s,t)$ were switchback. Clearly this is very unlikely since the switchback paths

are more expensive than the regular and braking segments. Therefore, we can easily conclude that the stated path accuracy is not a "tight" bound for the average case. It is very likely that the bound will be much better in practice. Due to its simplicity it is likely that this algorithm is of practical value and is likely to produce accurate paths with only a few Steiner points added per edge (as in Chapter 2). in Section 4.5 we describe some experiments performed on an implementation of our algorithm here and show that the accuracy converges after only a few Steiner points are added per edge.

4.4 An ϵ-Approximation

In this section, we describe how the graph G can be modified so that an ϵ-approximation is obtained. The algorithm itself is the same, and differs from the previous scheme mainly in the number and placement of Steiner points on the edges of $\mathcal{P}$.

From the results in Table 4.2, it is easily seen that in order to improve the path accuracy, we must ensure that segments s_i of $\Pi(s,t)$ are approximated with segments of $\Pi'(s,t)$ that have cost at most $(1+\epsilon)\|s_i\|$. We also make sure that approximation segments have the same direction type as the segments which they are approximating. For example, braking segments of $\Pi(s,t)$ must be approximated with braking segments of $\Pi'(s,t)$. These modifications will ensure that each segment s_i of $\Pi(s,t)$ is approximated with a corresponding segment s_i' of $\Pi'(s,t)$ such that $\|s_i'\| \leq (1+f(\epsilon))\|s_i\|$, where $f(\epsilon)$ is some function which depends on geometric parameters of $\mathcal{P}$, but is independent of n. By setting $\epsilon_1 = f(\epsilon)$, this will ensure that $\Pi'(s,t)$ is an ϵ_1-approximation of $\Pi(s,t)$. We will show that the main difference in the results of this algorithm and that of Chapter 3 is the number of Steiner points that are used, their interconnection, and in the size of each r_v.

4.4.1 Constructing the Graph

We begin by constructing a graph G_j for each face f_j of $\mathcal{P}$ by adding Steiner points along edges of f_j in three stages. In the first stage, we add enough Steiner points to ensure that the distance between adjacent Steiner points on an edge is sufficiently small. In essence, we will make this distance small enough to ensure that it is at most ϵ times the length of a shortest path segment which passes through it. This stage is analogous to the construction used in Chapter 3. For each vertex v of face f_j we do the following: Let e_q and e_p be the edges of f_j incident to v. First, place Steiner points on edges e_q and e_p at distance r_v from v; call them q_1 and p_1, respectively. By definition, $|\overline{vq_1}| = |\overline{vp_1}| = r_v$. Define $\delta = (1 + \epsilon \sin\theta_v)$ if $\theta_v < \frac{\pi}{2}$, otherwise $\delta = (1+\epsilon)$. We now add Steiner points $q_2, q_3, ..., q_{\iota_q - 1}$ along e_q such that $|\overline{vq_j}| = r_v \delta^{j-1}$ where $\iota_q = \log_\delta(|e_q|/r_v)$. Similarly, add Steiner points $p_2, p_3, ..., p_{\iota_p - 1}$ along e_p, where $\iota_p = \log_\delta(|e_p|/r_v)$. As with the strategy used in Chapter 3, overlapping sets of Steiner points on an edge are merged.

Now we add a second stage of Steiner points which are required to ensure that there exists an approximation segment of the same direction type as a shortest path segment which passes between the same Steiner points. Recall that our model of computation allows for eight direction ranges (three impermissible, one braking and four regular) per face as shown in Figure 4.3. We will add a set of Steiner points to f_j corresponding to the braking range and the (up to four) regular ranges. We give here a description of how to add the Steiner points corresponding to the braking range; the Steiner point sets for the regular ranges are constructed in the same manner.

The idea is as follows. For each vertex v of the face, we will extend rays in the directions of the critical angle vectors for the particular range. We will add a Steiner point at the intersection of this ray with an edge of the face. We then extend a ray from the newly created Steiner point in the opposite direction to a critical angle

vector to create another Steiner point. We then repeat the processes to generate sets of Steiner points on the edges of the face. The process of adding Steiner points stops when the last Steiner point added is sufficiently close to a vertex (which is not v) of the face. This stopping criteria provides a method of bounding the number of Steiner points added. As will be seen later, generating the Steiner point sets from the three vertices is sufficient.

Let $\vec{u}$ and $\vec{v}$ be the critical braking angle directions for the range on the plane defined by f_j. Consider now each vertex v of f_j and apply the following algorithm twice (once as is stated and then again where $\vec{u}$ and $\vec{v}$ are swapped):

LET $q = v$.
WHILE (q does not lie within C_{v_i} of f_j, where $v_i \neq v$) DO {
    LET x_q be a ray from q in direction $\vec{u}$.
    IF (x_q intersects an edge e of f_j) THEN {
        p = intersection point of x_q and e.
        Add p as a Steiner point on e.
        LET x_p be a ray from p in direction $-\vec{v}$.
        IF (x_p does not intersect an edge e of f_j) THEN STOP
        ELSE {
            q = intersection point of x_p with e.
            Add q as a Steiner point on e.
        }
    }
}

Figure 4.13 shows how these Steiner points are added using this technique for a

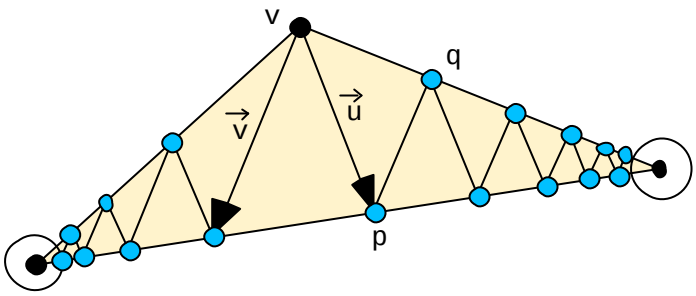

Figure 4.13: Adding Steiner points to a face corresponding to a braking range.

braking range. Note that when applying this stage to the adjacent faces of f_j, some additional Steiner points will be added to the edges of f_j. When creating the graph for an adjacent face f_{j+1}, each edge of f_j may also gain a set of Steiner points due to this second stage construction. Hence, each edge may have two such sets of Steiner points. The first and second stage of Steiner points along with the vertices of f_j become vertices of G_j.

Connect a pair of vertices in G_j by two oppositely directed edges if and only if 1) they represent Steiner points lying on different edges of f_j or 2) they represent adjacent Steiner points lying on the same edge of f_j. In addition, for each vertex of G_j which corresponds to a vertex, say v, of f_j, connect it with two oppositely directed edges to 1) all vertices of G_j that represent Steiner points lying on the edge opposite to v, and 2) the two vertices of G_j corresponding to the two closest Steiner points that lie on the two incident edges of v. This interconnection strategy is similar to the complete graph used in the previous schemes.

In the final stage, we expand G_j by placing additional Steiner points within at least one of the spheres around a vertex of f_j. Let q be a Steiner point (or vertex

of f_j) on edge e_q of f_j which was added during the first or second stage of Steiner placement (including those from adjacent faces as well). Extend rays from q in the directions of $\vec{u}$ and $\vec{v}$. For each ray, if it intersects an edge $e_p \neq e_q$ of f_j at some point x within some C_v of vertex v of f_j then add a Steiner point at x. Add x as a vertex of G_j and add edge $\overrightarrow{qx}$ to G_j. Also add edge $\overrightarrow{xv}$ to G_j. Now extend rays from q in the directions of $-\vec{u}$ and $-\vec{v}$. For each ray, if it intersects an edge $e_p \neq e_q$ of f_j at some point y within C_v then add a Steiner point at y. Add y as a vertex of G_j and add edge $\overrightarrow{yq}$ to G_j. Also add edge $\overrightarrow{vy}$ to G_j. Figure 4.14 shows how these new Steiner points are added near a vertex v.

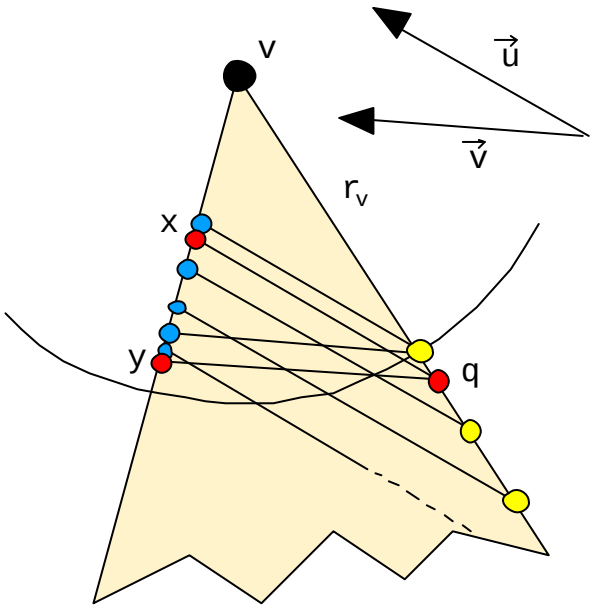

Figure 4.14: Adding Steiner points to a face within a sphere around vertex v.

Let q_a and q_{a+1} be two adjacent Steiner points added on an edge within a sphere C_v as just mentioned. Let p_b and p_{b+1} be the Steiner points that generated q_a and q_{a+1}, respectively. If $|\overline{q_a q_{a+1}}| > r_v(\delta - 1)$ then we add additional evenly spaced Steiner

points between q_a and q_{a+1}. We add only enough Steiner points to ensure that the distance between two adjacent points is at most $r_v(\delta - 1)$. Once again, connect each of these new Steiner points, say p to v with two oppositely directed edges. Also, connect p to p_b and p_{b+1} with two oppositely directed edges each. Keep in mind that although we just described the addition of these Steiner points with respect to the two critical directions $\vec{u}$ and $\vec{v}$ for the braking range, we must also add similar sets of Steiner points for the regular ranges.

We are done adding vertices and edges to G_j. We must now assign appropriate weights to the edges. For each edge $\overrightarrow{ab}$ of G_j, we set its weight as follows:

1. If $\overrightarrow{ab}$ is regular then its weight is set to $w_j|\overline{ab}|$.
2. If $\overrightarrow{ab}$ is braking then its weight is set to $-\sin\theta_i|\overline{ab}|$ where θ_i is the inclination angle of $\overrightarrow{ab}$.
3. If $\overrightarrow{ab}$ is switchback then its weight is set to $\frac{w_j}{\sin\frac{\alpha_j}{2}}|\overline{ab}|$.

This completes the construction of G_j. A graph G is defined to be the union $G_1 \cup G_2 \cup ... \cup G_n$.

Claim 4.16 *At most* $m = O(\log_\delta(|L|/r) + \log_{\mathcal{F}}(r/|L|) + \frac{1}{\delta - 1})$ *Steiner points are added to each edge of* f_j, *where* $\mathcal{F} = \frac{1+\cos(\theta+\lambda)}{1+\cos(\theta-\lambda)}$ *and* $\delta = 1 + \epsilon\sin\theta_v$.

Proof: Chapter 3 has shown that the number of Steiner points added in stage 1 of the algorithm is $O(\log_\delta(|L|/r))$. Consider the Steiner points added in stage 2 of the algorithm. Let $\vec{u}$ and $\vec{v}$ be the two critical angle vectors corresponding to either the braking range or one of the regular ranges of f_j. Let γ be the angle between $\vec{u}$ and $\vec{v}$. We will examine the creation of Steiner points on two particular edges e_1 and e_2 of f_j. We will bound the number of Steiner points on e_1 only, since the number of

points created on e_2 is of the same order. There are at most a constant times more Steiner points added to e_1 from each of the other ranges. Let v be the vertex at the intersection of e_1 and e_2 such that the angle formed by e_1 and e_2 in f_j is θ_v. Let $q_1, q_2, ..., q_\iota$ be the Steiner points added to e_1 during stage 2 of the algorithm such that q_1 lies at distance r from v. Also let $q_{\iota+1}$ be the vertex of e_1 that is not v. We will assume that $q_1 = r$ since we can always add an additional Steiner point to ensure this. Let $p_1, p_2, ..., p_{\iota+1}$ be the Steiner points on e_2 which were also created during stage 2 and correspond to the intersection of rays from $q_1, q_2, ..., q_{\iota+1}$, respectively in direction $\vec{u}$ or $\vec{v}$. Let $x_i = \overline{q_{i+1}p_{i+1}}, 0 \leq i \leq \iota$. Let $\sigma = \angle q_i q_{i+1} p_{i+1}$. Lastly, let $\delta_0 = |\overline{vq_1}| = r$ and let $\delta_i = |\overline{q_i q_{i+1}}|, 1 \leq i \leq \iota$. Figure 4.15 illustrates these important definitions. We will assume that $\theta_v = \theta$. Furthermore, we will assume the worst case in which $|e_1| = |L|$. Note that this will result in an upper bound on the number of Steiner points. We let $e = e_1$ below to reduce clutter. We will determine the length of $\delta_i, 0 < i \leq \iota$ and then determine ι. From the sine law:

$$\frac{x_\iota}{\sin\theta} = \frac{|e|}{\sin(\pi - (\theta + \sigma))} = \frac{|e|}{\sin(\theta + \sigma)} \tag{4.5}$$

Hence $x_\iota = \frac{|e|\sin\theta}{\sin(\theta+\sigma)}$. Again using the sine law:

$$\frac{\delta_\iota}{\sin\gamma} = \frac{x_\iota}{\sin(\pi - (\gamma + \sigma))} = \frac{x_\iota}{\sin(\gamma + \sigma)} \tag{4.6}$$

By combining Equations (4.5) and (4.6) we have

$$\delta_\iota = \frac{|e|\sin\theta\sin\gamma}{\sin(\theta+\sigma)\sin(\gamma+\sigma)} \tag{4.7}$$

Let $\Delta_0 = r$ and $\Delta_i = \delta_0 + \delta_1 + ... + \delta_i$, where $1 \leq i \leq \iota$. By definition, $\Delta_\iota = |e|$. Let $f(\gamma, \theta, \sigma) = \left(\frac{\sin\theta\sin\gamma}{\sin(\theta+\sigma)\sin(\gamma+\sigma)}\right)$. Then,

$$\Delta_\iota = |e|$$
$$\Delta_{\iota-1} = \Delta_\iota - \delta_\iota = |e|(1 - f(\gamma, \theta, \sigma)),$$

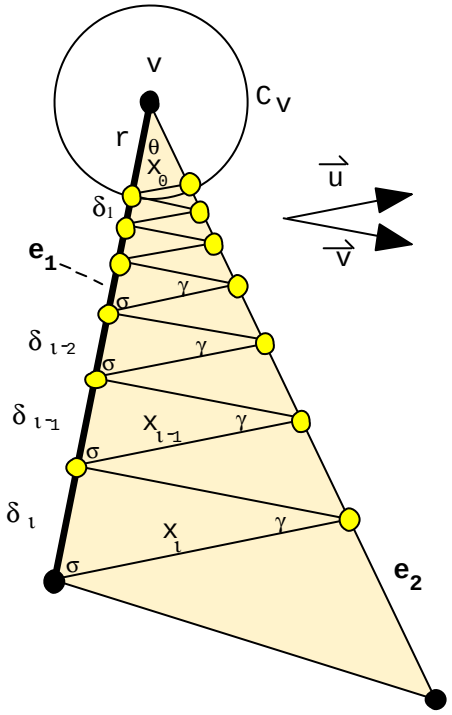

Figure 4.15: The angles defined during the creation of stage 2 Steiner points for a face f_j based on a single permissible range.

$$\Delta_{\iota-2} = \Delta_{\iota-1} - (\delta_{\iota-1}) = |e|(1 - f(\gamma, \theta, \sigma))^2$$

...

$$\Delta_{\iota-m} = \Delta_{\iota-(m-1)} - (\delta_{\iota-(m-1)}) = |e|(1 - f(\gamma, \theta, \sigma))^m$$

In order to bound ι, we would like to find a value for m such that $\Delta_{\iota-m} = \Delta_0 = r$ and so

$$(1 - f(\gamma, \theta, \sigma))^m = \frac{r}{|e|} \tag{4.8}$$

Let $\mathcal{F} = 1 - f(\gamma, \theta, \sigma)$. By taking the logarithm (base $\mathcal{F}$) of both sides of Equation

(4.8) we obtain:

$$\log_{\mathcal{F}}(\mathcal{F})^m = \log_{\mathcal{F}}\left(\frac{r}{|e|}\right)$$

and so

$$m = \log_{\mathcal{F}}\left(\frac{r}{|e|}\right) \tag{4.9}$$

For $\mathcal{F}$ to be a valid base, we need to show that $0 < f(\gamma,\theta,\sigma) < 1$. Clearly, $0 < f(\gamma,\theta,\sigma)$ since $\sigma+\theta+\gamma \leq \pi$. We will show that $\frac{1}{f(\gamma,\theta,\sigma)} > 1$:

$$\begin{aligned}
1 &< \frac{1}{f(\gamma,\theta,\sigma)} = \frac{\sin(\theta+\sigma)\sin(\gamma+\sigma)}{\sin\theta\sin\gamma} \\
&< \frac{(\sin\theta\cos\sigma+\sin\sigma\cos\theta)(\sin\gamma\cos\sigma+\sin\sigma\cos\gamma)}{\sin\theta\sin\gamma} \\
&< \frac{\cos^2\sigma\sin\theta\sin\gamma+\sin\sigma\cos\sigma\sin\theta\cos\gamma+\sin^2\sigma\cos\theta\cos\gamma+\sin\sigma\cos\sigma\cos\theta\sin\gamma}{\sin\theta\sin\gamma} \\
&< \cos^2\sigma+\sin\sigma\cos\sigma\cot\gamma+\sin^2\sigma\cot\theta\cot\gamma+\sin\sigma\cos\sigma\cot\theta \\
&< 1-\sin^2\sigma+\sin^2\sigma\cot\theta\cot\gamma+\sin\sigma\cos\sigma\cot\gamma+\sin\sigma\cos\sigma\cot\theta
\end{aligned}$$

This can be rewritten as:

$$0 < \sin^2\sigma(\cot\theta\cot\gamma-1)+\sin\sigma\cos\sigma\cot\gamma+\sin\sigma\cos\sigma\cot\theta$$

Since $0<\sigma<\pi$ then $\sin\sigma>0$ and we can divide both sides by $\sin\sigma$:

$$0 < \sin\sigma(\cot\theta\cot\gamma-1)+\cos\sigma\cot\gamma+\cos\sigma\cot\theta$$

With some final manipulation, we obtain:

$$\begin{aligned}
\sin\sigma\left(\frac{\cos\theta\cos\gamma-\sin\theta\sin\gamma}{\sin\theta\sin\gamma}\right) &> -\cos\sigma\left(\frac{\cos\gamma\sin\theta+\cos\theta\sin\gamma}{\sin\gamma\sin\theta}\right) \\
\sin\sigma\cos(\theta+\gamma) &> -\cos\sigma\sin(\theta+\gamma) \\
0 &< \sin\sigma\cos(\theta+\gamma)+\cos\sigma\sin(\theta+\gamma) \\
0 &< \sin(\sigma+\theta+\gamma)
\end{aligned}$$

which is true since $0 < \sigma + \theta + \gamma < \pi$.

For the purpose of simplifying $\mathcal{F}$, we will now place an upper bound on σ. In the worst case (i.e., the number of Steiner points is maximized), $\mathcal{F}$ is minimized which implies that $f(\gamma, \theta, \sigma)$ is maximized. To maximize $f(\gamma, \theta, \sigma)$, we must minimize $\sin(\theta + \sigma)\sin(\gamma + \sigma)$. Taking the derivative with respect to σ allows the problem to be transformed so that we need to prove $\cos(\sigma+\theta)\sin(\sigma+\gamma)+\cos(\sigma+\gamma)\sin(\sigma+\theta) = 0$. We can expand this and group the terms differently as follows:

$$
\begin{aligned}
0 &= (\cos\sigma\cos\theta - \sin\sigma\sin\theta)(\sin\sigma\cos\gamma + \cos\sigma\sin\gamma) + \\
&\quad (\cos\sigma\cos\gamma - \sin\sigma\sin\gamma)(\sin\sigma\cos\theta + \cos\sigma\sin\theta) \\
&= \cos\sigma\sin\sigma\cos\gamma\cos\theta - \sin^2\sigma\sin\theta\cos\gamma + \cos^2\sigma\cos\theta\sin\gamma - \sin\sigma\cos\sigma\sin\theta\sin\gamma + \\
&\quad \cos\sigma\sin\sigma\cos\gamma\cos\theta - \sin^2\sigma\cos\theta\sin\gamma + \cos^2\sigma\cos\gamma\sin\theta - \sin\sigma\cos\sigma\sin\theta\sin\gamma \\
&= 2\cos\sigma\sin\sigma\cos\gamma\cos\theta - \sin^2\sigma(\sin\theta\cos\gamma + \cos\theta\sin\gamma) + \\
&\quad \cos^2\sigma(\cos\theta\sin\gamma + \cos\gamma\sin\theta) - 2\cos\sigma\sin\sigma\sin\theta\sin\gamma \\
&= 2\cos\sigma\sin\sigma\cos\gamma\cos\theta - 2\sin\sigma\cos\sigma\sin\theta\sin\gamma + (2\cos^2\sigma - 1)\sin(\gamma+\theta) \\
&= 2\cos\sigma\sin\sigma\cos(\gamma+\sigma) + (2\cos^2\sigma - 1)\sin(\gamma+\theta) \\
&= \sin 2\sigma\cos(\gamma+\sigma) + (2(1+\cos 2\sigma)/2 - 1)\sin(\gamma+\theta) \\
&= \sin 2\sigma\cos(\gamma+\theta) + \cos 2\sigma\sin(\gamma+\theta) \\
&= \sin(2\sigma+\gamma+\theta)
\end{aligned}
$$

Since σ, γ and θ are all greater than zero, the only possible solution is when $2\sigma+\gamma+\theta = \pi$. This implies that $\sigma = \pi/2 - (\gamma+\theta)/2$. If we make this substitution into $\mathcal{F}$ we obtain:

$$
\begin{aligned}
\mathcal{F} &= 1 - \frac{\sin\theta\sin\gamma}{\sin(\pi/2 - (\gamma+\theta)/2 + \theta)\sin(\pi/2 - (\gamma+\theta)/2 + \gamma)} \\
&= 1 - \frac{\sin\theta\sin\gamma}{\sin(\pi/2 - (\gamma-\theta)/2)\sin(\pi/2 + (\gamma-\theta)/2)}
\end{aligned}
$$

$$
\begin{aligned}
&= 1 - \frac{\sin\theta \sin\gamma}{\cos((\gamma-\theta)/2)\cos(-(\gamma-\theta)/2)} \\
&= 1 - \frac{\sin\theta \sin\gamma}{\cos^2((\theta-\gamma)/2)} \\
&= 1 - \frac{\sin\theta \sin\gamma}{\left(\frac{1+\cos(\theta-\gamma)}{2}\right)} \\
&= 1 - \frac{2\sin\theta \sin\gamma}{1+\cos(\theta-\gamma)} \\
&= \frac{1+\cos(\theta-\gamma) - 2\sin\theta\sin\gamma}{1+\cos(\theta-\gamma)} \\
&= \frac{1+\cos\theta\cos\gamma+\sin\theta\sin\gamma - 2\sin\theta\sin\gamma}{1+\cos(\theta-\gamma)} \\
&= \frac{1+\cos\theta\cos\gamma-\sin\theta\sin\gamma}{1+\cos(\theta-\gamma)} \\
&= \frac{1+\cos(\theta+\gamma)}{1+\cos(\theta-\gamma)}
\end{aligned}
$$

Since γ is defined as the angle between the matched pair of critical angles for a braking or regular range, its minimum value is λ. This minimal value generates the most Steiner points and will be used as an upper bound for counting the maximum number of Steiner points added during stage 2. In stage 3, we add Steiner points in two phases. The first phase adds at most the same order of Steiner points as in stages 1 and 2 since the new Steiner points are computed by shooting at most two rays from the Steiner points created in stages 1 and 2. In the second phase we add enough Steiner points to the interior of C_v along e so that the distance between two adjacent points is at most $r_v \epsilon \sin\theta_v$. Hence we add $O(\frac{1}{\delta-1})$ points. By combining the number of Steiner points calculated for stages 1,2 and 3, we obtain the stated bounds. □

Claim 4.17 *G has $O(n\log_\delta(|L|/r)+n\log_{\mathcal{F}}(r/|L|)+\frac{n}{\delta-1})$ vertices and $O(n(\log_\delta(|L|/r)+\log_{\mathcal{F}}(r/|L|)+\frac{n}{\delta-1})^2)$ edges, where $\mathcal{F}=\frac{1+\cos(\theta+\lambda)}{1+\cos(\theta-\lambda)}$ and $\delta = 1+\epsilon\sin\theta$.*

Proof: Follows from applying Claim 4.16 to each edge of $\mathcal{P}$.

Lemma 4.3 *Let $\overline{ab}$ be the smallest segment contained within f_j such that one endpoint of $\overline{ab}$ intersects e_q between Steiner points q_i and q_{i+1} and the other endpoint intersects e_p, where q_i and q_{i+1} do not lie in any spheres C_v of f_j. It holds that $|\overline{q_i q_{i+1}}| \leq \epsilon|\overline{ab}|$. Furthermore, if $\theta_v < \frac{\pi}{2}$ then $\overline{ab}$ is a perpendicular to e_p and if $\theta_v \geq \frac{\pi}{2}$ then $|\overline{ab}| \geq |\overline{vq_i}|$.*

Proof: Follows from Lemma 3.1 due to the placement of Steiner points in stage 1 of creating G.

□

4.4.2 Choosing an Approximated Path For Analysis

We describe here the construction of a path $\Pi'(s,t)$ in G which will be bounded in Section 4.4.3. Note again that Dijkstra's algorithm may produce a better path than the one constructed here. As with the simple approximation scheme of Section 4.3, we will construct and bound the cost of $\Pi'(s,t)$. We do not describe the construction of $\Pi''(s,t)$ for reporting the path, although it is constructed similar to that from Section 4.3 and can be shown to have a finite number of segments.

Let x and y be the endpoints of some segment s_i of $\Pi(s,t)$ and let x (respectively y) lie on edge e_q (respectively e_p) of f_j. Let q_a and q_b (respectively p_a and p_b) be the Steiner points on e_q (respectively e_p) between which x (respectively y) lies.

Claim 4.18 *At least one of $\overrightarrow{q_a p_b}$ or $\overrightarrow{q_b p_a}$ is of the same direction type as s_i.*

Proof: Since s_i is an edge of $\Pi(s,t)$, it must travel in either a regular or braking direction (i.e., not impermissible). We will assume for this proof that it is braking and hence will show that at least one of $\overrightarrow{q_a p_b}$ and $\overrightarrow{q_b p_a}$ is also braking based on the Steiner points added in Stage 2. Note that the proof also applies in the case where

s_i is regular since the Stage 2 Steiner points were made for those ranges as well. We will say that the critical angle directions for this range are $\vec{u}$ and $\vec{v}$. Let q'_a and q'_b be the closest Steiner points to x that were created in Stage 2 from the same range type as s_i. Let p'_a and p'_b be the Steiner points that were also created in Stage 2 (also same range type) via the extension of rays from q'_a in the directions $\vec{v}$ and $\vec{u}$, respectively. Similarly, let p''_a and p''_b be the Steiner points that were created in Stage 2 via the extension of a ray from q'_b in the directions $\vec{v}$ and $\vec{u}$, respectively (note that $p'_b = p''_a$). Figure 4.16 a) and b) show these Steiner points. It may be that $q_a = q'_a$ and/or $q_b = q'_b$. We will assume that these are not equal (as shown in the Figure), although the claim will still hold when either of these are equal. By construction of G during Stage 2, p_a and p_b must lie between either 1) p'_a and p'_b or 2) p''_a and p''_b. Consider the first case and extend rays from q_a in the directions of $\vec{v}$ and $\vec{u}$. It is easily seen that $\overrightarrow{q_a p_b}$ lies between these rays and hence is of the same type as s_i. Consider now the second case and extend similar rays from q_b to show that $\overrightarrow{q_b p_a}$ lies between these rays as well. Hence, at least one of $\overrightarrow{q_a p_b}$ or $\overrightarrow{q_b p_a}$ is of the same direction type as s_i.

□

We begin our path construction by choosing a segment s'_i in G_j which approximates a segment s_i crossing face f_j. If s_i is an outside-sphere or overlapping-sphere segment, then choose s'_i to be one of $\overrightarrow{q_a p_a}$, $\overrightarrow{q_a p_b}$, $\overrightarrow{q_b p_a}$ and $\overrightarrow{q_b p_b}$ such that s'_i is of the same direction type as s_i and $\|s'_i\|$ is minimized. Claim 4.18 ensures that at least one of these segments is of the same type as s_i. For the sake of analysis, we will assume that s'_i is chosen so as to have the same direction type as s_i and we will bound s'_i accordingly. In practice however, we may choose a segment with less cost, since we are choosing the minimum of these four. Note that this choice also pertains to the special case in which $e_q = e_p$. Note also that if s_i is an overlapping-sphere segment, then one of q_a, q_b, p_a or p_b may degenerate to a vertex of f_j. In the case where s_i is an inside-sphere segment, there is no corresponding segment s'_i in $\Pi'(s,t)$.

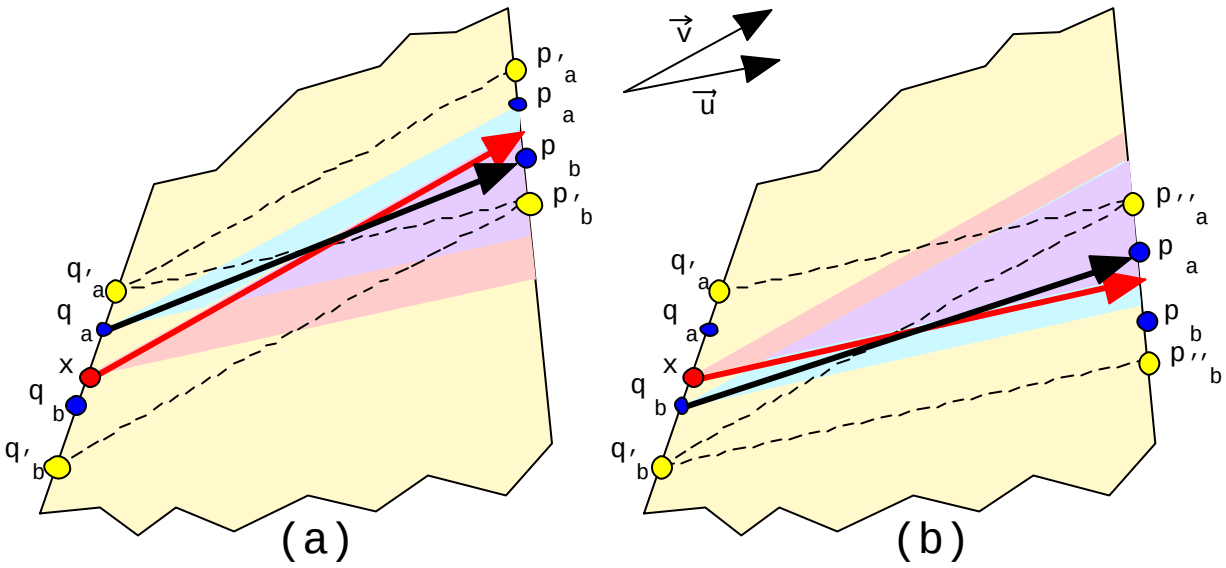

Figure 4.16: Example showing how s'_i is of the same direction type as s_i.

At this point, we have approximations for all outside-sphere and overlapping-sphere segments but they are disconnected and therefore do not form a path. We will now add edges joining consecutive non-inside-sphere segments s_i and s_{i+1} of $\Pi(s,t)$ with corresponding approximation segments s'_i of G_j and s'_{i+1} of G_{j+1}, respectively. Let e be the edge of $\mathcal{P}$ joining faces f_j and f_{j+1}. Let q be the endpoint of s'_i lying on e and let p be the endpoint of s'_{i+1} lying on e. It is easily seen that either $q = p$ or q and p are adjacent Steiner points on e. If $q = p$, then s'_i and s'_{i+1} are already connected. If $q \neq p$ then let s''_i be the edge in G_j from q to p. Figure 4.17 shows two examples of how s''_i is used to connect s'_i and s'_{i+1}.

The addition of these segments (i.e., all s''_i) ensures that all segments of between-sphere subpaths are connected to form subpaths. We now need to interconnect the between-sphere subpaths so that $\Pi'(s,t)$ is connected.

Consider two consecutive between-sphere subpaths of $\Pi'(s,t)$, say $\Pi'(\sigma_{a-1}, \sigma_a)$ and

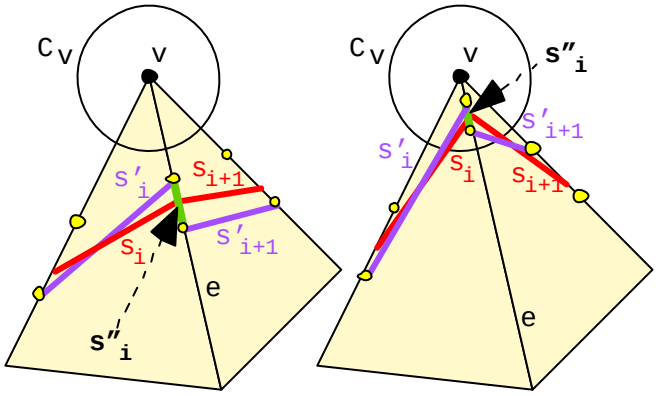

Figure 4.17: Examples showing the how segments s'_i and s'_{i+1} of $\Pi'(s,t)$ are connected via s''_i.

$\Pi'(\sigma_a, \sigma_{a+1})$. They are disjoint from one another, however, the first path ends at a Steiner point within sphere C_{σ_a} and the second path starts at a Steiner point within C_{σ_a}. Join the end of $\Pi'(\sigma_{a-1}, \sigma_a)$ and the start of $\Pi'(\sigma_a, \sigma_{a+1})$ to vertex v_{σ_a} by two segments (which are edges of G created in Stage 3). These two segments together will form an inside-sphere subpath and will be denoted as $\Pi'(\sigma_a)$. This step is repeated for each consecutive pair of between-sphere subpaths so that all subpaths are joined to form $\Pi'(s,t)$. (The example of Figure 4.18 shows how between-sphere subpaths are connected to inside-sphere subpaths.) Constructing a path in this manner results in a connected path that lies on the surface of $\mathcal{P}$.

4.4.3 Computing an ϵ-Approximation Bound on the Path

We give here a bound $\|\Pi'(s,t)\|$ on the cost of $\Pi'(s,t)$. We will assume that t lies outside τ_s. To begin, a bound is shown for each of the between-sphere path segments. The claims to follow bound each non-inside-sphere face crossing segment, say s'_i of $\Pi'(s,t)$. The claims give bounds for the three possible direction types of s'_i. That

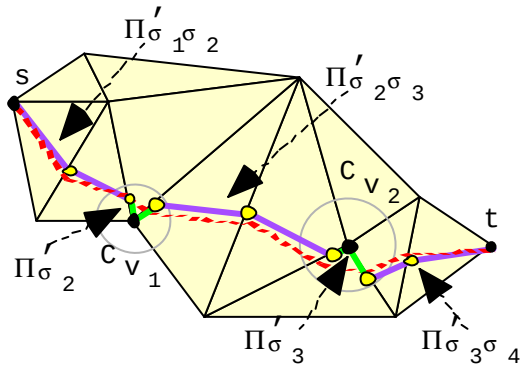

Figure 4.18: An example showing the between-sphere and inside-sphere subpaths that connect to form the approximated path $\Pi'(s,t)$.

is, we bound the weighted cost of s'_i for the cases in which s'_i (and hence s_i) are regular, braking and switchback. Recall that x and y are the endpoints of s_i and that x (respectively y) lies on edge e_q (respectively e_p) of f_j. Also recall that q_a and q_b (respectively p_a and p_b) are the Steiner points on e_q (respectively e_p) between which x (respectively y) lies. In the proofs of these claims, we will assume that $s'_i = \overline{q_a p_b}$; similar proofs hold when $s'_i = \overline{q_b p_a}$.

Claim 4.19 *If s_i and s'_i are regular then $\|s'_i\| \leq (1+2\epsilon)\|s_i\|$*

Proof: Through triangle inequality, we can state that: $|s'_i| \leq |\overline{q_a x}| + |s_i| + |\overline{y p_b}|$. Since Claim 4.18 ensures that s'_i is indeed regular, then $\|s'_i\| = w_j|s'_i|$ and $\|s_i\| = w_j|s_i|$ and so

$$\begin{aligned} \frac{\|s'_i\|}{w_j} &\leq |\overline{q_a x}| + \frac{\|s_i\|}{w_j} + |\overline{y p_b}| \\ &\leq |\overline{q_a q_b}| + \frac{\|s_i\|}{w_j} + |\overline{p_a p_b}|. \end{aligned}$$

Lemma 4.3 ensures that $|\overline{q_a q_b}| \leq \epsilon|s_i|$ and $|\overline{p_a p_b}| \leq \epsilon|s_i|$. Therefore,

$$\frac{\|s'_i\|}{w_j} \leq 2\epsilon|s_i| + \frac{\|s_i\|}{w_j}$$

$$= (1+2\epsilon)\frac{\|s_i\|}{w_j}.$$

and so $\|s_i'\| \leq (1+2\epsilon)\|s_i\|$.

□

Claim 4.20 *If s_i and s_i' are braking then $\|s_i'\| \leq \left(1+\frac{2\epsilon}{w_j}\right)\|s_i\|$, where w_j is weight of the face through which s_i and s_i' pass.*

Proof: First, examine the maximum possible height difference between s_i and s_i'. It is easily seen that $-\sin\varphi_i'|s_i'| \leq -\sin\varphi_i|s_i| + |\overline{q_aq_b}| + |\overline{p_ap_b}|$.

Since Claim 4.18 ensures that s_i' is braking, we can replace the terms above with the weighted costs $\|s_i'\|$ and $\|s_i\|$. Also, by using Lemma 4.3 again, we can show that $|\overline{q_aq_b}|$ and $|\overline{p_ap_b}|$ are at most $\epsilon|s_i|$ each. Therefore,

$$\begin{aligned}\|s_i'\| &\leq \|s_i\| + 2\epsilon|s_i| \\ &= \|s_i\| + \frac{2\epsilon\|s_i\|}{-\sin\varphi_i} \\ &= (1+\frac{2\epsilon}{-\sin\varphi_i})\|s_i\|.\end{aligned}$$

The minimum possible value of $-\sin\varphi_i$ is at the critical braking angles (i.e., see Property 4.1 in which $-\sin\varphi_i = w_j$). Therefore, $\|s_i'\| \leq (1+\frac{2\epsilon}{w_j})\|s_i\|$.

□

Claim 4.21 *If s_i and s_i' are switchback then $\|s_i'\| \leq \left(1+\frac{2\epsilon}{\sin\frac{\alpha}{2}}\right)\|s_i\|$*

Proof: Figure 4.19 shows segments s_i and s_i' along with their corresponding switchback paths $z_i = z_{xy}$ and $z_i' = z_{q_ap_b}$. It is easily seen that

$$|z_{q_ap_b}| = |z_{q_ax}| + |z_{xy}| + |z_{yp_b}|.$$

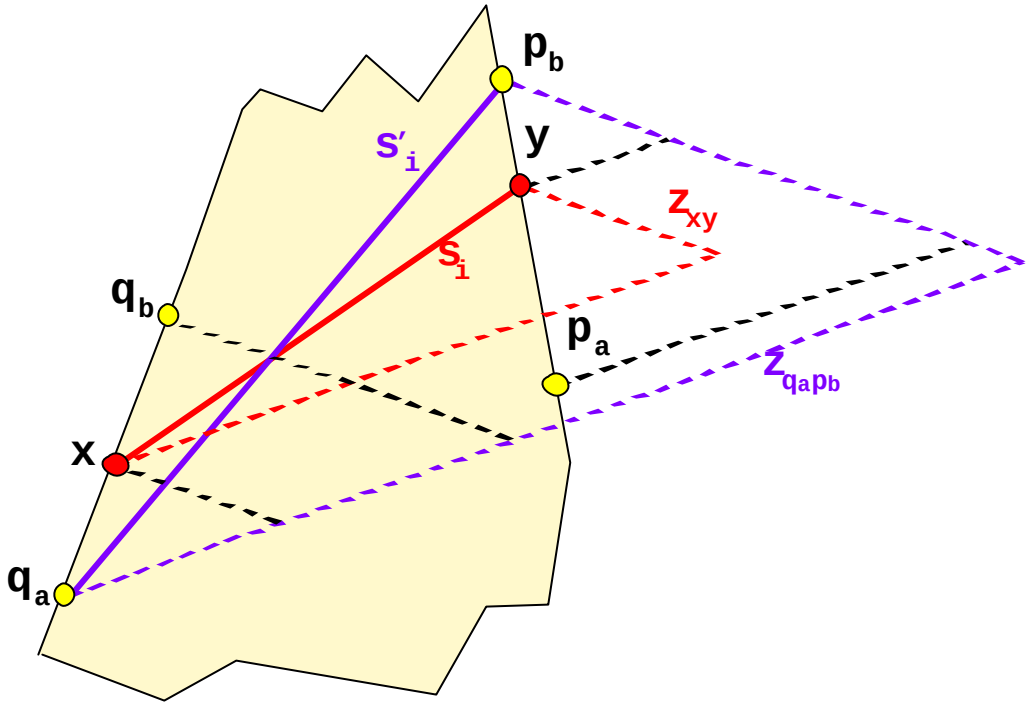

Figure 4.19: The switchback paths z_{xy} and $z_{q_a p_b}$ corresponding to segments s_i and s_i', respectively.

We can put an upper bound on $|z_{q_a x}|$ and $|z_{y p_b}|$ as follows:

$$\begin{aligned} |z_{q_a p_b}| &\leq |z_{q_a q_b}| + |z_{xy}| + |z_{p_a p_b}| \\ &\leq \frac{|\overline{q_a q_b}|}{\sin\frac{\alpha}{2}} + |z_{xy}| + \frac{|\overline{p_a p_b}|}{\sin\frac{\alpha}{2}}. \end{aligned}$$

From Lemma 4.3 we are ensured that $|\overline{q_a q_b}| \leq \epsilon|\overline{xy}|$ and $|\overline{p_a p_b}| \leq \epsilon|\overline{xy}|$. We can therefore make this substitution:

$$\begin{aligned} |z_{q_a p_b}| &\leq \frac{|\overline{xy}|}{\sin\frac{\alpha}{2}}\epsilon + |z_{xy}| + \frac{|\overline{xy}|}{\sin\frac{\alpha}{2}}\epsilon \\ &= \frac{2|\overline{xy}|}{\sin\frac{\alpha}{2}}\epsilon + |z_{xy}| \\ &\leq \frac{2|z_{xy}|}{\sin\frac{\alpha}{2}}\epsilon + |z_{xy}| \end{aligned}$$

$$= \left(1 + \frac{2\epsilon}{\sin\frac{\alpha}{2}}\right)|z_{xy}|.$$

Now $\|s_i'\| = w_j|z_{q_a q_b}|$ and $\|s_i\| = w_j|z_{xy}|$ so

$$\frac{\|s_i'\|}{w_j} \leq \left(1 + \frac{2\epsilon}{\sin\frac{\alpha}{2}}\right)\frac{\|s_i\|}{w_j}.$$

which leads to

$$\|s_i'\| \leq \left(1 + \frac{2\epsilon}{\sin\frac{\alpha}{2}}\right)\|s_i\|.$$

□

Claim 4.22 $\|s_i''\| \leq \frac{\epsilon}{\sin\frac{\alpha}{2}}\|s_i\|$.

Proof: By definition, s_i'' joins adjacent Steiner points on an edge of $\mathcal{P}$. Hence, from Lemma 4.3, it follows that $|s_i''| \leq \epsilon|s_i|$. It may be that s_i'' is impermissible, resulting in $\|s_i''\| \leq \frac{w_j}{\sin\frac{\alpha}{2}}|s_i''|$. Thus,

$$\|s_i''\| \leq \frac{w_j\epsilon}{\sin\frac{\alpha}{2}}|s_i|. \tag{4.10}$$

If s_i is regular or switchback, then $|s_i| \leq \frac{\|s_i\|}{w_j}$. If s_i is braking, then $|s_i| = \frac{\|s_i\|}{-\sin\varphi_i} \leq \frac{\|s_i\|}{w_j}$. Thus, we can say that $|s_i| \leq \frac{\|s_i\|}{w_j}$.

Making this substitution into Equation 4.10 results in $\|s_i''\| \leq \frac{w_j\epsilon}{\sin\frac{\alpha}{2}}\frac{\|s_i\|}{w_j} = \frac{\epsilon}{\sin\frac{\alpha}{2}}\|s_i\|$. □

Lemma 4.4 *Let $\Pi'(\sigma_{a-1}, \sigma_a)$ be a between-sphere subpath of $\Pi'(s,t)$ corresponding to an approximation of $\Pi(\sigma_{a-1}, \sigma_a)$ on a polyhedron $\mathcal{P}$ with minimum face weight w. Then $\|\Pi'(\sigma_{a-1}, \sigma_a)\| \leq (1 + \max(\frac{1}{\sin\frac{\alpha}{2}} + \frac{2}{w}, \frac{3}{\sin\frac{\alpha}{2}})\epsilon)\|\Pi(\sigma_{a-1}, \sigma_a)\|$.*

Proof: Let s'_i be a segment of $\Pi'(\sigma_{a-1}, \sigma_a)$ which approximates a segment s_i of $\Pi(\sigma_{a-1}, \sigma_a)$ passing through face f_j. By Claims 4.19 4.20 and 4.21 we have

$$\begin{aligned} \|s'_i\| &\leq \max(1+2\epsilon, 1+\frac{2\epsilon}{w_j}, 1+\frac{2\epsilon}{\sin\frac{\alpha}{2}})\|s_i\| \\ &= (1+2\epsilon\max(1, \frac{1}{w_j}, \frac{1}{\sin\frac{\alpha}{2}}))\|s_i\| \\ &\leq (1+2\epsilon\max(\frac{1}{w}, \frac{1}{\sin\frac{\alpha}{2}}))\|s_i\| \end{aligned}$$

We can charge the cost of each s''_i to s'_i. Therefore, we can say from Claim 4.22 that

$$\begin{aligned} \|s'_i\| + \|s''_i\| &\leq (1+2\epsilon\max(\frac{1}{w}, \frac{1}{\sin\frac{\alpha}{2}}))\|s_i\| + \frac{\epsilon}{\sin\frac{\alpha}{2}}\|s_i\| \\ &= (1+\epsilon(\frac{1}{\sin\frac{\alpha}{2}} + 2\max(\frac{1}{w}, \frac{1}{\sin\frac{\alpha}{2}})))\|s_i\| \\ &= (1+\max(\frac{1}{\sin\frac{\alpha}{2}} + \frac{2}{w}, \frac{3}{\sin\frac{\alpha}{2}})\epsilon)\|s_i\| \end{aligned}$$

Hence, each segment s'_i of $\Pi'(\sigma_{a-1}, \sigma_a)$ has cost at most $(1+\max(\frac{1}{\sin\frac{\alpha}{2}} + \frac{2}{w}, \frac{3}{\sin\frac{\alpha}{2}})\epsilon)\|s_i\|$ and $\|\Pi'(\sigma_{a-1}, \sigma_a)\| \leq (1+\max(\frac{1}{\sin\frac{\alpha}{2}} + \frac{2}{w}, \frac{3}{\sin\frac{\alpha}{2}})\epsilon)\|\Pi(\sigma_{a-1}, \sigma_a)\|$.

□

Claim 4.23 *Let $\Pi'(\sigma_{a-1}, \sigma_a)$ be a between-sphere subpath of $\Pi'(s,t)$ corresponding to an approximation of $\Pi(\sigma_{a-1}, \sigma_a)$. Then $|\Pi'(\sigma_a)| \leq \frac{2\epsilon}{1-2\epsilon}|\Pi(\sigma_{a-1}, \sigma_a)|$, where $0 < \epsilon < \frac{1}{2}$.*

Proof: Same as Lemma 3.11.

□

Claim 4.24 *Let $\Pi'(\sigma_{a-1}, \sigma_a)$ be a between-sphere subpath of $\Pi'(s,t)$ corresponding to an approximation of $\Pi(\sigma_{a-1}, \sigma_a)$ then $\|\Pi'(\sigma_a)\| \leq \left(\frac{2W\epsilon}{w(1-2\epsilon)\sin\frac{\alpha}{2}}\right)\|\Pi(\sigma_{a-1}, \sigma_a)\|$, where $0 < \epsilon < \frac{1}{2}$.*

Proof: Claim 4.23 ensures that $|\Pi'(\sigma_a)| \leq \frac{2\epsilon}{1-2\epsilon}|\Pi(\sigma_{a-1}, \sigma_a)|$. In the worst case, we can assume that segments of $\Pi'(\sigma_a)$ are impermissible and that $\Pi(\sigma_{a-1}, \sigma_a)$ uses minimum cost of w. Hence,

$$\begin{aligned} \|\Pi'(\sigma_a)\| &\leq \frac{W}{\sin\frac{\alpha}{2}}|\Pi'(\sigma_a)| \\ &\leq \frac{2W\epsilon}{(1-2\epsilon)\sin\frac{\alpha}{2}}|\Pi(\sigma_{a-1}, \sigma_a)| \\ &\leq \frac{2W\epsilon}{w(1-2\epsilon)\sin\frac{\alpha}{2}}\|\Pi(\sigma_{a-1}, \sigma_a)\|. \end{aligned}$$

□

We have made the assumption that $\Pi'(\sigma_a)$ consists of segments passing through faces that have weight W. Although this may be true in the worst case, we could use the maximum weight of any face adjacent to v_{σ_a}, which typically would be smaller than W. In addition, we have assumed that $\Pi'(\sigma_{a-1}, \sigma_a)$ traveled through faces with minimum weight. We could determine the smallest weight of any face through which $\Pi'(\sigma_{a-1}, \sigma_a)$ passes and use that in place of w. This would lead to a better bound.

Lemma 4.5 *If $\Pi(s, p)$ is a shortest anisotropic path in $\mathcal{P}$, where s is a vertex of $\mathcal{P}$ and p is a vertex of G then there exists an approximated path $\Pi'(s, p) \in G$ such that $\|\Pi'(s, p)\| \leq \left(1 + \epsilon\left(\frac{2W\epsilon}{w(1-2\epsilon)\sin\frac{\alpha}{2}} + \max\left(\frac{1}{\sin\frac{\alpha}{2}} + \frac{2}{w}, \frac{3}{\sin\frac{\alpha}{2}}\right)\right)\right)\|\Pi(s, p)\|$ where $0 < \epsilon < \frac{1}{2}$.*

Proof: Using the results of Claim 4.24 and Lemma 4.4, it can be shown that $\|\Pi'(\sigma_{a-1}, \sigma_a)\| + \|\Pi'(\sigma_a)\| \leq \left(1 + \epsilon\left(\frac{2W\epsilon}{w(1-2\epsilon)\sin\frac{\alpha}{2}} + \max\left(\frac{1}{\sin\frac{\alpha}{2}} + \frac{2}{w}, \frac{3}{\sin\frac{\alpha}{2}}\right)\right)\right)\|\Pi(\sigma_{a-1}, \sigma_a)\|$. This essentially "charges" the length of an inside-sphere subpath to a between-sphere subpath. The union of all such subpaths form $\Pi'(s, p)$. This allows us to approximate $\Pi'(s, p)$ within the bound of $1 + \epsilon\left(\frac{2W\epsilon}{w(1-2\epsilon)\sin\frac{\alpha}{2}} + \max\left(\frac{1}{\sin\frac{\alpha}{2}} + \frac{2}{w}, \frac{3}{\sin\frac{\alpha}{2}}\right)\right)$ times

the total cost of all the between-sphere subpaths of $\Pi(s,p)$. Since $\Pi(s,p)$ has cost at least that of its between-sphere subpaths, the lemma holds true.

□

Theorem 4.2 *If $\Pi(s,t)$ is a shortest anisotropic path on a polyhedral surface $\mathcal{P}$,where s and t are vertices of $\mathcal{P}$ then there exists an approximated path $\Pi'(s,t)$ for some $0 < \epsilon < \frac{1}{2}$ such that*

$$\|\Pi'(s,t)\| \leq \left(1+\epsilon\left(\frac{2W\epsilon}{w(1-2\epsilon)\sin\frac{\alpha}{2}} + \max\left(\frac{1}{\sin\frac{\alpha}{2}} + \frac{2}{w}, \frac{3}{\sin\frac{\alpha}{2}}\right)\right)\right)\|\Pi(s,t)\|.$$

Proof: The proof follows from Lemma 4.5 since t is also a vertex of G. Note that if there is information about α and w, then these expressions can be simplified with an upper bound. For example, if $\sin\frac{\alpha}{2} \leq w$ then the following upper bound can be used: $\|\Pi'(s,t)\| \leq \left(1+\epsilon\left(\frac{2W\epsilon+3w}{(1-2\epsilon)\sin^2\frac{\alpha}{2}}\right)\right)\|\Pi(s,t)\|$. Similarly, if $\sin\frac{\alpha}{2} > w$ then we can use this upper bound: $\|\Pi'(s,t)\| \leq \left(1+\epsilon\left(\frac{2W\epsilon+3\sin\frac{\alpha}{2}}{(1-2\epsilon)w^2}\right)\right)\|\Pi(s,t)\|$.

□

4.5 Experimental Results

Results of our experiments in Chapter 2 showed that the algorithm is practical, requiring only a few Steiner points to be added per edge in order to obtain good path accuracy and algorithm running time. In this chapter, we gave a similar algorithm which differed mainly in the weights assigned to each edge of the graph. In order to validate its practicality, we implemented the algorithm and performed tests with various terrains. We give here an explanation of these tests and their results.

The implementation is based on the code of the algorithm in Chapter 2. However, with this new implementation, the direction of each edge of the graph is important. Each graph edge is assigned two weights, one for upward traversal and one for downward traversal. Costs are assigned according to the direction range in which the edge

is directed (i.e., impermissible, braking or regular). Dijkstra's algorithm is used as before, but now without the A^* improvement. The estimated cost to the destination as used in the implementation of the algorithm in Chapter 2 was set to the 3D Euclidean distance between the point at hand and the destination. This was always an under-estimate, which was necessary to ensure correct results. However, since the cost of edge with the anisotropic implementation may be less than one (i.e., they are based on cosine operations), this Euclidean distance estimate may be an over-estimate and so invalid results may be obtained.

The test procedures were similar to those used in the experiments of Chapter 2. The purpose of the tests were to determine whether or not similar convergence in accuracy can be obtained as the number of Steiner points added is increased and also to investigate the effect on run time as these points are added. The tests were performed on four of the terrains that were used in Chapter 2: the 10,082 face terrain, the 5,000 face terrain with random heights, the Madagascar terrain and the North America terrain.

The tests were run by varying the number of Steiner points per edge from 0 to 16. The tests were also performed for various impermissibility angles. That is, different critical impermissibility angles of 5, 10, 15, 30, 45, 60 and 90 degrees were used throughout the tests. The coefficient of static friction for each face was set to one and remained a constant throughout the experiments. Also, the sideslope overturn ranges were not considered throughout the tests. As with our previous work, we ran tests for 100 pairs of randomly selected source and destination vertices and computed the average path cost and average computation time for all the tests.

The graphs of Figure 4.20 show the convergence behaviour of the path accuracy for the different numbers of Steiner points and impermissibility angles. The graphs show that the path cost decreases with an increase in the number of Steiner points per

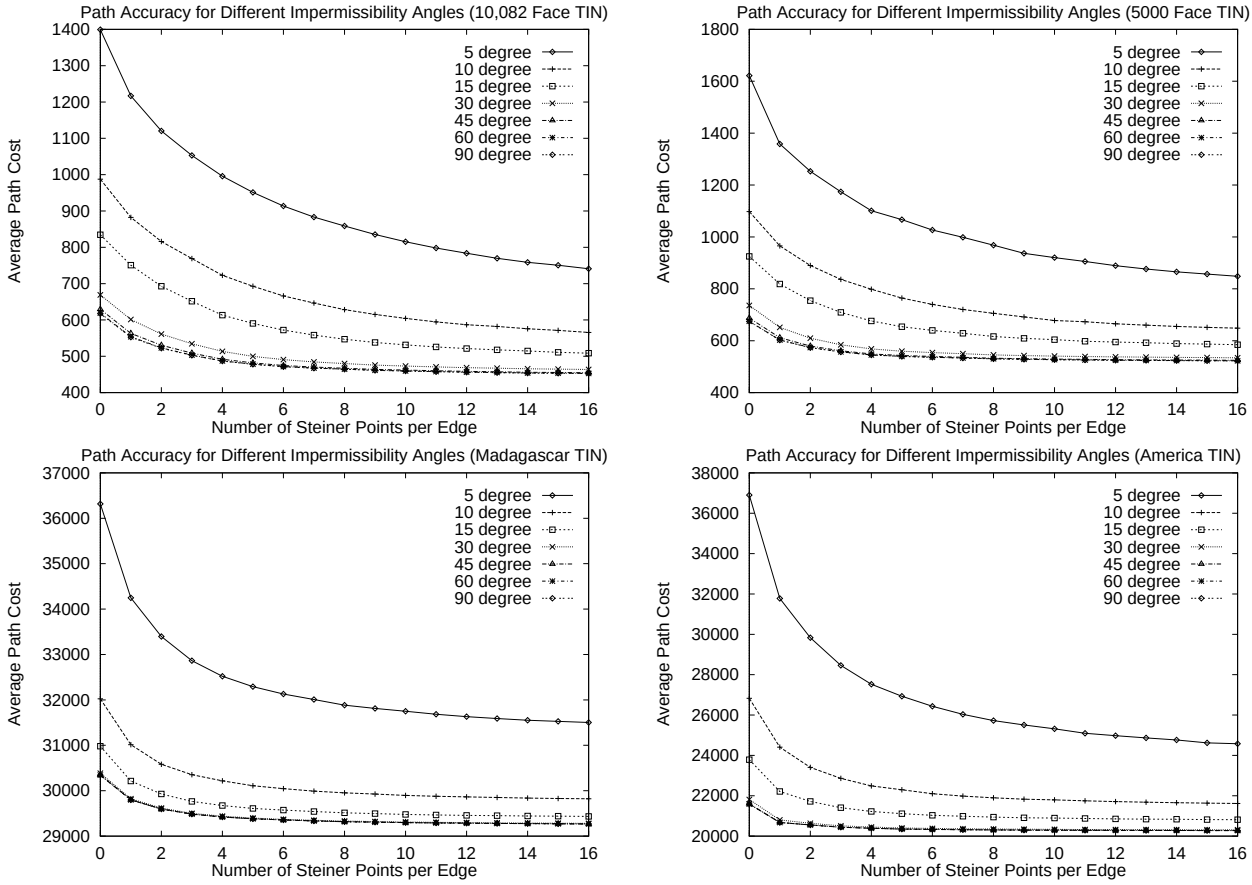

Figure 4.20: Graphs showing average path cost for four terrains.

edge of the terrain. Notice however that in some cases, more than six Steiner points are necessary before the convergence occurs. Recall that the 90 degree tests have no impermissibility angles and so the results there are comparable to our weighted isotropic shortest path results of Chapter 2. As the impermissibility angle decreases, we notice that the path cost also increases. This is because a decrease in impermissibility angle causes a larger impermissibility range and hence more switchback paths are required. Also noticeable is the slower convergence with the smaller impermissibility angles. The switchback paths in a face do not make use of the Steiner points. Therefore, by increasing the number of Steiner points on an edge, this does nothing to decrease the cost of a particular switchback path within a face. With the larger impermissibility ranges, the path produced by the algorithm has more switchback paths within it and so a high percentage of path segments are unable to make use of the increase in the number of Steiner points.

The graphs of Figure 4.21 show the average computation time for computing the shortest path costs for the 100 pairs of source/destination vertices. Notice that the graphs exhibit quadratic growth as the number of Steiner points is increased. Also notice that there is no noticeable difference in time between the tests with small or large impermissibility ranges.

The results of our experiments indicate that the algorithm is most practical when impermissibility angles are large. With the larger ranges, only a few Steiner points (i.e., six) are necessary to achieve accurate paths. According to the results of Rula and Nuttall [109], most terrain becomes untraversable when the inclination angle exceeds 30 degrees. On the other hand, 5 degree inclinations are commonly traversable by even the simplest vehicles (i.e., bicycles) without requiring switchbacks. The more realistic values of inclinations that could cause switchbacks are likely between 10 and 30 degrees. By looking at the accuracy graphs of Figure 4.20, it can be seen that

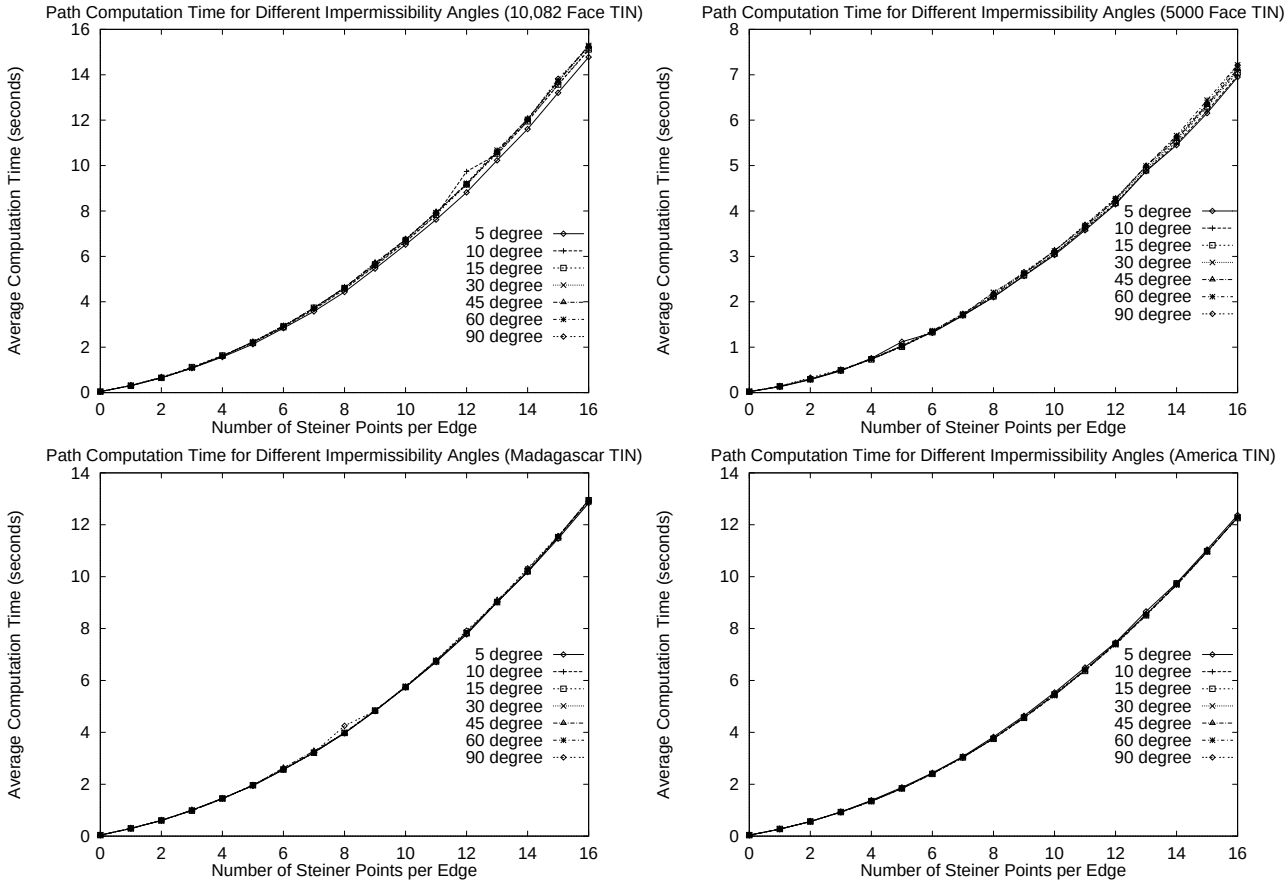

Figure 4.21: Graphs showing average path computation time for four terrains.

the convergence is better in this range than the 5 degree angle. With additional experimental testing, it is likely that the optimal number of Steiner points to be used may be a function of the impermissibility angles, which in turn depend on vehicle-specific information.

Chapter 5

A Parallel Shortest Path Simulation

The main objective of parallelizing a shortest path algorithm is to reduce the algorithm's runtime significantly. This time saving would allow larger data sets to be used and more complex calculations can be performed, hence achieving often better solutions (i.e., higher accuracy) in less time than would be possible sequentially. For time-critical computations, parallelism may provide the only solution. In addition, with sequential algorithms, it is often the case that the data is too large to be stored in internal memory. The use of parallelism can allow the data to be distributed across multiple processors so that the storage requirements per processor is reduced. Still, it may be the case that the portion of the data assigned to a processor is too large to be loaded into memory and we may be forced to use other strategies such as external memory algorithms. Unfortunately, the use of external memory often requires more complex data structures and modifications to the algorithm are necessarily required.

In this chapter, we present a parallelization of the algorithms given in the previous chapters. That is, we give a parallel algorithm that computes shortest paths (unweighted/weighted/anisotropic) between pairs of query points on a (possibly large) polyhedral or terrain surface. The algorithm also applies to the single-source problem. In essence, the algorithm here attempts to maintain the simplicity of our sequential algorithms while providing a decrease in the running time and allowing larger terrains to be used. The problem is again reduced to solving the shortest path between vertices of a graph. Although algorithms already exist for computing shortest paths in graphs in parallel, these algorithms often assume a large number of processors (e.g., $O(n)$ [105]) and sometimes assume constraints on the data such as being evenly distributed [1][18][118] and/or being sparse [1]. In addition, some of this research pertains to the all-pairs shortest path problem which is not studied here. More recent work has aimed at providing implementations of distributed algorithms that obtain "reasonable" performance and in analyzing the factors that affect the performance [1][18][118]. We illustrate how our algorithm's performance varies with some of these factors as well.

We use a general spatial indexing storage structure (Multidimensional Fixed Partition (MFP) tree of Nussbaum [97]). The structure implicitly achieves a form of load balancing as well as allows the processor idle time to be reduced in cases where the terrain has dense clusters of data. In addition, the structure can also be used for other types of propagation applications besides shortest path such as visibility, weighted Voronoi diagrams etc. This flexibility motivates the use of such a general data structure since it allows our application to be incorporated into our parallel GIS project ATLANTIS.

This chapter is organized as follows. Section 5.1 presents preliminaries by beginning with a discussion of various factors that can affect the performance of a parallel

algorithm. This includes a brief discussion of previous work on distributed shortest paths and how this research investigates different performance factors. Section 5.2 describes our parallel algorithm, beginning with a description of the preprocessing step and then describing its execution for given queries. Our experimental results are then given in section 5.3, beginning first with a description of the test data and testing procedures. We give results for single-level partitioning in subsection 5.3.2. In that section we also analyze the effects of varying the computational intensity of the cost function, the relative location of the source/target pairs, the number of Steiner points and the amount of over-processing is also investigated here. Section 5.3.3 then describes our tests and results with multiple levels of partitioning for the one-to-all and few-to-all shortest path problems.

5.1 Preliminaries

Before describing our algorithm, it is important to explain the many factors that can affect the performance of any parallel algorithm. We discuss these factors here and how they relate to the previous work.

One of the main characteristics that distinguish different parallel algorithms from one another is the model of computation. Our algorithm uses the MIMD (Multiple Instruction Multiple Data) model with distributed memory. That is, each processor has its own local memory and performs computations on its own independent set of data. Also, communication between processors is asynchronous.

Distributed algorithms typically partition their data and distribute it among the processors. Each processor keeps a portion (i.e., a partition) of the data and must communicate with other processors if it requires computations that use data which resides on another processor. Two adjacent partitions are said to *share a boundary*

if any of the data (i.e., vertices, edges or faces) belongs to both partitions. There are various methods of data partitioning which provide different characteristics that pertain to the partition size, the spatial relationship within a partition, the amount of data duplication and in the simplicity of implementation. The way in which this partitioning is done can significantly affect the running time of the application.

Load balancing is the term used to indicate how the data is distributed among processors with respect to how much work is to be done. The size of the individual partitions is often used as an indication of the amount of work to be done. Intuitively, load balancing is a desirable characteristic of partitioning schemes and it plays a vital role in reducing the idle time of individual processors.

The *spatial relationship* of data is often of interest as well during partitioning. This relationship often pertains to topological and/or geometric properties of the data itself. It is often desirable to partition the data into "groups" such that the data in one group has small diameter and is connected. For example, assume that n faces of a polyhedron are distributed to p processors such that each processor gets n/p faces. If the faces are distributed randomly, there may be no spatial relationship among data within a processor or between data of adjacent processors. That is, a particular processor may end up with a set of n/p disjoint faces. Implementation of an algorithm that requires traversal between sleeves of faces (e.g., shortest path on terrains), would be inefficient with such a distribution since every time a face is processed, the next face (physically adjacent) to be processed will typically reside on a different processor. This will incur unnecessary communication overhead at each processing step. It would be more advantageous to distribute connected groups of faces to a processor so that a significant amount of processing can be accomplished before requiring communication with another processor.

Let p_1 and p_2 be two processors that share a vertical or horizontal partition boundary. Each time a message is sent from p_1 to p_2 this is called a *processor hop*. For example, in a 4x4 mesh, adjacent partitions are one hop away where as the furthest partitions are six hops away. It is typically desirable to minimize the number of processor hops for each message being sent or the total number of such hops. This will help to reduce message traffic in the system. A *bottleneck* can form if too many messages are routed through the same processor. Thus, a single processor might be overwhelmed with messages so as to slow down the entire intercommunication network of the parallel machine. It is more efficient to spread out the message sending so that no bottlenecks are formed.

An *asynchronous* algorithm allows multiple processors to perform multiple independent computations simultaneously. Each processor computes at its own rate and communicates with other processors only when communication is necessary. Processors communicate with others at different times and rates. A *synchronous* algorithm however, requires each processor to perform some computations and then synchronize with all other processors before continuing. A *step* is typically the notation used to represent one phase of computation and communication. Computation by steps can be advantageous for those shortest path algorithms that require a global minimum cost to be maintained throughout the execution of the algorithm. It is often the case, however, that synchronization can cause processors to sit idle while others complete their step.

The actual runtime of a parallel algorithm is typically broken down into three types. The *computation time* is that time (number of seconds or milliseconds) which the processor spends doing computations. The *communication time* is the time spent sending or receiving data from other processors. The *idle time* is the time that the processor spends outside of computation and communication (i.e., waiting with no

work to do).

The runtime performance is typically a good indicator of the usefulness of a parallel algorithm and it is often measured in terms of speedup. Let the *speedup* of a parallel algorithm with p processors be the time it takes to run the best sequential algorithm divided by the time it takes to run the algorithm with p processors. This is the true notion of speedup although it is slightly "unfair" from the parallel algorithm's point of view. This is because with a parallel algorithm, there is always an overhead due to communication that cannot be avoided. Even if the parallel algorithm is executed with $p = 1$, this overhead is still noticeable although there is no communication since the code that is often in place must still be evaluated. To help overcome this overhead factor, speedup is sometimes measured as the time it takes to run the parallel algorithm using one processor divided by the time it takes to run the algorithm with p processors. This measure of speedup is used in the literature (see e.g., [1], [18]); we also use it when evaluating our experiments. A *slowdown* is the term used when a parallel algorithm performs slower on multiple processors than on a single processor. Factors that most often contribute to slowdowns are small data sets, high communication and too much synchronization between processors.

The *efficiency* of a parallel algorithm is often measured as $\frac{speedup}{p}$ when p processors are used. A program is said to exhibit *linear speedup* if the speedup roughly equals the number of processors. If linear speedup is maintained as the number of processor increases, the algorithm is said to be *scalable*. Adamson and Tick [1] state that no known one-to-all graph shortest path algorithms are scalable. Having said this, the aim of most implementations is to achieve reasonable efficiency. An efficiency of between 25% to 60% is common.

5.1.1 Performance Factors and Previous Work

A parallel algorithm's runtime performance depends on many factors that are due to the algorithm, data, machine and implementation. Unfortunately, there are so many factors that it becomes difficult to make a thorough comparison and even more difficult to form any conclusions on how each factor (in seclusion) affects the performance. Much of the research has been geared towards analyzing the effects of one or two of these factors in seclusion. That is, various assumptions are often made on the other factors so that meaningful conclusions can be drawn pertaining to the factors under analysis. For example, one study may examine the effects of algorithmic factors while making assumptions on the data and machine related factors such as assuming efficient load balancing of data on a specific machine architecture. Another study may assume a large number of processors (i.e., $O(n)$) and a very specific architecture while focusing only on different partitioning strategies. As a result of these variations, it is difficult to compare results of one study with those from another. We will discuss some of the important factors in this section as they have appeared in the literature.

5.1.1.1 Algorithm Related Factors

Much of the published results on parallel distributed shortest paths are aimed at the one-to-all (also few-to-all) shortest path problem (see [1], [18], [67], [66], [105], [118]). A study by Gallo and Pallottino [49] has shown that the most efficient label-correcting algorithms are faster than the most efficient label-setting methods in the serial case for the one-to-all problem when small, sparse graphs are used. Note that graphs with an average node degree of six or less are typically considered to be sparse. Bertsekas et al. [18] have suggested that the one-to-all label-correcting algorithms for sparse graphs also provides an advantage in the parallel shared memory setting. They do

state however that when only a few destinations are used (i.e., one-to-one), then the label-setting algorithms are better suited (see also [60],[102]).

In contrast, Hribar et al. [64] have shown that the label-correcting algorithms have inconsistent performance for their networks. They state that there is no one shortest path algorithm that is best for all shortest path problems, nor for problems with similar data set sizes. They conclude that generalizations cannot be made about the best algorithm for all networks. They do however, attempt to give an indicator where the label-setting algorithms outperform label-correcting algorithms which is based on the expected number of iterations per node. In a later study by Hribar et al. [66], they give experiments that demonstrate that label-setting algorithms perform best for distributed memory parallel machines when large data sets (16k - 66k vertices) are used. They experimented with five transportation networks corresponding to actual traffic networks as well as random grid networks.

Besides the categories of label-correcting and label-setting, there are variations of each that differ in the number and size of queues, and in the order that items are removed. Bertsekas et al. [18] experimented with single vs. multiple queues for the label-correcting algorithms in the shared memory setting. Their multiple queue version assigns a local queue to each processor, however, the data in the queue has no particular relationship (such as spatial) with that processor. They show that the multiple queue strategy is better, due to the reduction in queue contention. Their experiments also investigated synchronous vs. asynchronous algorithms and their results show that the asynchronous algorithms always outperformed the synchronous ones. Träff [118] has compared a synchronous strategy with an asynchronous strategy using a label-setting algorithm and he indicates that synchronous strategies will perform poorly in systems where there the cost of communication is high.

Adamson and Tick [1] compared five algorithms that use a single queue for all

processors which is either sorted or unsorted. They compare partitioned and non-partitioned algorithms. The partitioned algorithms assign vertices to each processor according to some modulus mapping function, whereas the unpartitioned version assigns vertices to processors arbitrarily. They show that the unsorted queue works best for the unpartitioned data with a small number of processors and that the sorted queues are better for the partitioned data when a higher number of processors are used. They have two varying parameters of granularity and eagerness which specify the size of the partitions and frequency of queue extraction requests, respectively. They were able to obtain a best efficiency of 44% with 16 processors.

The number and frequency of communication steps that each processor performs throughout the algorithm execution is also an important performance issue. Some algorithms (e.g., [66]) allow each processor to empty out their local queues as a common step and then all boundary node information is communicated to all adjacent processors. A different strategy would be to communicate shared information as soon as it is available (e.g., [118]). The advantage of the first strategy is that only a few large messages need to be communicated, whereas the second strategy would have many more smaller messages being communicated which could cause runtime delays. The use of few large messages is a standard approach when the BSP model of computation is used. However, when data is unbalanced slightly among processors, the first approach may have some processors sitting idle while other are still emptying their queue. In addition, much of the work done by a processor may necessarily be discarded as a result of new information coming from its adjacent processor after its queue has been emptied. A further disadvantage of the first approach is that it requires synchronization which can lead to a performance decrease as shown by Träff [118].

One algorithmic factor in parallel shortest path computations that is often overlooked is that of *termination detection frequency*. This defines the amount of times that the algorithms checks for termination. With a high frequency detection, a processor will spend much of its time checking whether or not the algorithm has terminated, whereas a low detection frequency may check less but processors may do more computational work than is necessary. Hribar and Taylor [67] provided research that analyzed the impact of the detection frequency on the performance of a synchronous label-correcting shortest path algorithm. Their analysis indicates that the optimal frequency for detecting termination depends on the number of processors and the size of the problem. They state that high detection frequencies are best when a large number of processors are used and low frequency otherwise. To validate their claim, they tested with 4 variations of termination detection which varied with respect to the time interval between communication steps as well as the frequency of synchronization.

The final algorithm-related factor that we will discuss is that of the cost function. The cost function used in the shortest path calculation can also affect the performance. If a simple cost function is used such as the Euclidean metric, then little time is spent on computation and a higher percentage of time is spent on communication. When a more computationally-intensive cost function is used, such as anisotropic metrics used in Chapter 4, then the percentage of compute time increases, which often leads to more efficient use of the processors. Most of the existing research has focussed on weighted graphs with no discussion as to how the weights were obtained. To our knowledge, there has not been any papers that analyzed the effects of the cost function.

5.1.1.2 Data Related Factors

We now turn our discussion towards data related factors. Perhaps the largest factor in determining the runtime performance of a parallel shortest path algorithm is based on characteristics of the data itself. Clearly, the amount of data is of paramount importance since everything slows down as larger data sets are used. All of the researchers have realized this and always give a comparison of different data set sizes so as to get a feel for how the algorithm will scale with large amounts of data compared to small amounts of data. For graph data (e.g., transportation networks) the size of the graph (with respect to the number of vertices and edges) can make a significant impact on the runtime. Adamson and Tick [1] state that the amount of parallelism is dependent on the average degree (connectivity) of the graph. Most of the research has concentrated on sparse graphs (i.e., average degree of about six). In practice, it is important to choose the "right" algorithm that has a good performance for the type of data being used. Sometimes however, it is not so easy to classify a graph as being sparse since the definition of sparseness implies some maximum or average node degree which depends on some fixed constant. For example, in our algorithm, we compute complete graphs on each face which is classified as a high-density graph, but the interconnections of these face graphs is sparse (if the original polyhedral data is sparse). It could be argued that since our algorithm has been shown to be quite practical with only six Steiner points per edge, then the entire graph is sparse since only a constant (up to 28) number of edges are connected to any Steiner point node (assuming that terrain vertices have degree less than 28 as well).

In addition to the amount of data present, the distribution of the data also plays a key role in the performance. There has been quite a bit of work pertaining to investigating different approaches to data partitioning. These approaches differ in the number of partitions made, the sizes of the partitions, the amount of shared data

between adjacent processors and the topological order of partitions (i.e., if they are recursively partitioned). Due to the vast nature of the data partitioning problem, we do not describe the various techniques here. Hribar et al. [65] state that determining good decompositions is crucial for all parallel applications. For their algorithm, as applied to transportation networks, they show that the best decomposition minimizes the number of connected components, diameter and number of partition boundaries, but maximizes the number of boundary nodes. Hribar et al. [66] have shown for transportation network graphs that distributing the data among the processors has roughly half the execution time as replicating the graph on all processors. Even with distributed data, if the amount of work (i.e., data) given to one processor is significantly less than that of another processor, then that processor may sit idle while the other processor is kept busy with computations. Basically, the more time a processor spends idle, the higher the loss of parallelism.

The relative locations of sources and targets is another factor of significant importance. If the source and target points both lie on one processor, the algorithm may not use more than one processor and the parallelism of the problem is lost. If they are far away on different processors, then more processors get involved in the computation and some speedup should be noticeable. Of course, when they lie in different processors, there is some additional communication required. There has not been much research in determining the effects of source/target locations. Instead, most of the previous experimental work has concentrated on the one-to-all and all-to-all problems.

5.1.1.3 Machine Related Factors

One of the reasons why it is often difficult to compare and contrast results from different researchers is that the machine architecture is never quite the same. The

machines differ in the number of nodes (e.g., fine or coarse grained), the type and speed of processors (e.g., Power PC, Sparc, Pentium, Pentium II, Transputer, i860), the interconnection topology (e.g., ring, mesh, hypercube, network of workstations), the interconnection strategy (e.g., ethernet, crossbar, dedicated links), the amount of internal and virtual memory per processor (including levels of caches) and more importantly, the accessibility of memory to each processor (i.e., shared or distributed memory). All of these differences can significantly affect the runtime performance of any parallel algorithm.

Most experimental studies perform tests on a single machine that is either fine or coarse grained with a single type of processor, interconnection topology and strategy. This allows a reasonable comparison of algorithms and partitioning techniques. However, the internal and virtual memory factors make it difficult to determine which algorithm or partitioning technique is better than others. For instance, one algorithm may outperform another for some fixed size of internal memory, say M. If M is then increased to $2M$, the other algorithm may perform better. To understand this, consider two algorithms, one being an internal memory based algorithm A_1 and another being external-memory based A_2. When M is too small, both algorithms must access the disk. It is likely that A_2 will outperform A_1 in this case since A_2 probably has special data structures for disk access and care has been taken to limit the amount of disk reads/writes. If however, M is increased, then the problem may fit into internal memory and hence neither algorithm may require disk access. In this case, the additional data structures of A_2 would likely provide a runtime overhead and hence A_1 may be quicker. Of course, there are other factors at play here such as data partitioning and storage space requirements for each algorithm. Nevertheless, it can make a dramatic difference in the performance if virtual memory is never used vs. always used.

One of the significant differences between machines is that of the accessibility of the memory to each processor. For example, some algorithms are meant to be run on a shared memory machine (see [1], [2], [18]) in which all processors have access to some common memory. This is useful if all processors need to access some common data structure such as a priority queue which is necessary for label setting algorithms. The other choice is to have a distributed memory machine (see [118], [65], [67]) in which each processor has its own local memory. Any processor requiring access of memory or data from another processor must do so through message-passing.

5.1.1.4 Implementation Related Factors

The last set of factors to be examined are implementation-specific factors. For example, two programs implemented by different people are likely to have variations in styles and efficiency, even at the geometric primitive level such as calculating the slope of a line or intersecting two line segments. It would therefore be advantageous to promote sharing between programmers with libraries such as LEDA, or CGAL. There are other issues such as numerical stability and correctness that can be related to performance as well.

In addition to the programmers' implementation discrepancies, there are efficient and inefficient implementations of libraries from vendors. For message passing strategies alone, there are choices as to which libraries to use such as Trollius, MPI, MPICH and PVM. In addition, there are different vendors that produce versions with their own characteristics which can lead to versions which are slower than others. Hence, the use of software from one vendor to another can lead to significant fluctuations in runtime performance. Moreover, the cost of communication versus cost of computation can be affected by the choice of libraries. We have chosen MPI for our implementation and have not used any specific computational libraries such as LEDA

or CGAL, so as to keep the ATLANTIS system code simpler and less dependent on external libraries.

5.2 The Parallel Algorithm

Our sequential algorithms (Chapters 2, 3 and 4) allow a preprocessing step which is to build a graph. Queries are then answered by running Dijkstra's shortest path algorithm. The parallel algorithm here follows the same strategy. Essentially, each processor executes its own instance of Dijkstra's algorithm on its own graph data (i.e., each keeps its own priority queue). Since Hribar et al. [66] state that label-setting algorithms perform best for distributed memory parallel machines, we have chosen this variation as well. Like Hribar et al. [66], our algorithm is somewhat a mixture of both label-setting and label-correcting in that each processor implements a label-setting strategy, however the removal of a node from the queue is based solely on local information. Hence, when a processor's queue becomes empty, it does not mean that the global solution is available. The global information is implicitly stored across the processors and interprocessor communication is required to update the local information at each processor. Unlike Hribar et al. [66] however, we have decided to use asynchronous communication (as done by Träff [118]) which will avoid the performance decrease inherent with the synchronization steps.

During propagation, the active border for a given processor will often reach its partition boundary. At this time, the cost of some node v_i that is shared with its neighbour partition is updated. This adjacent processor obtains the updated cost for v_i and continues processing within its partition. Eventually, each processor will have exhausted their priority queues and the algorithm halts. At this point a target query can be presented and the resulting path can be computed. In fact, the implementation

provides a mechanism for detecting when the target(s) has been reached and allows the simulation to halt then. The simulation itself can be broken down into three steps: preprocessing, running of Dijkstra's algorithm, tracing back the path. Each of these is explained in more detail in the subsections to follow.

The algorithm is described for the special case in which there is a single source and a single target. The following variations are also possible with the algorithm as well:

- Single Source / Multiple Targets: (ONE-TO-ALL or ONE-TO-ANY)
- Multiple Sources / Single Target: (ALL-TO-ONE or ANY-TO-ONE)
- Multiple Sources / Multiple Targets: (ANY-TO-ANY or ANY-TO-ALL)

These variations require only a simple modification of the algorithm to store arrays of sources or targets to be used. Some variations, however, would require much more effort. For example, ALL-TO-ANY or ALL-TO-ALL operations would require one priority queue for each source. This may not be possible to store in memory if too many sources are used. Instead, the application provides the capability of running multiple *sessions* in which each session represents a variation mentioned in the list above. Note that the ALL-TO-ANY and ALL-TO-ALL can be solved using multiple ONE-TO-ALL and ONE-TO-ANY sessions, although this is a brute force solution and more efficient solutions are possible.

5.2.1 Preprocessing

We describe here our partitioning phase of the algorithm which uses the *Multidimensional Fixed Partition (MFP) tree* scheme of Nussbaum [97]. The MFP scheme

divides the data spatially, takes care of clustering problems and provides a natural mapping to processors. This tree is similar to the well known quadtree and octree data structures in that it divides the data recursively according to spatial characteristics. It differs, however in the shape of the partitioning grid (i.e., it generates decompositions according to the number of available processors as opposed to a fixed dimensional grid). The tree represents a decomposition of the data into levels, where lower levels represent a recursive (refined) partitioning of the level above it. We first describe the single-level partitioning for the MFP and then discuss the multi-level partitioning. The entire preprocessing step is done separately from the simulation. It is assumed that some other application has partitioned the data and this data resides on the local disk of the proper processor.

During preprocessing, the terrain $\mathcal{P}$ with n triangular faces is partitioned among $p = R \times C$ processors. We say that processor p_{rc} is in row r, column c of the mesh (where p_{00} is the bottom left). Each level of decomposition in the MFP tree consists of p sub-partitions. The top level represents all of $\mathcal{P}$ and is denoted as level 0. Each further level of partitioning is created by dividing a partition from the previous level into an $R \times C$ grid of cells whose boundaries are defined by horizontal and vertical cut lines. More formally, let the smallest enclosing bounding box of $\mathcal{P}$ be defined by (x_{min}, y_{min}) and (x_{max}, y_{max}). Divide $\mathcal{P}$ into an $R \times C$ grid such that each grid cell defines an equal area of the terrain defined by $((x_{max} - x_{min})/C) \times ((y_{max} - y_{min})/R)$. For example, Figure 5.1 shows how two terrains are divided into 3 x 3 grids when 9 processors are used. Each face of $\mathcal{P}$ is assigned to each processor whose corresponding grid cell it is contained in or intersects with. Using this technique, faces of $\mathcal{P}$ that intersect the boundary of two adjacent grid cells are shared between the two partitions as shown in Figure 5.2. We call the shared faces *border faces* or *border triangles*.

Two other approaches could have been taken here. The faces that intersect the

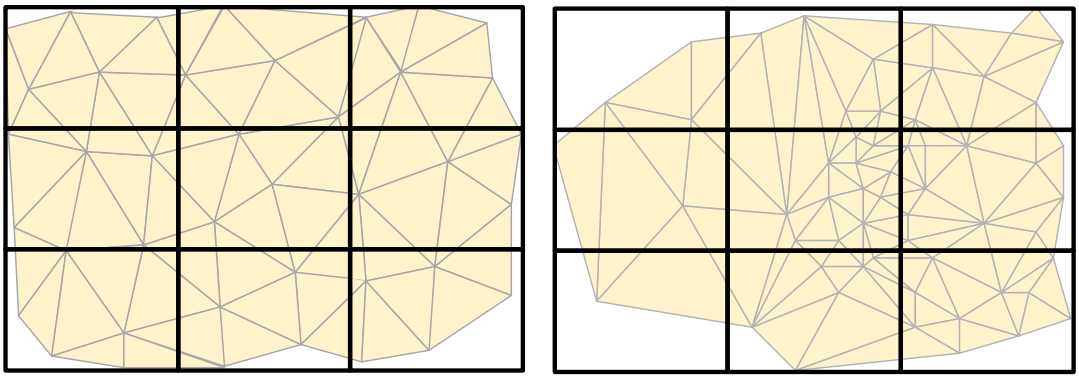

Figure 5.1: Dividing non-clustered and clustered terrains into equal-area 3 x 3 grid.

boundaries could have been split into two pieces. This however, could generate additional faces and require re-triangulation of existing faces since the cut line may have generated quadrilaterals. In addition, the newly produced faces and edges can cause problems with some algorithms that rely on geometric properties of $\mathcal{P}$. For instance, our shortest path algorithm must add Steiner points along the cut line so as to provide a connection between the two partitions. This increases the size of the graph, but more importantly, it can change the "bending" characteristics of the path. The other approach that could have been taken is that of separating the faces along edges of $\mathcal{P}$ instead of along the cut lines. This is done by finding a polygonal chain of edges in $\mathcal{P}$ that would partition $\mathcal{P}$ into two. All faces to one side of this chain are separated from those on the other side. As a result, the partitions share edges in common along the chain. The advantage of this technique is that no faces are duplicated and there is a clean separation in that no geometric properties are lost. The disadvantage of course is that it is not trivial to find a good separator (i.e., polygonal chain) and once found, the computations required to determine which side of the chain a face lies is more complex since the grid cells are no longer defined by rectangles. (In a separate study carried out by the Paradigm group [121], separator schemes are being investigated.)

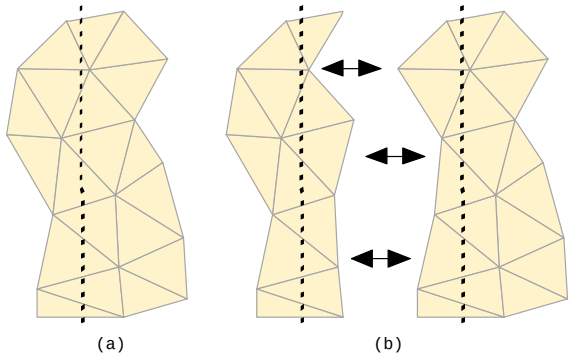

Figure 5.2: a) A set of faces to be shared along a cut line. (b) After cutting, faces are shared between the two partitions.

This single-level partitioning of the MFP scheme (as well as most other partitioning techniques) are susceptible to large processor idle times when used with a parallel shortest path algorithm. Consider the example of Figure 5.3 in which a wavefront propagates across nine processors beginning at processor 6. A processor does work only when a portion of the active border lies within its region of $\mathcal{P}$. While the active border expands within processor 6, all other processors sit idle. Eventually, the active border will reach processors 3, 4 and 7. Still, many processors sit idle. It is easily seen that processor 2 remains idle until the propagation is near completion; thus it is barely used and is therefore wasted. Moreover, once the active border has left the region assigned to processor 6, it also becomes idle while the other processors take over.

One way of avoiding this idling problem is to assign to each processor, more than one connected portion of the terrain. That is, we can *over-partition* $\mathcal{P}$ which involves a recursive decomposition of the partitions. The result is that each processor will have scattered portions of $\mathcal{P}$. It is therefore more likely that the active border will be processed by many processors throughout its growth. Figure 5.4 shows how the

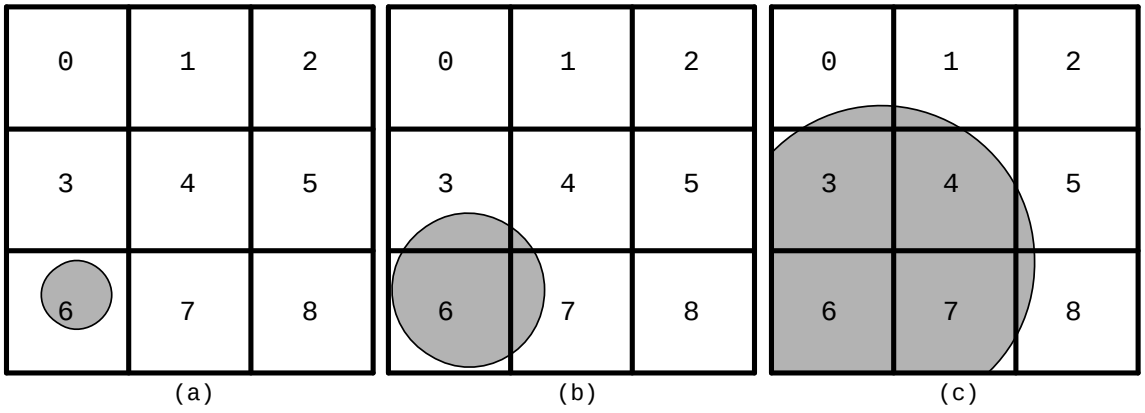

Figure 5.3: Expansion of the active border showing that processors may sit idle during a shortest path computation.

MFP tree re-partitions each partition one level further. We can see now that in the example, the active border expands quickly to allow four processors to participate in the calculations. Soon afterwards, all processors get involved and are continually used throughout the propagation.

Consider dividing $\mathcal{P}$ repeatedly to obtain finer and finer grid sizes. To produce multiple levels of partitioning, it is necessary to choose a threshold which pertains to the maximum number of vertices that a single partition can have before being re-partitioned. If the number of vertices in a partition (called the *tile size* exceeds the threshold, it is re-partitioned further. Deciding on the "best" threshold depends on factors such as terrain size, number of processors and the degree of clusterization. For example, Figures 5.3 and 5.4 depict 1-level and 2-levels of decomposition, respectively. If we make the grid too fine, then the regions assigned to each processor are too small to work with and most of the processor time is spent doing interprocessor communication. On the other extreme, if we make the grid too coarse, a processor may have too much work to do and very little communication with others. This reduces

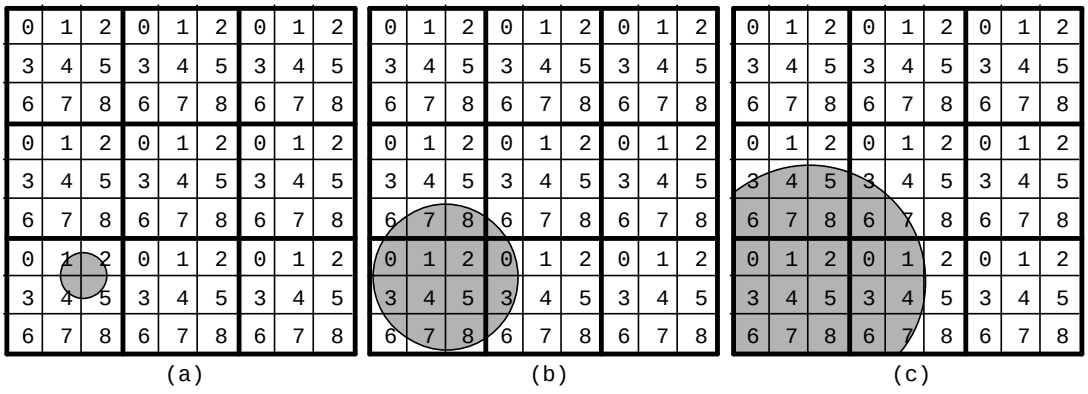

Figure 5.4: Over-partitioning allows all processors to get involved quickly in the computations and remain involved longer, thus reducing idle time.

the parallelism and ultimately results in a running time equivalent to a sequential algorithm. There is a specific number of decomposition levels that permits a nice tradeoff between processor work and communication. For typical terrain data, we expect that no more than three or four levels of recursion will be required and that two to three levels are typical for achieving good results for many applications.

Assuming that we determine this number, we still have a problem with data clustering. That is, if we partition the entire terrain equally, some groups of data may be sparse, while others are quite dense. This can result in a deficiency in parallelism throughout shortest path propagation since the algorithm would slow down at the dense clusters. The MFP scheme helps avoid the clustering problem by re-partitioning only those larger partitions that are dense. This will result in partitions of different sizes that cover different amounts of spatial area. In order to determine when to re-partition, a threshold is chosen as the maximum number of vertices allowed per partition. If a partition exceeds this threshold, it is re-partitioned and a new level is formed in the tree. Figure 5.5 shows a 3-level partitioning using MFP for nine

processors. Notice the refined partitioning in the dense areas. When partitioning with

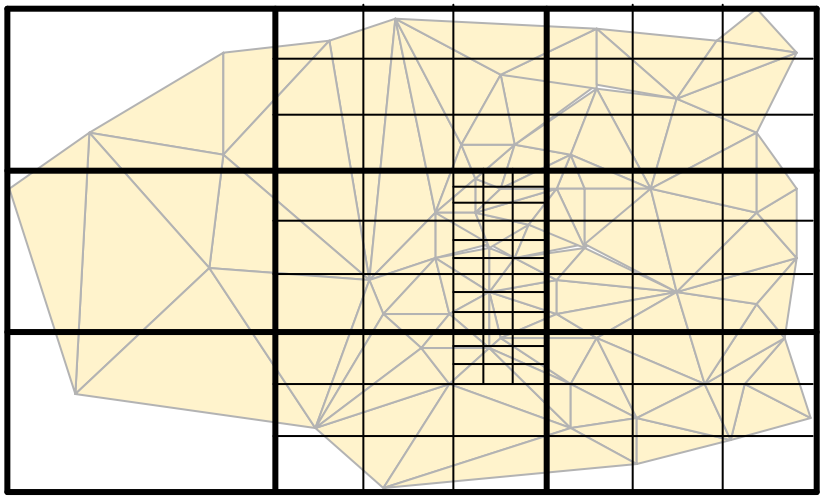

Figure 5.5: A 3-level MFP partition for a 3x3 processor configuration.

multiple levels, each processor must know which other processors it shares a partition boundary with in order to allow propagation over the boundary. The method of which a processor determines which other processors it needs to communicate with is known as a *mapping scheme*. Two important issues that must be considered in any mapping scheme are those of minimizing processor hops and avoiding bottlenecks. The first of these issues is important in a parallel machine in which processors are connected to each other by localized communication links, such as a mesh or torus (i.e., a mesh interconnection scheme with wraparound) topology. Thus the mapping scheme should allow messages to be sent to processors that are nearby (perhaps in the next row or column of the grid, as opposed to two or three rows/columns away). This will also help in reducing the congestion in the intercommunication network as there will be less messages "in transit" at any time. For machines in which a single shared "bus" is used between all processors, this is not an issue since there are no intermediate processors. The second issue is somewhat more important. If the mapping scheme

is poor, then it may be the case that certain processors are "favored" over others as being a receptacle for messages which can lead to bottlenecks. A good mapping scheme should treat all processors equally in terms of how much communication and work they are to perform.

The multilevel mapping scheme of the MFP handles both of these issues. It is best applied to a torus type of parallel computer since it handles wraparound. The mapping is done with respect to the levels of the tree (i.e., levels of recursive partitioning). We say that unpartitioned data is at level 0, while a single partitioning is at level 1, and so on.

Consider re-partitioning the data in processor p_{rc} to create one more level. The data must be re-partitioned again by splitting it up into R rows and C columns which are smaller partitions (called *cells*). The data for each of these newly created cells must now be re-assigned to the $p = R \times C$ processors. Let $cell_{ij}$ be the cell at row i, column j of the grid of cells where $cell_{00}$ is at the bottom left. We denote the *trivial mapping scheme* as that which maps $cell_{ij}$ to processor p_{ij}. The *MFP mapping scheme* is as follows. If a re-partition of the cells will represent an even level number in the MFP tree, a *backward mapping* is used, otherwise a *forward mapping* is used. A forward mapping assigns $cell_{ij}$ to processor $p_{((r+i)modR)((c+j)modC)}$ and a backward mapping assigns $cell_{ij}$ to processor $p_{((R+r-i)modR)((C+c-j)modC)}$.

Figure 5.6 shows an example of how the MFP partitioning scheme maps a partition recursively for a 3x3 mesh of processors. The example shows three levels of recursive partitioning, assuming a uniform distribution of data. The actual algorithm would only re-partition those partitions which are dense. To compare with the trivial mapping scheme, Figure 5.7 shows two examples of a level two partitioning for a 4x4 set of processors. Both the trivial and MFP mappings are shown. Notice that the MFP scheme tends to promote larger groupings of similar processor assignments. For

this particular example, when using the trivial mapping scheme each internal partition (i.e., non-corner and non-edge) is always adjacent to exactly four other partitions belonging to different processors. With the MFP scheme however, there are many partitions that are adjacent to only two or three other partitions belonging to other processors. This reduces the amount of overall communication during shortest path computation.

A further benefit of the MFP mapping scheme is that it does not "favour" any particular processor with respect to communication time. Consider a small portion of an MFP partitioning for a 3x3 configuration of processors as shown in Figure 5.8. The figure shows a small portion of partitioning and indicates both the trivial and MFP mappings. With the trivial mapping scheme, the bordering processors must communicate more than the internal (i.e., p_{11}) processor. With the MFP mapping scheme however, the border processors do not necessarily require more communication than the internal processor. The difference in communication requirements between internal and border processors becomes more apparent as the number of levels in the MFP tree increases. As R and C increase, a larger percentage of processors are internal which can also make the differences more noticeable.

5.2.2 Running the Simulation

We now describe the execution of the algorithm which assumes that the data has already been partitioned. Each processor begins with an initialization step in which upon startup, the processor loads its partition data from disk into memory. This is done only once and the data remains in memory until a new terrain is loaded. In a separate initialization step, it then initializes the cost to each vertex and initializes its own priority queue. The source vertex is of course given a cost of zero, while all other vertices begin with infinite cost. This step is kept separate since if many sets of

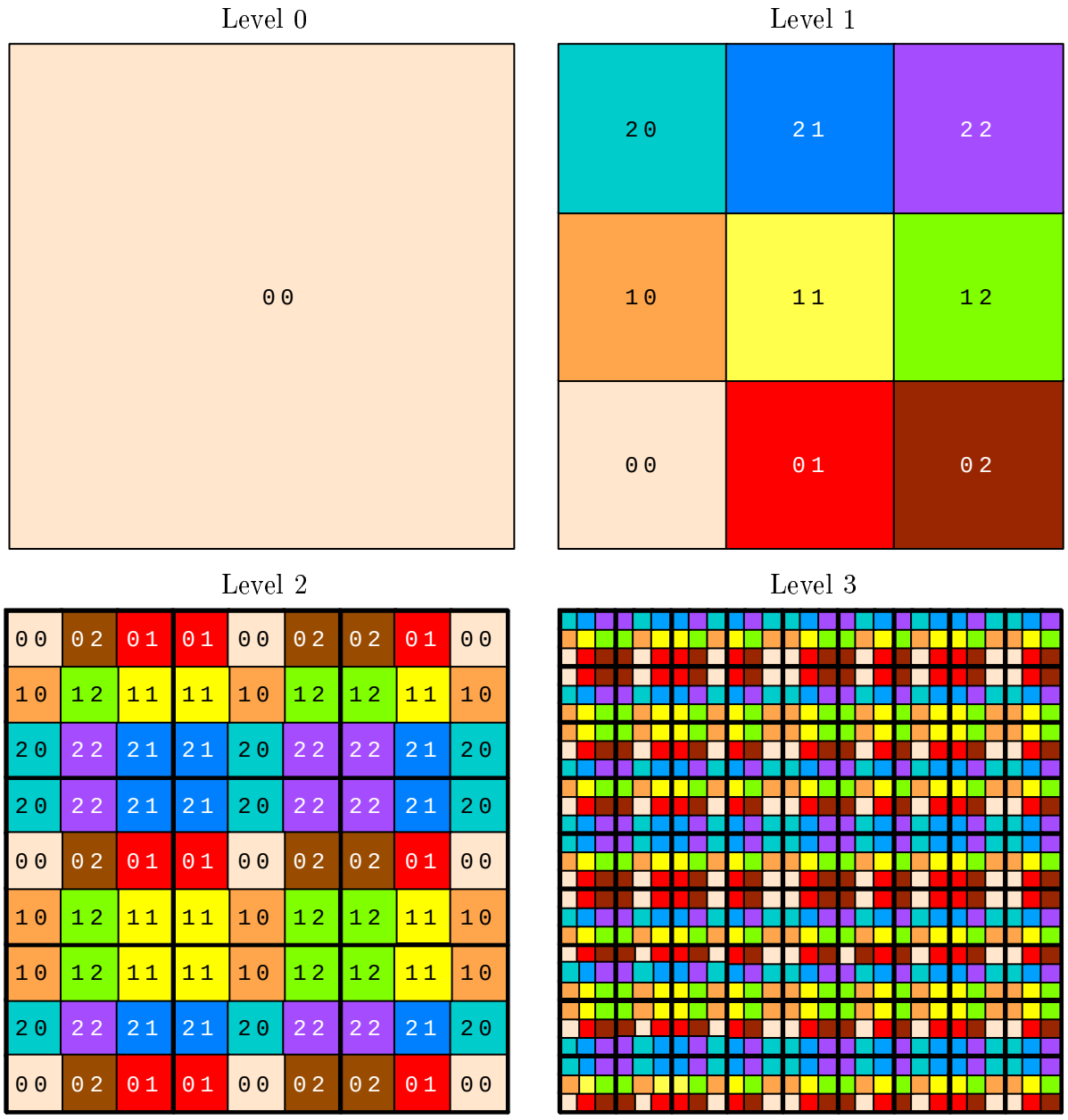

Figure 5.6: Levels 0, 1, 2 and 3 of the MFP mapping scheme for a 3x3 mesh of processors.

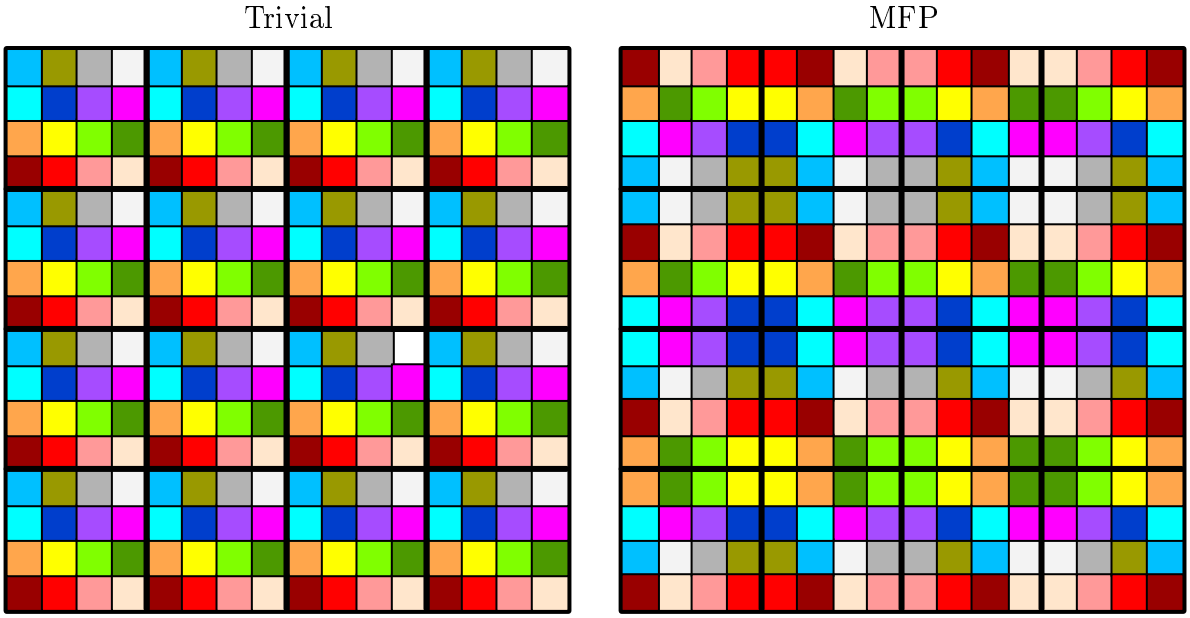

Figure 5.7: The trivial and MFP mapping schemes at level 2 for a 4x4 mesh of processors.

shortest path queries are to be performed (denoted as *sessions*), only this second initialization step needs to be done for each session. Two additional LOCAL_COST and MAX_COST variables are maintained locally for each processor. The LOCAL_COST variable maintains the cost of the last node which has been removed from the queue. The MAX_COST variable represents the maximum cost that the program will simulate up to. That is, once all nodes have a LOCAL_COST exceeding the MAX_COST, the processor will stop processing data. The MAX_COST has an initial value of infinity.

After initialization, the processor enters into a loop which does the actual shortest path propagation. The algorithm differs from Dijkstra's algorithm in that it must inform adjacent processors of vertex costs whenever the active border crosses a partition boundary. In addition, the condition for deciding when to stop the processing is

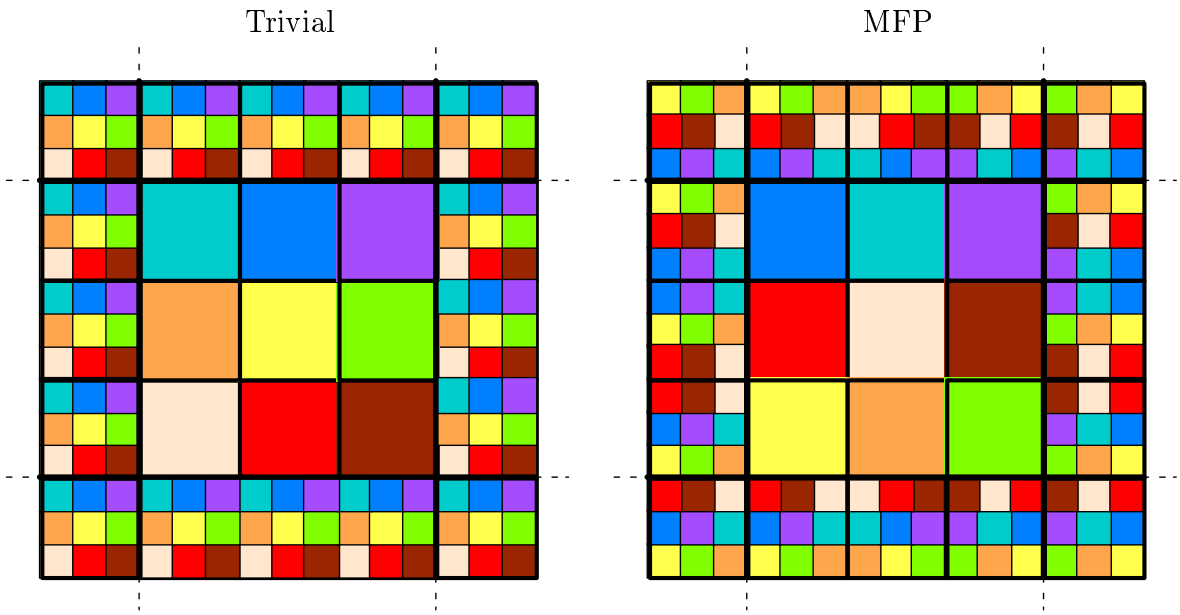

Figure 5.8: The trivial and MFP mapping schemes on a portion of a 3x3 partitioning showing that the border processors require more communication than the center processor.

slightly more complicated and involves passing around tokens to all processors. The pseudo-code for the algorithm is shown in Figure 5.9. Note that Q is the priority queue and TARGET is the target vertex.

It is easily seen that the bottom portion (i.e., last 14 lines) is quite similar to that of Dijkstra's algorithm. There is added code for sending updated costs to adjacent processors when a vertex is processed on a partition boundary (i.e., shared between two processors). If a processor receives a cost for a vertex from an adjacent processor, it must check to see if this cost is better than its own locally stored cost for that vertex. If so, this vertex is updated to have this new smaller cost and this change may cause a re-ordering of the queue. Processing then continues as before ... by removing the vertex with minimum cost (which may now be this newly updated vertex).

```
MAX_COST = infinity;  LOCAL_COST = 0;
WHILE(TRUE) DO {
    IF (there is an incoming message) THEN {
        IF (message is a cost token with cost C) THEN {
            MAX_COST = C;
            IF (this vertex was not the originator of the token) THEN
                Send cost token to an adjacent vertex; }
        IF (message is a done token) THEN {
            IF (this processor was the originator of the token and
                the token has gone around twice) THEN {
                Send a TERMINATE message to an adjacent processor;
                Quit: ``no more work to do''; }
            ELSE  Store the done token; }
        IF (message is an updated cost C for a vertex V and C < cost(V)) THEN
            cost(V) = C;
        IF (message is TERMINATE) THEN
            Quit: ``all processors have finished''; }
    IF ((Q is empty) or (LOCAL_COST > MAX_COST)) THEN {
        IF (this processor has the done token) THEN
            Send done token to an adjacent processor;
        Wait for an incoming message; }
    ELSE {
        Vmin = vertex with minimum cost;
        Remove Vmin from Q;
        LOCAL_COST = cost(Vmin);
        IF (LOCAL_COST < MAX_COST) THEN {
            IF (Vmin == TARGET) THEN {
                Set MAX_COST to be cost(Vmin);
                Send cost token to adjacent processor with cost(Vmin); }
            ELSE {
                FOR (each edge E incident to Vmin) DO {
                    V = vertex at other end of E than Vmin;
                    IF (cost(V) > (cost(Vmin) + cost(E))) THEN {
                        cost(V) = cost(Vmin) + cost(E);
                        IF (V is a shared vertex with processor p) THEN
                            Send cost(V) to p; }}}}}}
```

Figure 5.9: Pseudo-code for algorithm on each processor.

In order to reduce the graph storage overhead, the edges which are internal to the terrain faces are not explicitly stored. However, the terrain edges themselves (called *arcs*) are stored. The coordinates of the Steiner points along each arc are not stored, but their costs are kept in an array associated with the arc. For each arc that crosses a partition boundary, it is stored in both partitions. Whenever the cost changes for any of the endpoints or Steiner points along this arc, the whole arc (along with the Steiner point costs) is sent to the adjacent processor. The adjacent processor then decides which costs need to be updated.

The algorithm uses tokens which are sent to adjacent processors when the target is processed. A *cost token* is used to keep track of the maximum cost that the processor is allowed to process to. Once a processor extracts the target from its queue, it updates its local MAX_COST to be this cost and sends a cost token indicating this target cost to an adjacent processor. This cost token is passed from processor to processor in round-robin fashion. When a processor receives this token message from another processor, it stores the token's cost as its MAX_COST. If there are no more nodes in the queue with cost less than this new MAX_COST (i.e., LOCAL_COST exceeds MAX_COST), processing pauses for this processor.

In addition to the cost token, an additional *done token* is passed around to determine which processors have completed their computations. The done token originates from the processor that first reaches the target node. Upon receiving this token, if the processor has more work to do, it keeps the token. Once this processor then has no more work to do, it passes the token to the next processor in the ring. A counter is maintained within the token indicating the number of processors that had no work to do when the token arrived. Once this token has gone around twice with a count equal to that of the number of processors, a TERMINATE message is then sent to all processors to halt the processing and begin a traceback (construction) of the path.

Note that unlike Dijkstra's algorithm, if the queue becomes empty at any time, the algorithm does not stop. Also note that some minor details have been left out from the pseudo-code. For example, when a vertex is updated, the vertex must also store which vertex led to that one. That is, each vertex must keep a pointer to the vertex above it in a shortest path tree. In a sense, the union of all these pointers implicitly represents a shortest path tree.

Once processing has completed, the cost to the target is known and all that remains is to trace the path back from the target to the source. All processors receive a traceback message. The processor on which the target lies begins the traceback by piecing together the graph edges backwards from the target. The path is repeatedly traced back within the processor that contains the target by accessing the vertex before it in an implicit shortest path tree. The traceback continues until either the source is reached, or until a partition border is reached. In the later case, the processor must package up all path information obtained so far and pass it to the adjacent processor which must "take over" the traceback. Eventually, the path will be traced back to the source and this path is reported.

5.3 Experimental Results

In this section, we describe the experiments that we performed in order to verify the usefulness of our algorithm. The experiments were performed on our distributed memory ASP machine which is made up of sixteen 166Mhz Pentium computers connected via a dedicated crossbar switch. Each processor has 96MB of internal memory except processor 0 which has 128MB. MPI was used as our means of message passing between processors.

The objectives of the experiments were to investigate the performance characteristics of the algorithm by observing the effects when certain parameters are varied. The tests were run on actual terrain data as done in Chapter 2. We focus on timing results and ignore here path accuracy (since this has already been shown in Chapter 2 to be efficient). There are many factors that can affect the performance of our algorithm such as terrain size, partition strategy, partition sizes, number of Steiner points, cost function and location of source/target pairs. In order to provide a complete report on the effects of all of these parameters it would require testing all combinations of parameters for many iterations. However, if not enough variations of data and parameters are used, it is not possible to make an adequate assessment of their effects on the algorithm's performance. We have chosen a compromise in that we use an accumulative experimentation approach. That is, we test one set of data for a specified parameter set, then modify the parameters in stages to show the effects at each stage. Our experiments focused on the effects of partition, terrain and graph sizes as well as cost function variance and relative location of source/target pairs. We examine the effects on the overall runtime as well as the individual compute, idle and communication times of the processors.

The remainder of this section is organized as follows. To begin, the experimental test data procedures are described. We then describe our performance results for the single-level MFP partitioning scheme. There, we show the effects of varying the cost function, varying the relative locations of the sources/targets, varying the number of Steiner points and finally measure the amount of over-processing and re-processing. The results are then given in the case where multi-level partitioning is used.

5.3.1 Test Data and Procedures

As in Chapter 2, we attempted to run our tests on various terrains that differed in size and height variations. Since many partitioning schemes are susceptible to performance differences when clusterization occurs in the data, we also attempted to test terrains with different clustering situations. Table 5.1 shows the specifications of the terrains that were tested. The terrains were all constructed from cropped portions

NAME	#VERTICES	#FACES
Africa	5,000	9,799
Sanbern	8,000	15,710
Madagascar15k	15,000	29,582
America40k	40,000	79,658
Madagascar50k	50,000	99,268

Table 5.1: Terrains used for testing the performance of the algorithm.

of actual Digital Elevation Model data. Once cropped, the obtained grided terrain was then simplified through the removal of vertices, thereby obtaining a non-grid terrain. The vertices chosen to be eliminated were those such that when removed, the volume difference of the terrain was minimal. Figure 5.10 shows top-down view snapshots of the five terrains tested. Notice the left-sided clusterization of the America40k TIN.

In order to obtain proper speedup comparisons, the tests were run many times for a variety of processor configurations. The configurations were obtained by varying the number of rows and columns of processors. The configurations used were 1x1, 2x1, 2x2, 3x2, 3x3, 4x3, 4x4. For later tests with multiple levels of partitioning, only the 2x2, 3x3 and 4x4 configurations were used.

All tests performed used the "Fixed" scheme of Chapter 2. It has been shown previously that the interval scheme and additional sleeve-based schemes had near

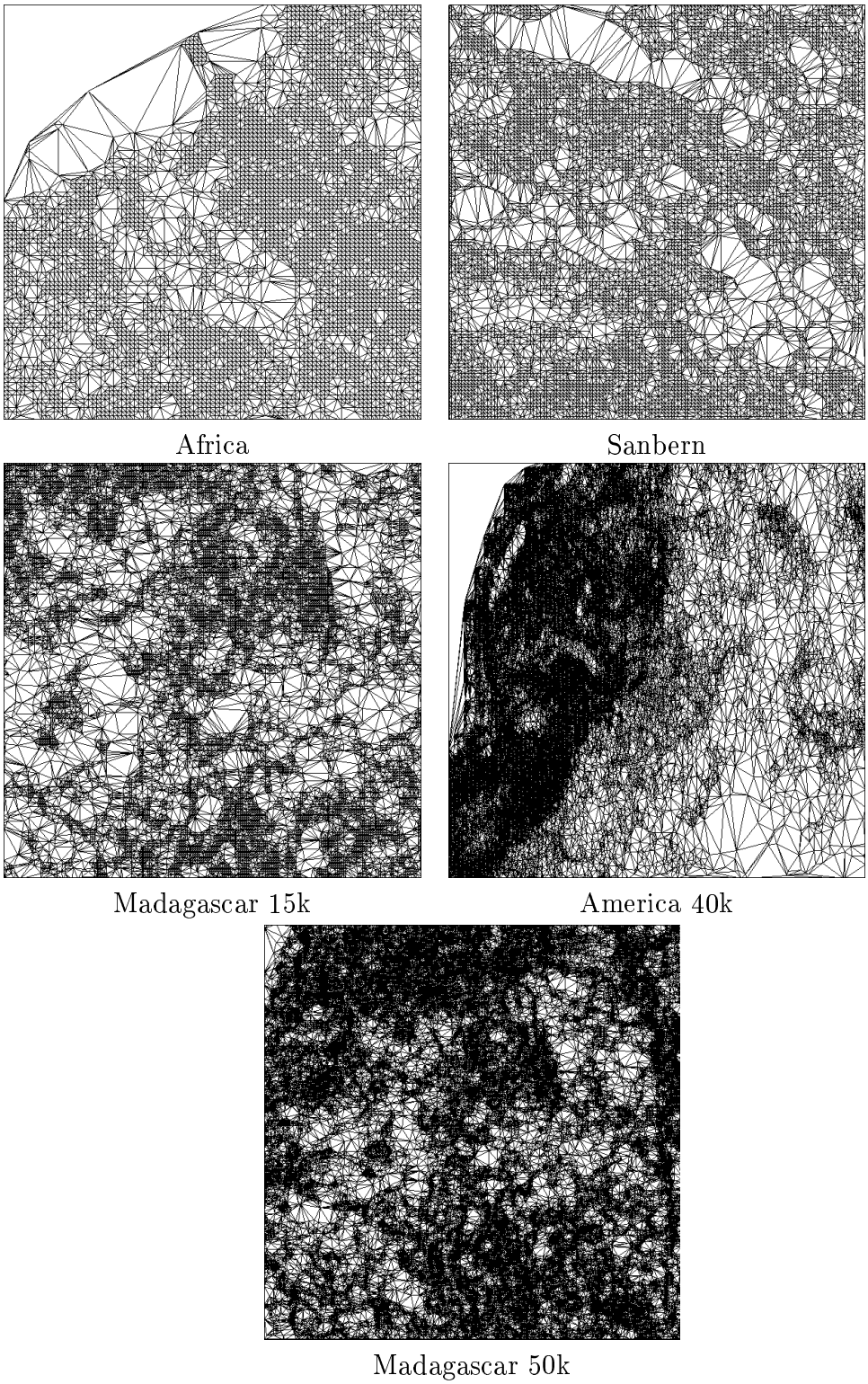

Figure 5.10: Top-down view snapshots of the five terrains tested.

identical timing in the sequential setting. We did not feel that it was necessary to run tests for all scheme variations since they differed mainly in path accuracy, not runtime. All our tests were run in the Euclidean setting although we also ran tests with a weighted (i.e., cost-intensive) cost function which are described later.

Our tests included both one-to-one and one-to-all variations. For the one-to-one tests, we computed a set of 50 random vertex pairs, each pair is called a *session*. For each session, we generated overall statistics that included path cost and overall run time. Most of our graphs show averaged information which is obtained by summing all the results per session and dividing by the number of sessions. In some cases, minimum and maximum results are also shown.

The *overall run time* is the time (in seconds) elapsed from when the session first starts until the final path is returned. The timing results presented here include the time required to compute the path itself, not just to produce the cost [1]. In addition to these, we also generated timing information per processor that included total run time, compute time, idle time and communication time.

Here, the *compute time* represents the amount of time that the processor spends doing computations with respect to its own queue. In addition, the time for generating the final path is also encapsulated within here. The *idle time* is the total amount of time that the processor remains idle with no computation or communication being performed. The *communication time* is the amount of time that the processor spends on communicating with adjacent processors. The *total run time* for a processor is the sum of these three times. The communication time in our experiments was determined by subtracting the compute and idle times from the total run time [2]. Lastly, we

[1] The overall run time was computed using the time() function in C which rounds off to the nearest number of seconds.

[2] These time were computed using the clock() function in C which rounds off to the nearest number of milliseconds.

generated (per processor) information pertaining to causality errors. This included the number of messages "sent to" and "received from" adjacent processors as well as the number of processed vertices, vertices inserted into the queue, vertices re-inserted into the queue and updated vertices.

5.3.2 Results For Single-Level Partitioning

Our initial tests used the one-level MFP partitioning scheme in which each processor is assigned a single equal-area portion of the terrain. Figure 5.11 shows the speedups obtained for our five terrains with various processor configurations. The graphs show the maximum, minimum and average case speedup based on the suite of 50 source/target pairs. From these graphs we can see that with 16 processors we reach an average speedup of around 2.4. In fact, in many cases, we have a slowdown. With at least one of the terrains (Madagascar50k) we were able to reach a maximum speedup of almost 4 when 12 and 16 processors were used. The performance is typically better for the larger terrains than for the smaller terrains. To see this, consider a small partition which has about x^2 internal nodes and hence roughly $4x$ boundary nodes. If the partition is increased by a factor of 4, the internal nodes increase to $16x^2$ whereas the the border nodes increase to only $16x$. It is easily seen that the amount of computation required before reaching a partition boundary also grows with the partition size and hence more work can be done before communication is required. This leads to a better speedup. However, we do notice that the performance of the smaller Madagascar15k terrain is better than the larger America40k terrain. This is easily explained since the one-level partitioning of the America40k terrain is unbalanced (the terrain is dense on one side and sparse on the other). Therefore, two processors are overloaded with computation while the other two finish quickly and then sit idle.

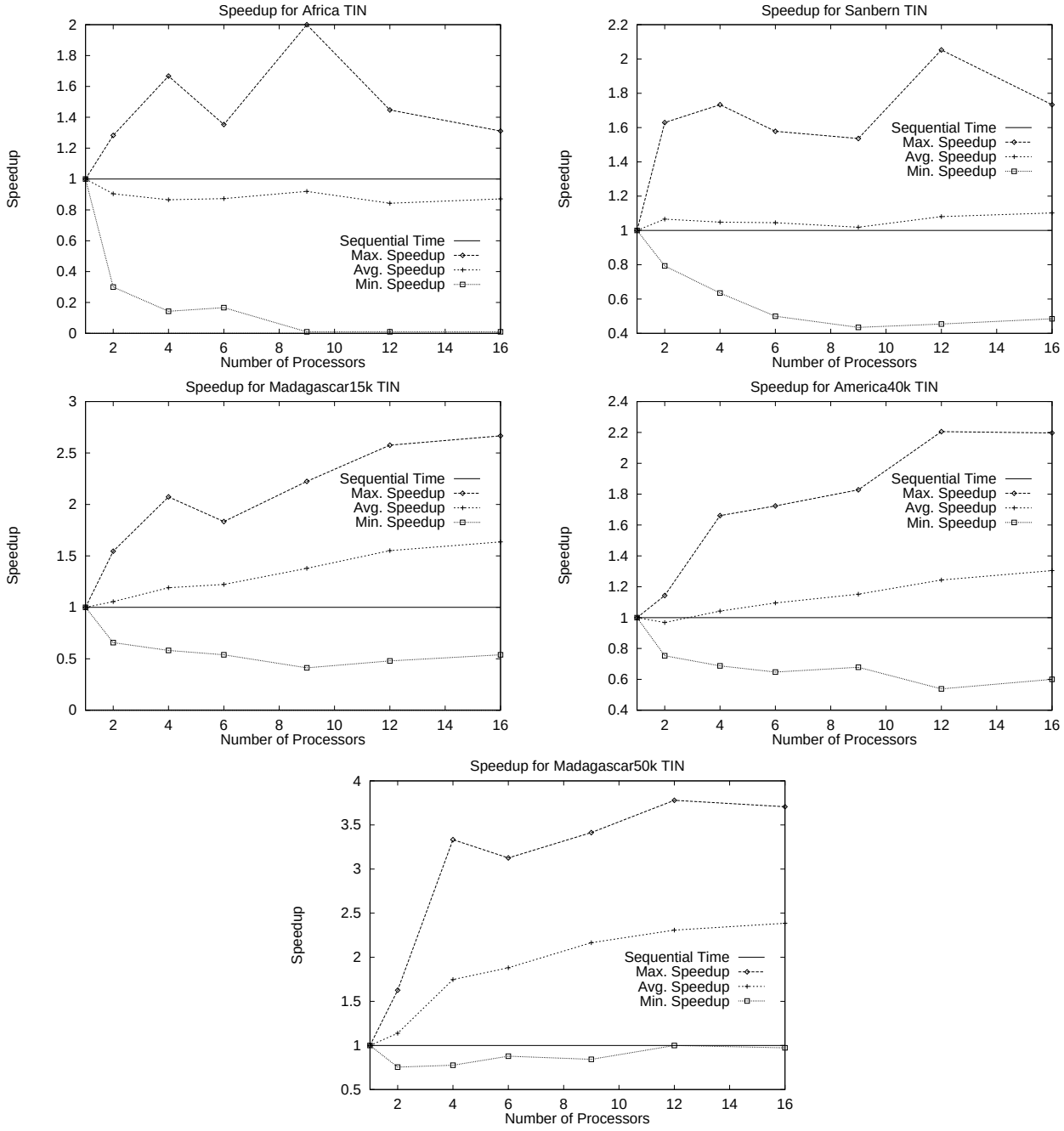

Figure 5.11: Graphs showing speedup obtained for five terrains.

In order to gain more insight as to the behaviour of the algorithm, it is necessary to analyze the distribution of the computation times, idle times and communication times. Figure 5.12 shows the processor usage for the same tests. Each bar depicts the amount of time that the processor spends on computation, communication and sitting idle. These values reflect an average over all processors and all sessions. All of the graphs show that as more processors are used, the idle time increases. With 16 processors we see that the processors sit idle more than half of the time. It is also noticeable from the graphs that the average communication time remains relatively constant regardless of the number of processors used (except of course for the 1 processor situation). This would seem to imply that the number of partitions does not affect the amount of communication between processors. This however, cannot be the situation in general since if we have a very large number of partitions (i.e., O(n)), then there MUST be a lot more communication across boundaries. We will investigate further the effect that multi-level partitioning has on communication time in a later section.

The graphs of Figure 5.12 do not give any indication as to how a single processor is affected with respect to the number of processors used. To help gain insight as to some characteristics of the partitioning scheme, Figure 5.13 shows a decomposition of the processor usage for each processor when the number of processors used is varied. The graphs depict the results from testing one particular terrain (Madagascar15k) for processor arrangement of 2x1, 2x2, 3x3 and 4x4. As is readily seen, there is quite a bit of variation in the idle times and compute times for the 3x3 and 4x4 tests. Consider the partition layout of these two tests. For the 3x3 test, processor 4 is at the center (hence, not a bordering processor). For the 4x4 test, processors 5, 6, 9 and 10 are at the center as well. Notice that the idle times for these non-bordering processors is smaller than others. Also, notice that the communication times are slightly larger. Intuitively, these center partitions are more likely to be crossed by a

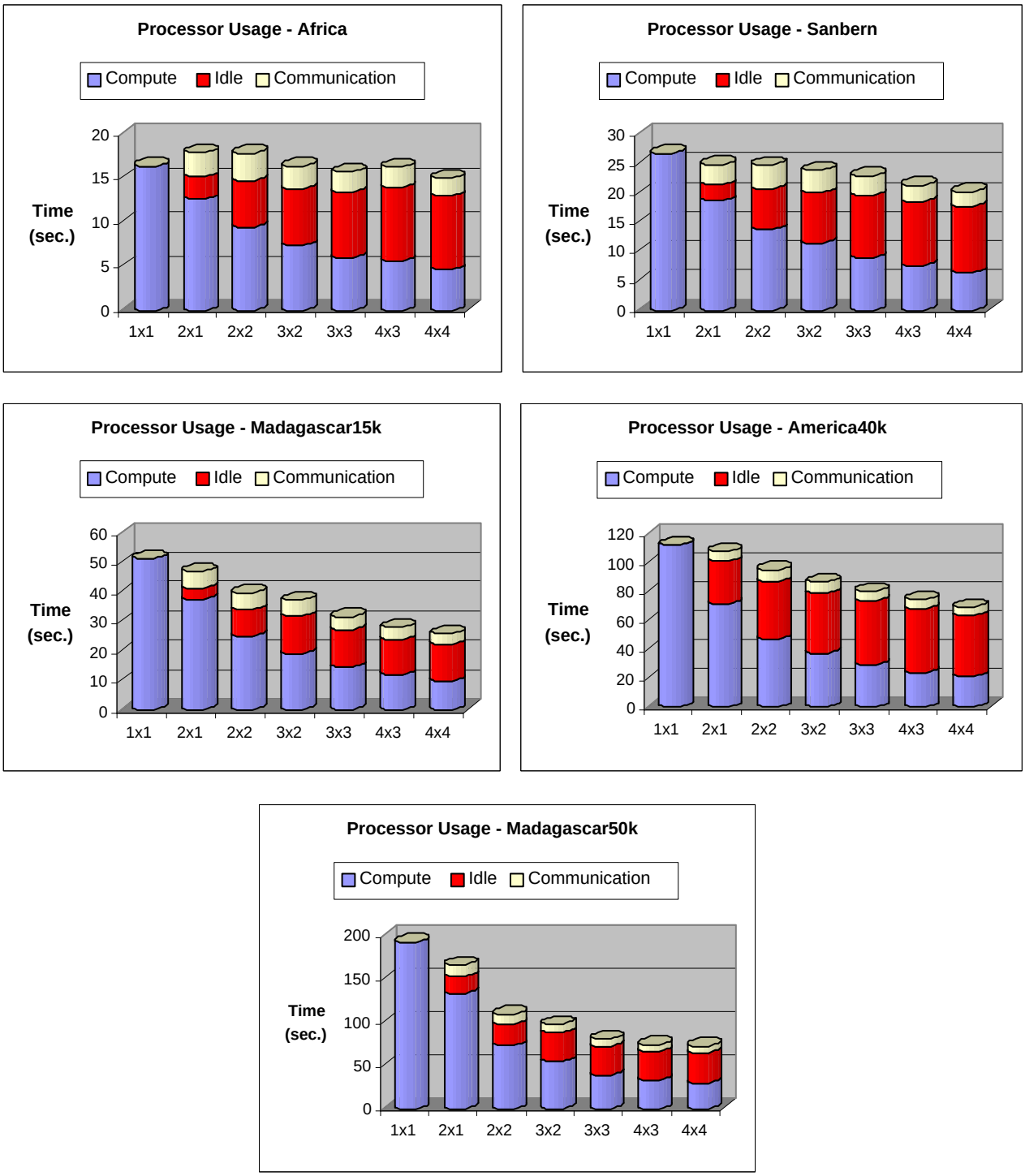

Figure 5.12: Graphs showing processor usage for five terrains.

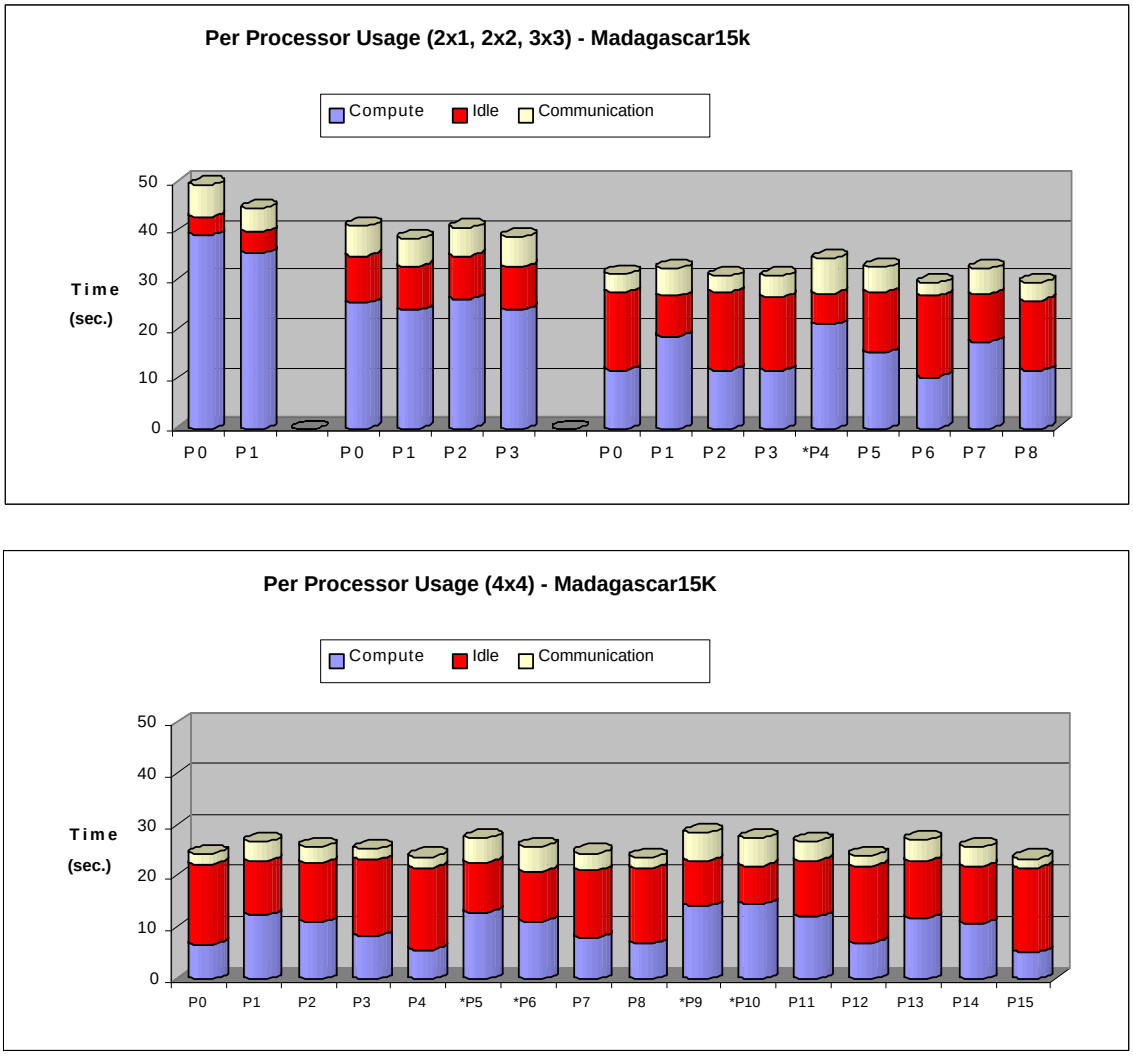

Figure 5.13: Graphs showing per processor usage for the Madagascar15k TIN for processor configurations of 2x1, 2x2, 3x3 and 4x4.

shortest path than a bordering partition. Hence, they are busier on average and also must communicate with more processors than the bordering processors communicate with. The differences between the corner partitions are also distinguishable from the side partitions as well, with respect to communication and idle times.

5.3.2.1 Effects of Varying the Cost Function

One method of increasing parallel algorithm performance is to increase the amount of computation done per processor relative to the amount of communication and idle times. It is by using computationally-expensive models that the true benefits of parallel computing become apparent. That is, with a higher amount of computation, the percent of communication time decreases with respect to overall runtime. To investigate the effect that the computation time has on overall runtime, we ran tests that incorporated a delay in the cost function. The delay actually performs roughly the same kinds of computations that would be required for the anisotropic path calculations of our algorithm in Chapter 4. This delay involves line segment intersection tests as well as computing numerous trigonometric functions. It was our intent to show that this delay would provide a noticeable change in speedup for larger data sets. Although this new cost function performs calculations for use with weighted (and/or anisotropic) paths, the computations are not used in determining the final path. Hence, the new cost function is only used as a delay and the path produced throughout our tests always corresponds to the Euclidean shortest path.

The running time for the cost-intensive tests were approximately twice that of the normal cost function tests. The percentage of time spent on computation increased with the cost-intensive function while the percentage of communication time decreased. This accounts for the increase in speedup that we observed in our tests as shown in the graphs of Figure 5.14. The graphs show the average case speedup for the

50 one-to-one sessions. It is clear from the graphs that with the more computationally-intensive cost function (and hence higher percentage of computation), the speedup is noticeably better.

5.3.2.2 Effects of Relative Source/Target Locations

As mentioned earlier, one important factor that affects the performance of a parallel shortest path algorithm is the relative location of the source and target with respect to one another. For example, if both the source and target lie within the same partition which is assigned to a single processor, most of the computations that will take place will be within that processor and all other processors sit idle. In some situations there will be excursions of the wavefront over the borders of adjacent partitions but these are on average minor with respect to the amount of computation done by the processor containing the source. Intuitively, the source target pairs that are in different partitions should produce better performance speedups. Also, the results should be better if the resulting path passes through many partition boundaries belonging to different processors. This was verified through our tests and is shown through graphs of Figure 5.15.

On the left are histograms showing the number of processor hops required for the different source/target pairs. The graphs depict results from the Sanbern, Madagascar15k and Madagascar50k TINs using the weighted cost function with a 4x4 mesh configuration. The histogram shows that of the 50 source/target pairs tested, most (76% - 86%) used less than four processor hops while very little (14% - 24%) used more than three hops. The graphs on the right indicate that as the source/target pairs get further away from one another (in terms of processor hops), the speedup improves on average. Since there are very few pairs in the 4 to 6 hop range, it is difficult to calculate a meaningful average and this accounts for the strange dips for

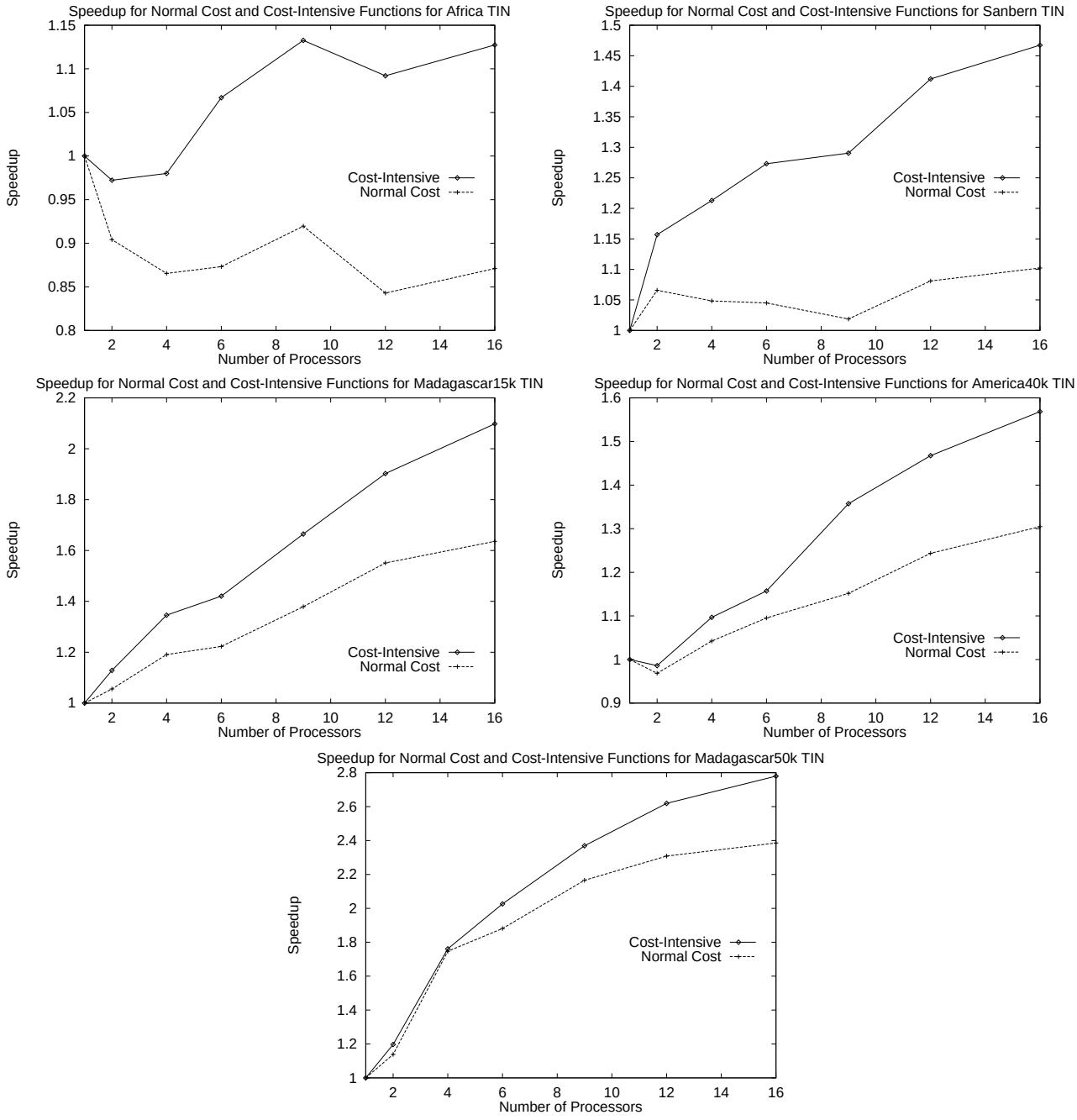

Figure 5.14: Graphs showing effect on speedup of increasing the cost function for five terrains.

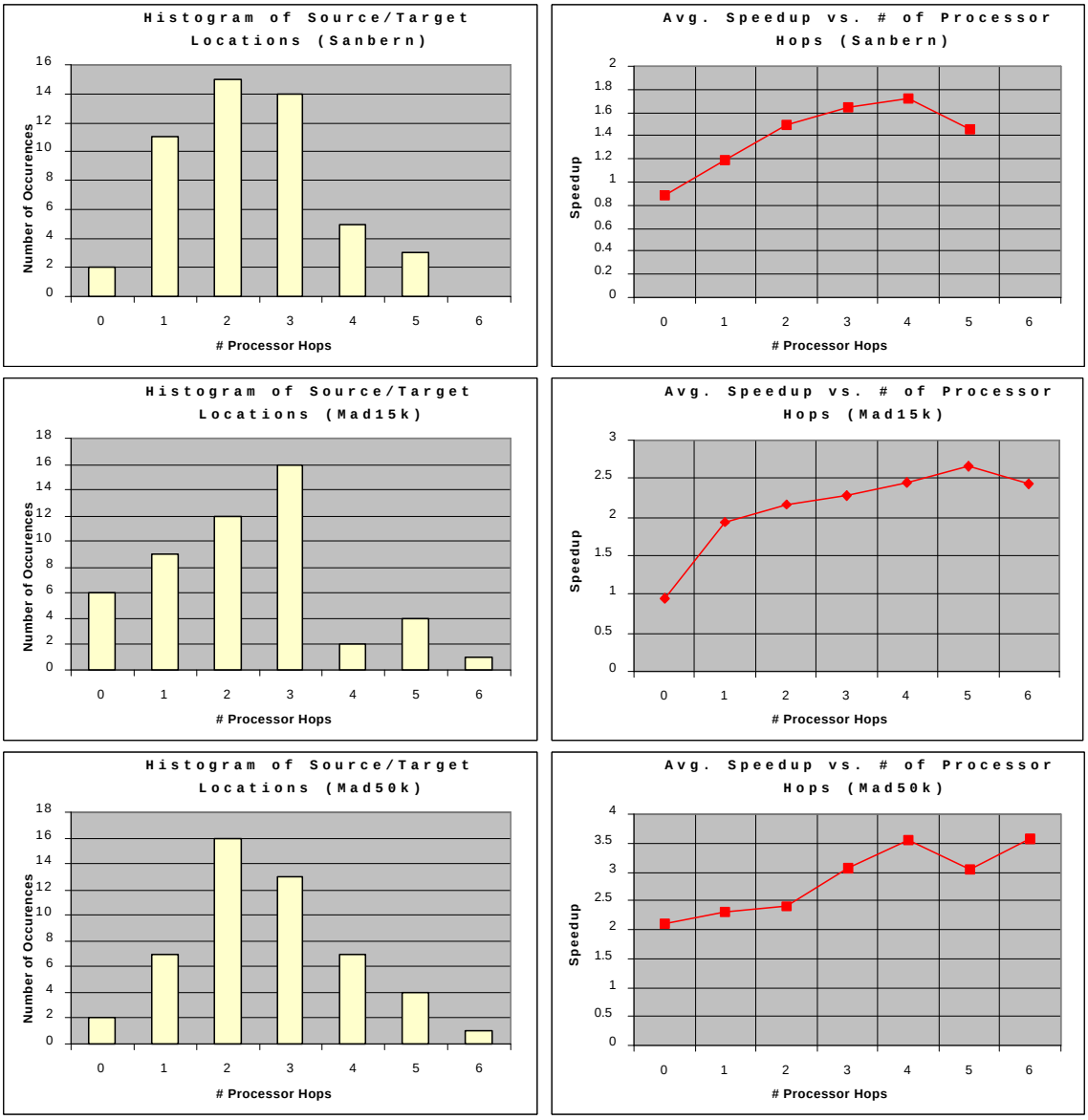

Figure 5.15: Histograms showing the number of processor hops between source/target pairs and corresponding graphs showing the effect it has on speedup.

the larger number of hops. It is therefore our conjecture that if the tests were run with source/target pairs further away, then better speedups would be obtained on average.

The best speedups (on average) should be obtained with one-to-all computations since we are ensured that all processors participate in the computation, which should eliminate the worst cases in which processors sit idle. To verify this, we ran one-to-all tests. In order to avoid biasing towards the selection of the source point, we chose two sources. The first test used a source which was centered in the terrain (middle) and the second used a source that was placed at the bottom leftmost vertex (corner) of the terrain. The results of these tests are shown in Figure 5.16 where they are also compared with the one-to-one tests.

The graphs show that better speedups are obtained for the one-to-all tests than with the average of the 50 one-to-one sessions (improvement of up to a factor of 2.7). Moreover, there is a significant difference in the results depending on the location of the source point. The results using the middle source are better than those using the corner source. The reason for this large difference can be explained by examining the idle times. Figure 5.17 shows the percent of idle time for both tests using the corner and middle source points for the Madagascar50k TIN. Notice that the processors are idle more when the corner source is used since the wavefront reaches all processors later than with the middle source point. Also, note in Figure 5.16 that the 3x3 processor configuration results in a noticeable speedup decrease for the smaller terrains. The reason for this is also related to the idle time since the outer processors remain idle until the middle processor has propagated to its boundary.

It is clear that the one-to-all shortest path problem obtains better parallelism than that of the one-to-one problem. This may explain why most research has been geared towards the one-to-all shortest path problem.

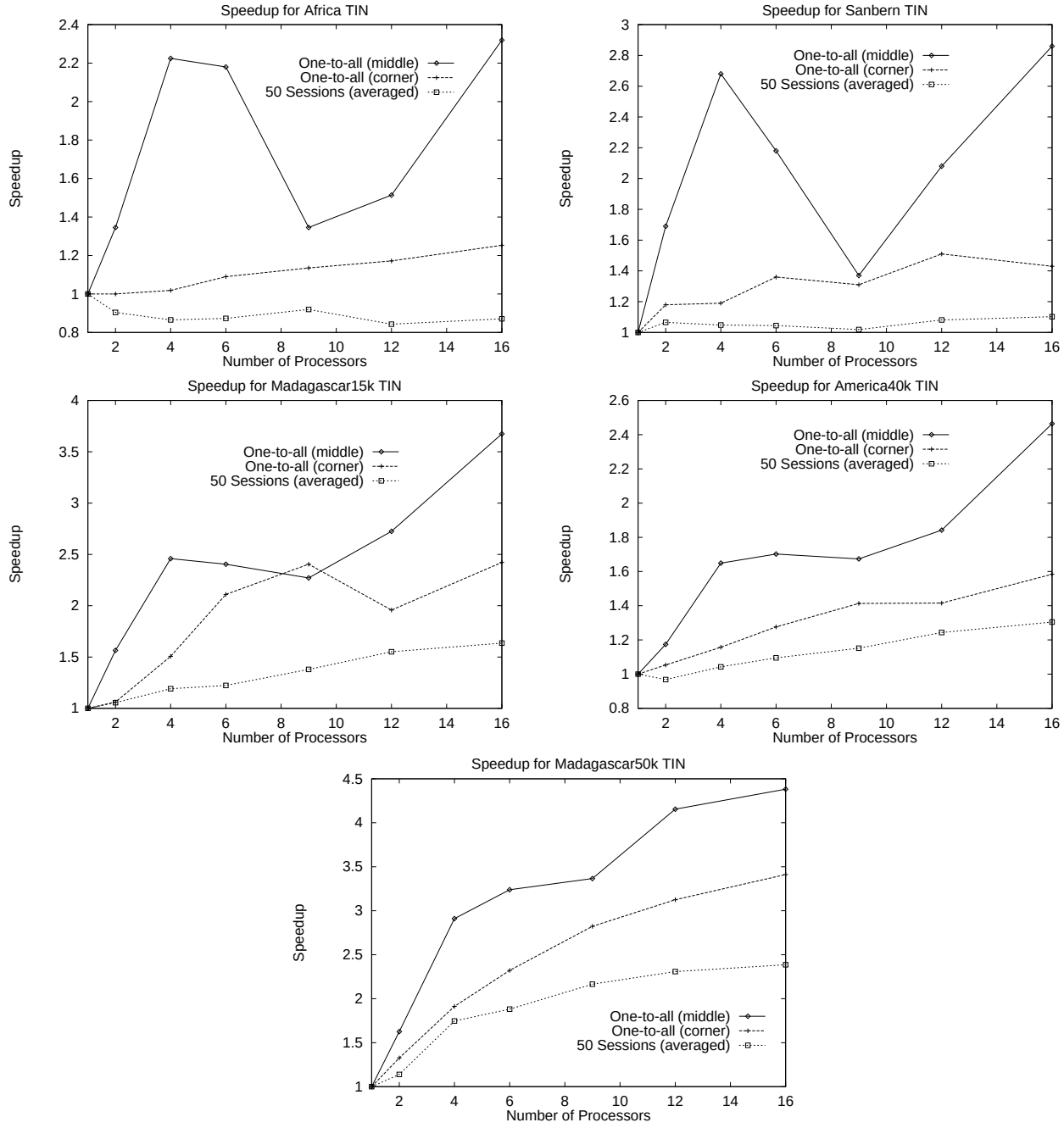

Figure 5.16: Graphs comparing one-to-one with one-to-all speedup for five terrains.

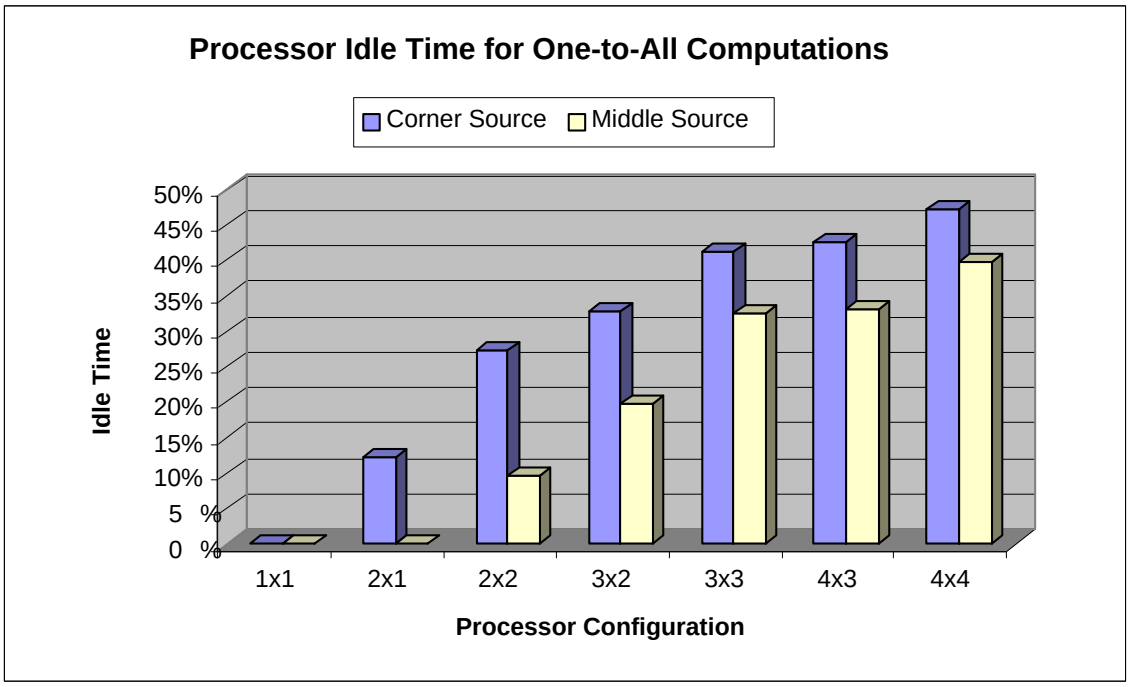

Figure 5.17: Graphs comparing processor idle time for the two starting source locations in the one-to-all tests for the Madagascar50k TIN.

5.3.2.3 Effects of Varying the Number of Steiner Points

Since the graph size and density depends heavily on the number of Steiner points, it is natural to assume that by varying the number of Steiner points added per edge, the performance of the algorithm would be affected. To verify this, we conducted tests that compared the usage of 3, 6 and 12 Steiner points per edge. All tests were ran using the weighted cost function. The running time increased on average by about a factor of between 2.0 and 2.7 as the number of Steiner points per edge doubled. An examination of running times showed that the percentage of compute, idle and communication times stayed the same (relative to one another) and was independent of the number of Steiner points. Since these three time components were unchanged

with respect to each other, the speedups also did not vary much. The graphs of Figure 5.18 show the speedups obtained for the 3, 6 and 12 Steiner point tests. The graphs do indicate a variance in the speedups. They show that better speedups are obtained using 6 Steiner points per edge as opposed to 3 or 12. However, the differences are small.

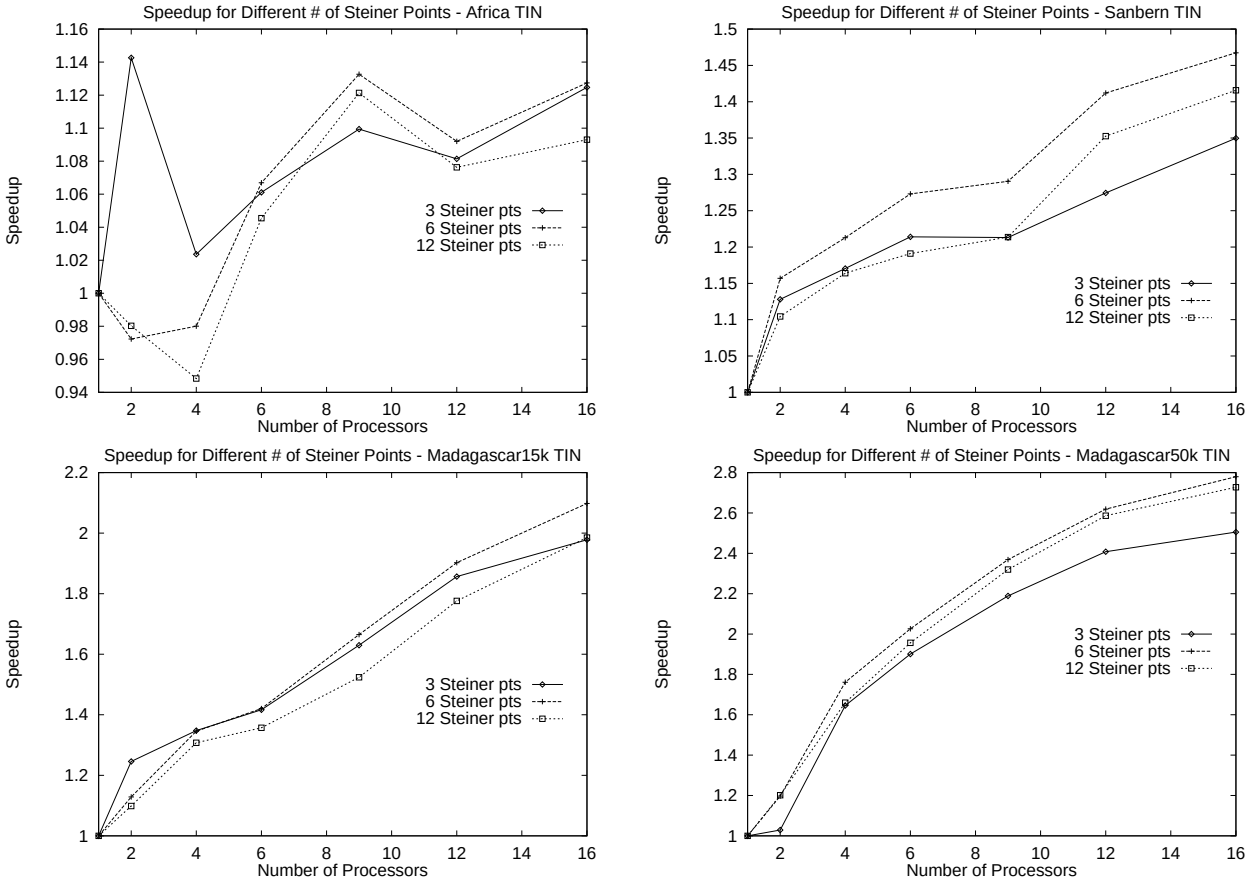

Figure 5.18: Graphs showing effect on speedup of increasing the number of Steiner points for four terrains.

Despite the number of Steiner points used, a processor always sends across an entire arc once it reaches a partition boundary. This includes all cost information

for every Steiner point along that edge. Therefore, for every crossing of a partition boundary (i.e., every communication step), the same number of messages are sent (i.e., only one) regardless of the number of Steiner points. The size of this message does vary slightly of course since more cost information is needed when more Steiner points are used. Also, even though the number of messages sent per boundary crossing is constant, the number of messages sent per arc is not. For example, consider propagation across an edge that crosses a partition boundary. Every time a Steiner point on that edge gets an updated cost, the arc is sent to the neighbouring processor. Thus, the arc is sent at least once per Steiner point (assuming that processing does not terminate part way through). This results in an increase in communication time as the number of Steiner points is increased. Figure 5.19 compares the total number of messages sent across partition boundaries for 3, 6 and 12 Steiner point tests. It clearly shows that the number of sent messages roughly doubles as the number of Steiner points doubles as well. However, the compute and idle times also increase since there are more Steiner points to process.

It is safe to say that the number of Steiner points used per edge does not play a significant role in the performance of our algorithm when the fixed placement scheme is used on small terrains. With every linear increase (i.e., m) in the number of Steiner points per edge, there is a quadratic increase (i.e., m^2) in the number of arcs of its incident faces. With the smaller terrains, the difference between m and m^2 has a more significant effect on the overall size of the graph. Hence, with larger terrains, the use of more Steiner points should result in a higher percent of computation time, thereby attaining better speedup.

Since we have shown in Chapter 2 that 6 Steiner points was enough to achieve good accuracy, we continued the remainder of our tests with this same number of Steiner points.

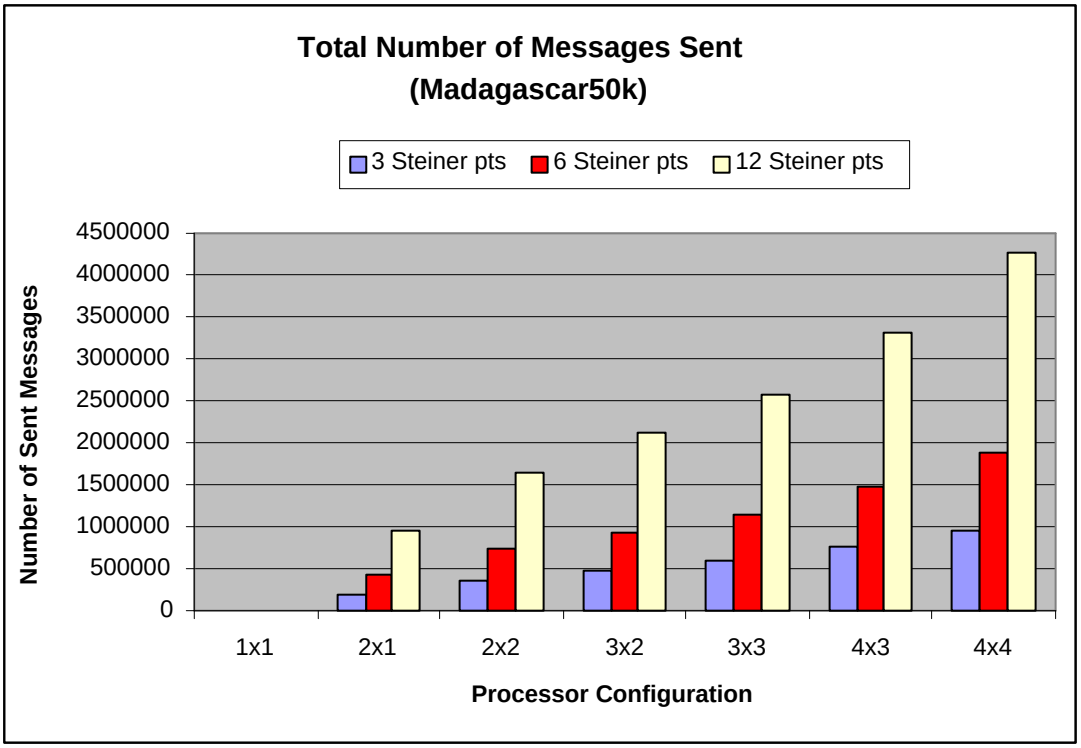

Figure 5.19: Graph showing the total number of messages sent for the 50 sessions as the number of Steiner points per edge is varied.

5.3.2.4 Measuring the Amount of Over-Processing and Re-processing

One measure of the efficiency of a parallel shortest path algorithm is the number of vertices that are processed. Hribar et al. [66] use a similar strategy in that they measure the total number of updates. Since faces (and hence graph vertices) are shared between adjacent partitions, the cost to these vertices may change many times if a shortest path crosses the boundary many times. More importantly, the cost updates along border vertices may cause a rippling of cost updates throughout the partition, thereby causing a re-computation of previously computed costs to many non-border vertices of the partition as well. As a result, the cost to any particular

vertex may be re-computed many times. In a sequential label-setting algorithm, once a vertex is removed from the priority queue, it never gets updated again. The parallel distributed algorithm is less efficient in terms of the number of vertices processed. We call this amount of additional computation *over-processing*. We also define the *re-processing count* of a vertex to be the number of times that vertex had been processed (i.e., removed from the priority queue).

When propagating over a boundary, another processor "takes over" a portion of the active border and processes in parallel to other portions of the active border on other processors. Therefore, the global active border will typically grow larger than with a sequential algorithm for any particular source/target pair. This causes vertices to be processed by the parallel shortest path algorithm that otherwise would not have been processed by a sequential shortest path algorithm. Therefore, in order to obtain a "good" estimate as to the amount of re-processing done by our algorithm, we ran tests that processed the entire terrain. By doing this, we eliminate the need to consider processed vertices that would not have even been processed by the sequential algorithm.

We analyzed the results of our one-to-all tests with the six Steiner points per edge using the weighted cost function. We computed the amount of over-processing as follows:

$$\left(\frac{|V_p| - |V_1|}{|V_1|} * 100\right)\%$$

where $|V_p|$ is the number of processed vertices when p processors are used and $|V_1|$ is the number of processed vertices when one processor is used.

The graph of Figure 5.20 shows that the amount of over-processing generally increases with the number of processors but also that the partition itself plays a role as well. For example, notice that the 4x4 partition has less over-processing than the 3x3 partitioning for the two small terrains. Notice also that the America40k

and Africa terrains have the largest amount of over-processing. This is due to the unbalanced load across the processors due to clustering. Figure 5.21 shows how an increase in the number of processors causes an increase in over-processing on average. The graph shows the results for the corner source, middle source and an average of both. These three curves present an average over all five terrains. The graph of figure 5.22 indicates the effects of terrain size on over-processing (averaged over all processor configurations). It shows that the size of the terrain has neither an increasing nor decreasing "trend" on the percent of over-processing.

5.3.3 Results For Multi-level Partitioning

We have shown through our graphs that without multi-level partitioning, processor idle time is a significant factor that hinders the parallelization efforts of the algorithm, resulting in poor performance. In some cases, many processors sit idle for up to 50% of the time. This phenomenon has also been observed by Hribar et al. [66]. We have also shown that the communication time does not play a large role in the poor performance with the single-level partitioning since little time is spent on communication between adjacent processors. It is clear that a better partitioning scheme that will reduce the processor idle time is necessary.

We now discuss our test results for the multilevel MFP partitioning scheme. Our tests were run on two of the same terrains as before (Madagascar 50k and America 40k) so as to allow a comparison with our previous tests. We ran two one-to-all sessions as before (i.e., with the sources being at the corner and middle of the terrain). Our test objectives for this scheme were to show that the processor idle time would be reduced, thus resulting in better speedup for all terrains.

In our tests, we have chosen 3 different tile sizes for each terrain and for processor

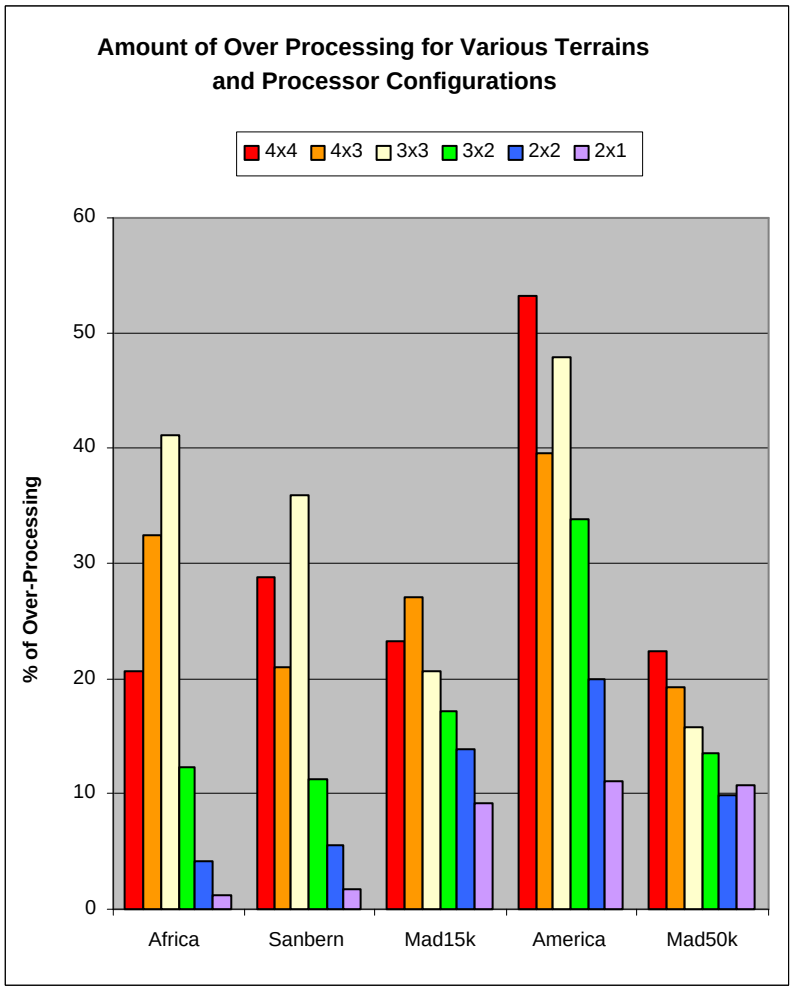

Figure 5.20: Graph showing over-processing for various terrains and processor configurations.

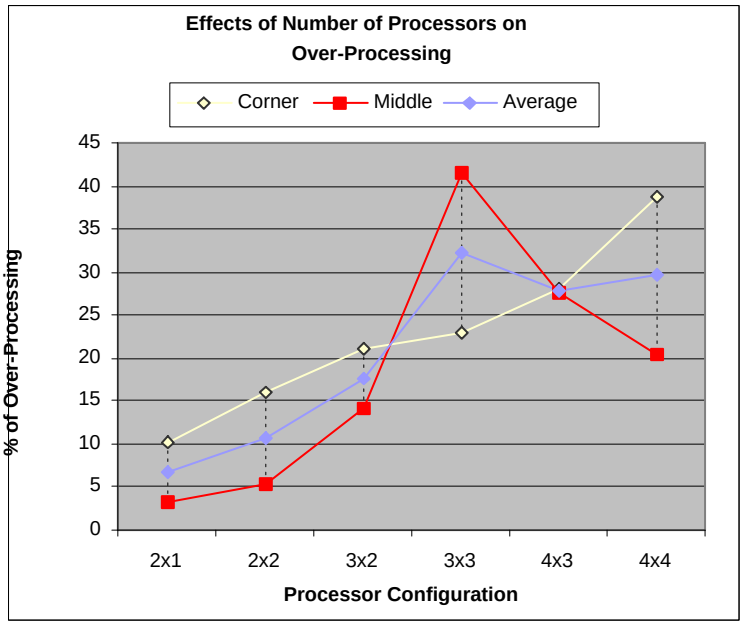

Figure 5.21: Graph showing effects of processor configuration on over-processing. Over-processing percentages are averaged for all terrains.

configurations of 2x2, 3x3 and 4x4. The tests were not aimed at determining the "best" tile size for each terrain but rather to observe the effects on speedup and idle time as the tile size is varied. Note also that we produced decompositions that used at most 3 levels of partitioning. We then computed the speedup and compared it to the single level of partitioning. With our initial implementation, the cost of communication was high and hence the benefits of the multi-level scheme were not as apparent in the results of our tests. At this point, we investigated our implementation further and made some improvements in order to reduce the communication speed. Our main changes were to decrease the size of each message being sent and to change to a more efficient version of LAM MPI.

Our initial change reduced the size of all messages sent during propagation across

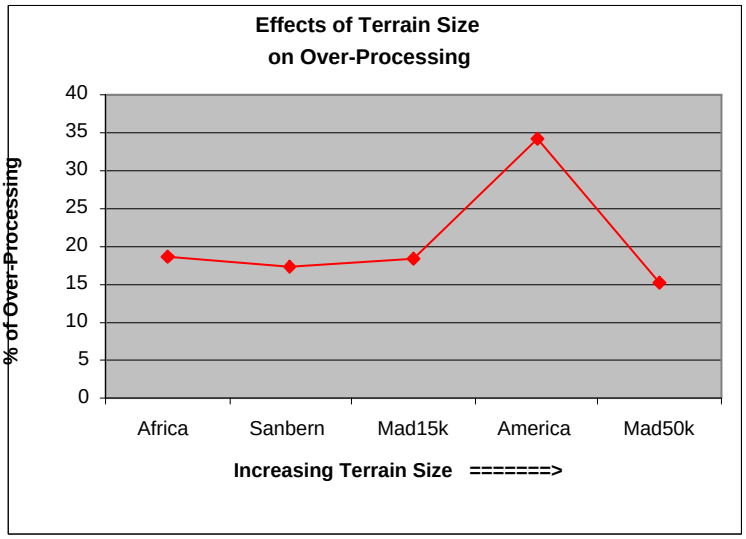

Figure 5.22: Graph showing effects of terrain size on over-processing. Terrain sizes increase from left to right.

the partition boundaries. Our original implementation used a fixed size array to store all incident arcs from a vertex. Whenever propagation reached the boundary from some Steiner point which was shared between two partitions, this entire array was sent in the message. This is not always necessary however, since propagation from Steiner points on an arc requires at most four arcs to be sent in the message. Our initial implementation sent the entire array in each message. Our improved implementation sent only those arcs that were necessary. The second change was to switch from version 6.1 of the LAM MPI libraries to the more efficient version 6.3b libraries. In version 6.1, the MPI_Iprobe function was implemented rather inefficiently and a call to this function would cause a significant delay. The newer 6.3b version has improved significantly and this function is much more efficient and returns much quicker than with version 6.1. As will be seen, the difference in this function's efficiency accounts for the difference in computation time between the slow and fast communication tests.

The graphs of Figure 5.23 show the differences in speedup between our original implementation (i.e., the slow communication) and our improved implementation (i.e., the fast communication). The difference in speedup between the implementations is significant for the smaller tile sizes and less significant for the larger tile sizes. The effect of increasing the communication time with the smaller tile sizes is not as noticeable when the faster communication is used. That is, the communication overhead of having many small tile sizes becomes insignificant as communication speed increases and this leads to better overall speedup. With the slower communication, the larger tile sizes (and hence the single-level partitioning) outperform the smaller tile sizes. Notice that the America40k terrain (which was the most clustered terrain) showed a more significant speedup than the Madagascar50k terrain when using the multi-level partitioning. This indicates that the multi-level partitioning scheme has an advantage over the single-level partitioning scheme for terrains that are more clustered. Also shown on the graphs is the maximum efficiency that was obtained from our tests. Notice that efficiencies of around 25% were obtained with the 4x4 processor configuration and this efficiency increases to up to 42% and 59% for the 3x3 and 2x2 processor configurations, respectively.

Figure 5.24 shows the processor usage for different processor configurations and tile sizes for the Madagascar50k and America40k terrains. The leftmost set of 12 bars in each graph represent the processor usage when the slow communication speeds were used and the rightmost for the fast communication speeds. Notice that the overall communication time is negligible with the fast communication. The single-level partitioning is the rightmost bar in each group four bars. As expected, the graphs show that as the tile size decreases (i.e., more re-partitioning is used) the idle time is traded off with communication time. Since our objective was to reduce the idle time of processors, the multi-level partitioning scheme has accomplished this. This reduction is more pronounced with the America40k terrain since it is much more

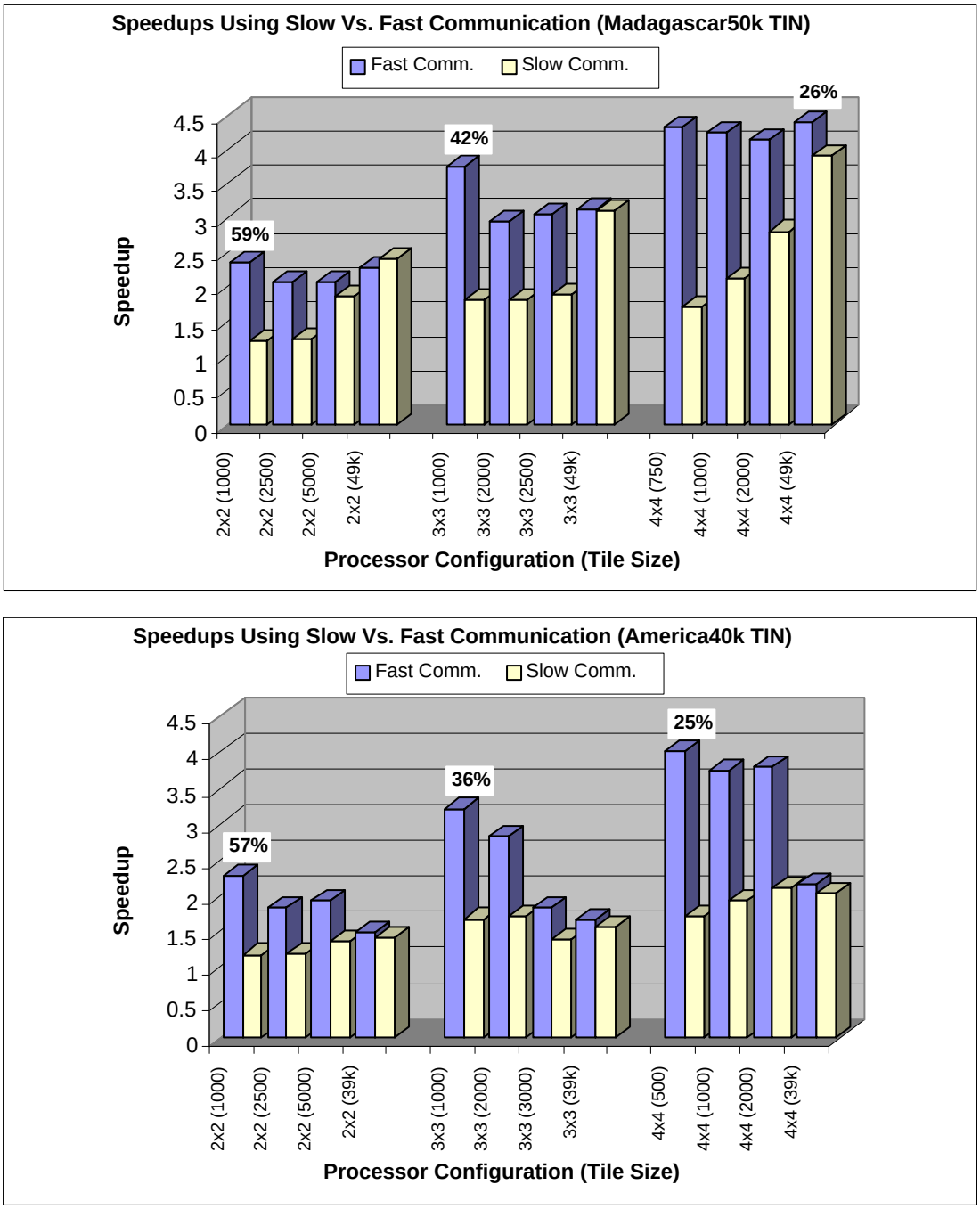

Figure 5.23: Differences in speedup between the slow and fast communication.

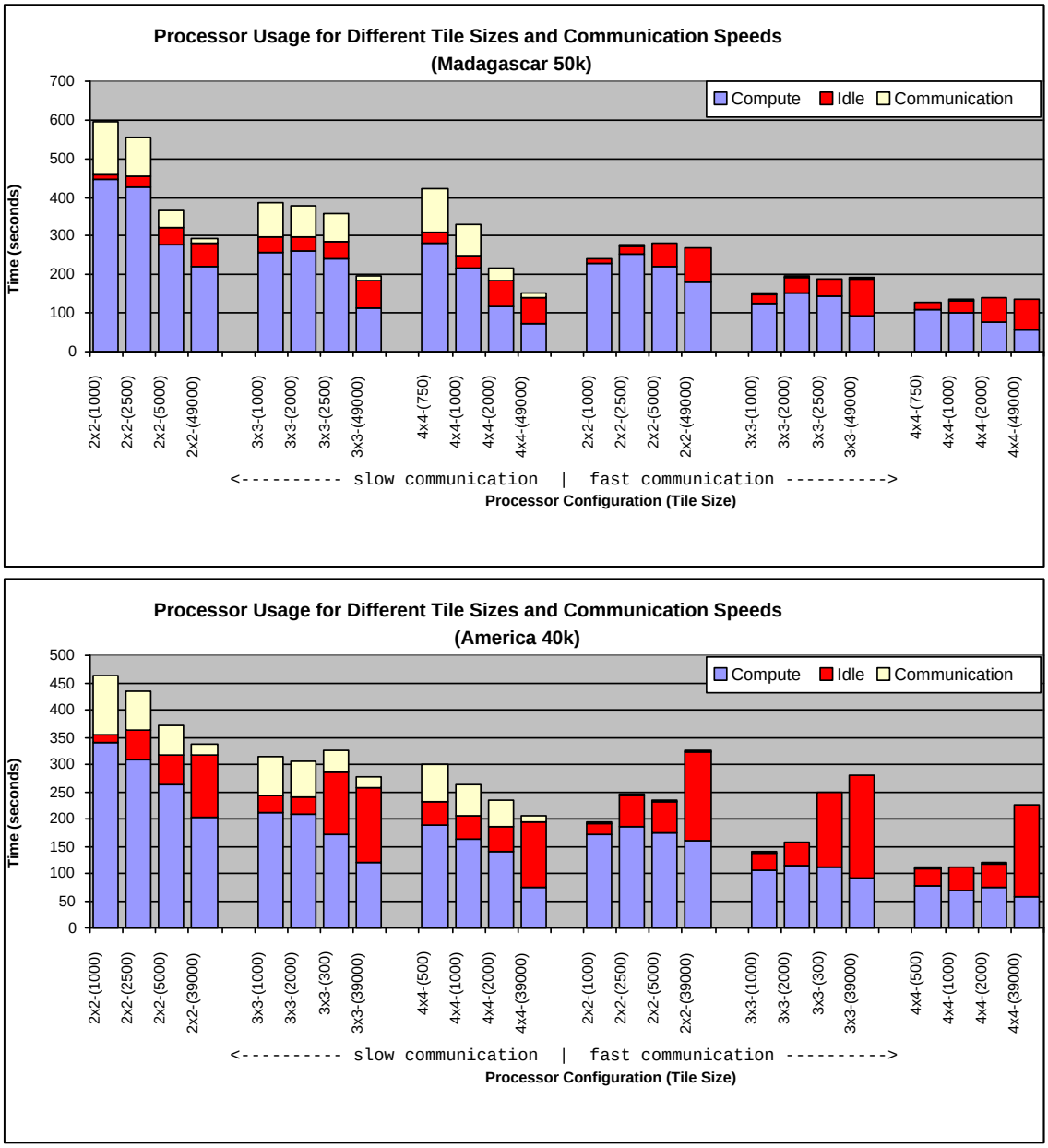

Figure 5.24: Graphs showing processor usage as processor configuration and tile sizes are changed for implementations with either slow or fast communication speed.

clustered than the Madagascar 50k terrain. With slow communication, this tradeoff is not as helpful with respect to reducing the overall runtime. However, with the faster communication speed, the tradeoff of idle time for communication proves to be advantageous and provides a significant improvement in speedup for the smaller tile sizes (when compared to the slow communication results).

The graphs of Figure 5.24 also show that the computation time significantly decreases with the faster communication (up to a factor of roughly two with the smallest tile sizes). This discrepancy is caused by the change in MPI versions between the two implementations. The algorithm checks occasionally to see if there are any incoming messages from other processors by using the MPI_Iprobe function. Since it is not used to send or receive messages, we decided not to count this as communication overhead, but rather as a computation overhead. The more efficient version of MPI as used in our improved implementation allows the computation time to be significantly reduced since the calls to the MPI_Iprobe function return quickly.

The graphs of Figure 5.24 also show that the percentage of computation time increases (in general) with the number of partition levels (i.e., the smaller tile sizes). Much of this is due to the increase in over-processing that is inherent to the multi-level partitioning. Figure 5.25 shows the amount of over-processing for each of the tile sizes for the original implementation (graphs for the improved implementation were similar). The graphs show that the single-level partitioning has the least amount of over-processing. Also, as more processors are used, the amount of over-processing increases dramatically with the smaller tile sizes.

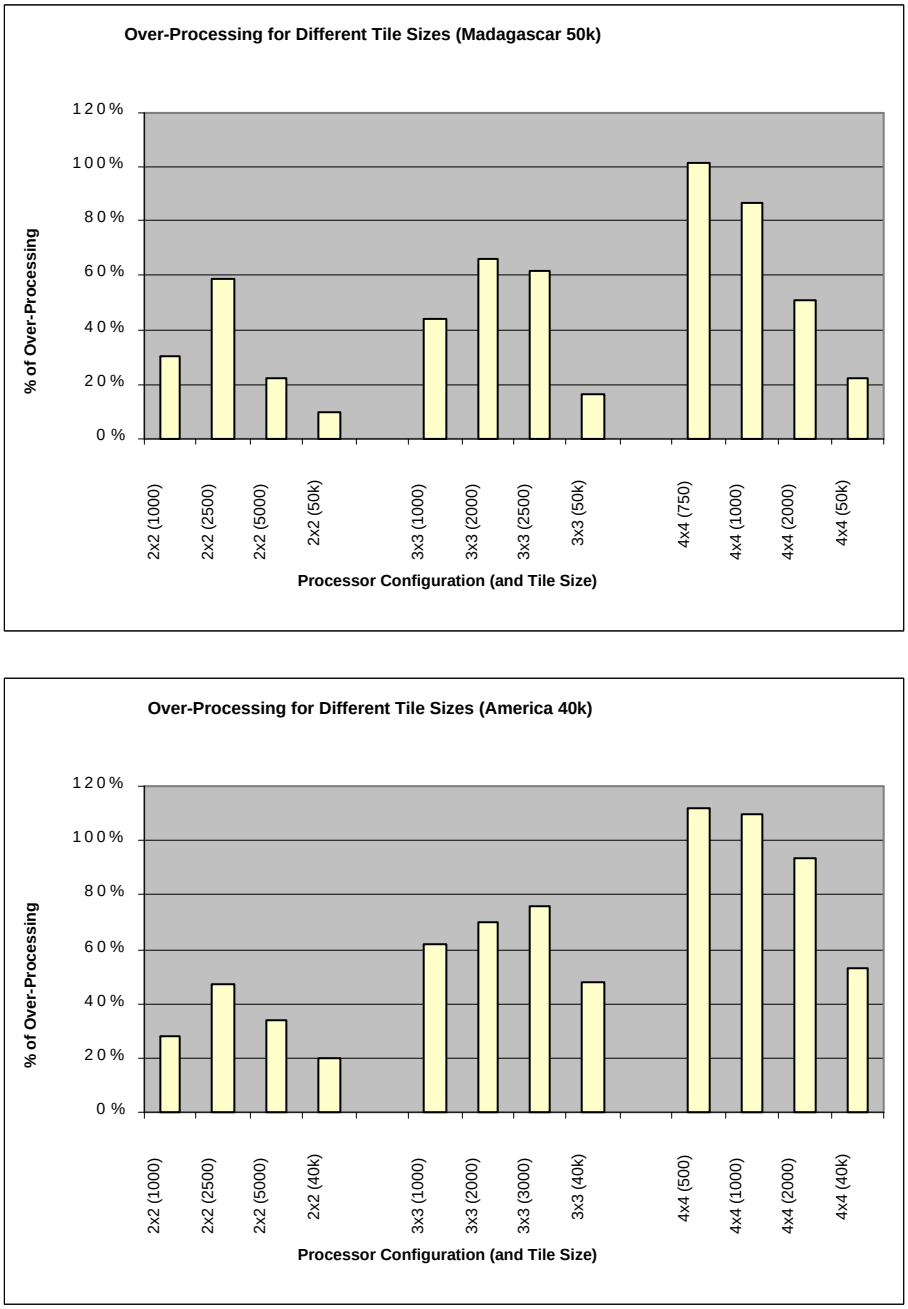

Figure 5.25: Graphs showing amount of over-processing as processor configuration and tile sizes are changed for two terrains.

5.3.3.1 Few-to-All Tests

One of the main advantages of the multilevel partitioning scheme is that it helps to reduce causality errors. To fully see the effects of causality errors, it is necessary to compute few-to-all shortest paths using multiple sources, each with a different starting weight. This kind of computation can be useful for facility location problems in which each facility has a different "attraction" factor and we would like to compute a weighted Voronoi diagram on the polyhedral surface.

We ran few-to-all tests on our America40k and Madagascar50k terrains. The tests used five source vertices on the terrain: four placed near the four corners of the terrain boundary and one near the center. The sources were assigned weights of 1, 50, 100, 150 and 200. We used four different partitionings which were formed using different tile sizes as our one-to-all tests as well as various processor configurations of 2x2, 3x3 and 4x4. The speedups obtained are shown in Figure 5.26.

The speedups are better than those of the one-to-all tests. Through these tests, the maximum efficiency has almost doubled for our tests with the 4x4 processor configuration. Increases in efficiency are also noticeable in the 3x3 and 2x2 processor configuration tests. The maximum overall efficiency obtained was observed from the 2x2 tests which reached a value of 64%.

Examine the results from the 4x4 tests on the America 40k terrain. Notice that the efficiency increases as the tile size decreases. If however, the tile size is made too small, then the efficiency will begin to decrease due to the overhead of communication. Hence, the peak in efficiency occurs with the optimal tile size which is likely less than or around 500. It should be noted that even better speedups overall can be obtained once the optimal tile size has been determined experimentally.

Figure 5.27 shows the difference in the amount of over-processing between the

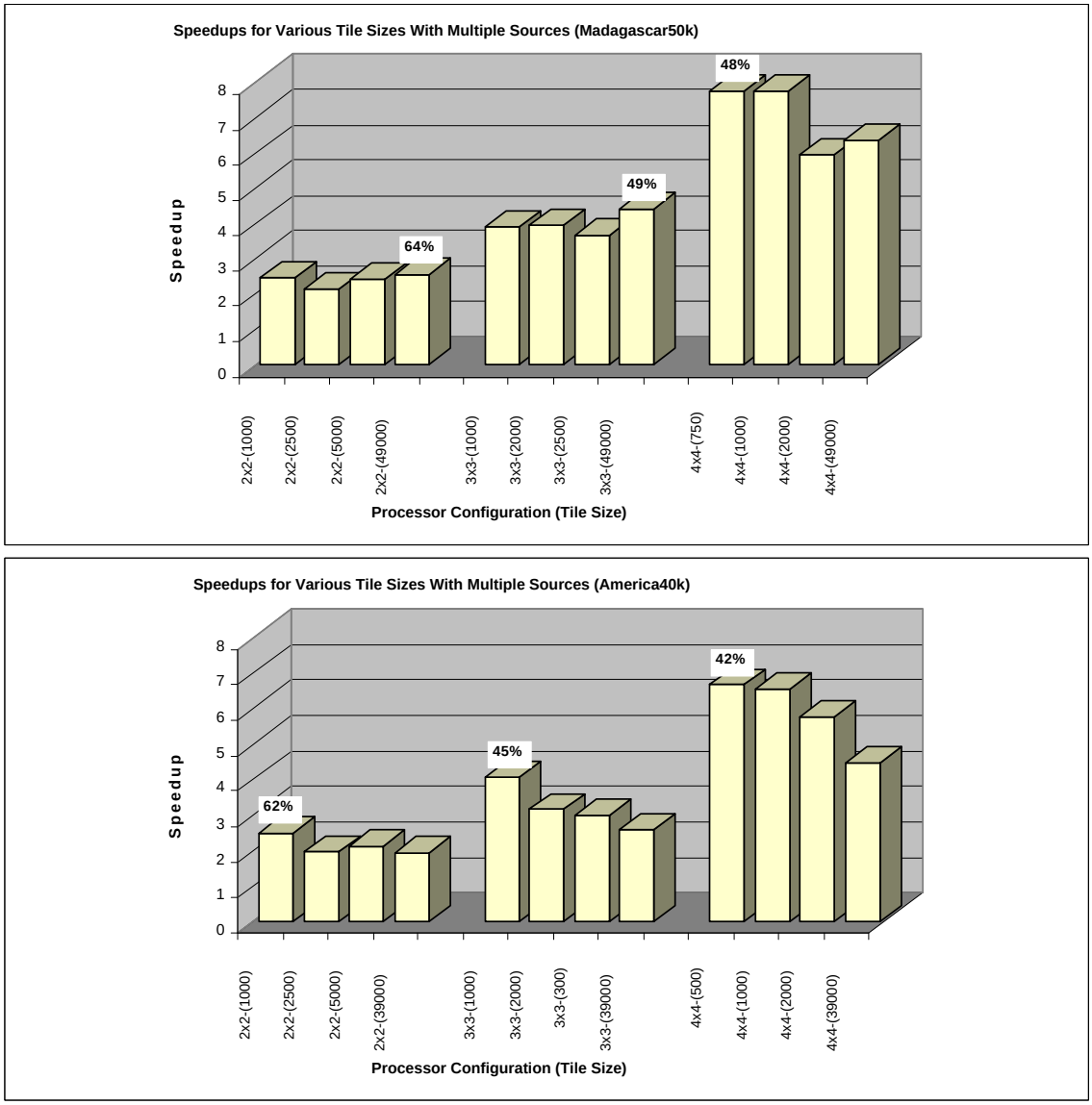

Figure 5.26: Speedups for few-to-all tests on America40k and Madagascar50k TINs.

single source tests and these new multiple weighted source tests. Notice that the amount of over-processing is significantly reduced in many cases when the multiple weighted sources are used. This improvement is most noticeable with the smaller tile sizes and larger number of processors. This indicates that the MFP partitioning scheme is better suited for weighted shortest paths with multiple sources.

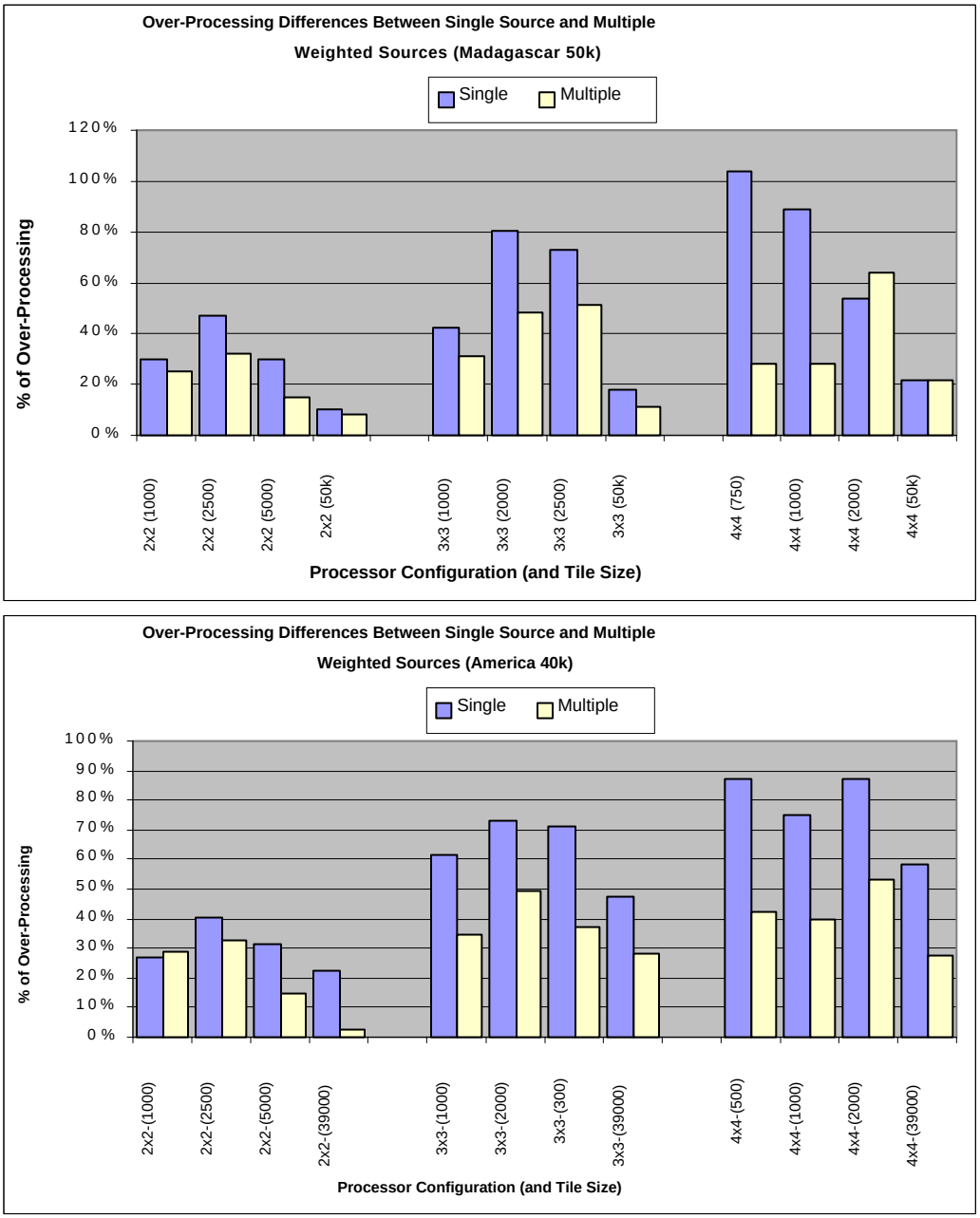

Figure 5.27: Comparison of over-processing between single source and multiple source tests.

Chapter 6

Conclusions and Open Problems

Shortest path problems belong to a class of geometric problems that are fundamental and of significant practical relevance. More specifically, the problem of computing shortest paths on polyhedral surfaces is of high interest in the area of GIS since it is a required computation for applications such as hydrology, search and rescue, road design, hazardous waste removal, visibility etc.. While realistic shortest path problems frequently arise in applications where the cost of travel is not uniform over the domain, the time, space and implementation complexities of existing algorithms even for the planar case are extremely high. These high complexities have motivated our study of approximation algorithms.

In Chapter 2, we began by describing a simple approximation algorithm in which a shortest path between vertices could be formed by traveling only along the edges of the polyhedral surface $\mathcal{P}$. As a result, we were able to transform the problem to a graph problem, thereby allowing efficient (and well-studied) graph shortest path algorithms to be applied. Our empirical results have shown that even this simple approximation strategy provided decent results which were less than twice that of the shortest path

cost. To increase the accuracy, we then developed a scheme that placed Steiner points along the terrain edges and interconnected them with a complete graph per face. As the number of Steiner points per edge is increased, the graph becomes more dense per face and the resulting path is more accurate. Experimental results have shown that only a few (constant) Steiner points needed to be added per edge in order to achieve high accuracy. This reduces the running time to $O(n \log n)$ in practice which has a lower dependency on n than previous approximation algorithms for the same problem.

We also theoretically establish bounds on the approximation quality and give worst-case bounds on the run-time of our algorithms. For the unweighted scenario, we compared our accuracy to that of Chen and Han [21] and gave results indicating that our algorithm performs up to 50 times faster with a minimum observed speedup of 14 times and produces nearly identical path results. We claim that our algorithm is efficient with respect to accuracy versus running time and is simple to implement. In order to validate our claim of practicality, we ran tests on many terrains which varied with respect to size, density, and height variation. We also ran tests on general polyhedral models (based on 3D data) in order to further verify the schemes.

In addition to the schemes presented in Chapter 2, we have designed other Steiner placement and interconnection schemes. Tests have shown that these other schemes did not perform well. This brings to mind a couple of interesting questions:

- Is there any advantage to placing Steiner points within the interior of the faces of P ? All the schemes that we have tested with Steiner points in the interior of faces have shown worse accuracy than our schemes that do not contain points in the interiors. The intuition of why this may be the case is that shortest paths do not bend in the interior of faces, so it allowing our approximated path to bend within a face would be counter-productive when computing path costs.

- How can m Steiner points be placed on each face such that the best accuracy is obtained ? Once again, intuition tells us that points should probably not be placed within a face but perhaps there are better ways to place the points along the edges of $\mathcal{P}$. For example, consider the subclass of TINs. Shortest paths are probably less likely to travel up mountains and then back down again, so it may be better to place less Steiner points around the peaks and more along the lower elevation edges.

Besides the variation in the Steiner point placement schemes, there are additional open problems. For the Euclidean shortest path problem, our implementation of Chen and Han's algorithm allowed us to compare the path accuracies obtained by applying our schemes to the true value. For the weighted shortest path problem, no algorithm exists to produce an exact solution. It is not even clear as to whether or not the weighted shortest path on polyhedral surfaces can even be computed to arbitrary accuracy. It remains thus open to establish whether our conjecture that the accuracy of our weighted shortest paths converges to the true value is correct. (The curves for weighted and unweighted look very similar suggesting correctness).

Our algorithm is also of particular interest for the case of queries with unknown source and destination. The preprocessing time taken to answer queries efficiently, increases the internal storage space requirements and, as also observed by Mata and Mitchell [86], it remains an open question as to how our shortest path algorithms fare experimentally in external memory settings.

A different class of approximation is an ϵ-approximation where we would like to find a path such that $\|\Pi'(s,t)\| \leq (1+\epsilon)\|\Pi(s,t)\|$ for some $\epsilon > 0$. We can prove that our vertex-to-vertex schemes of Chapter 2 are ϵ-approximations. However, the number of Steiner points required and preprocessing cost could be high, making this scheme of little practical value. The worst case bounds, when compared in the integer

coordinate model, match that of Mata and Mitchell [86]. In Chapter 3, we have presented algorithms to compute ϵ-approximate paths between two points on $\mathcal{P}$. The purpose of the algorithm was to improve upon the theoretical accuracy of our work in Chapter 2 as well as to improve upon previous research by reducing the dependency on n in the running time.

The complexity of our ϵ-approximation algorithm depends upon polyhedral parameters, including the number of faces, minimum vertex angle, maximum edge length and the ratio of weights. The actual path accuracy varies slightly depending on whether s and t are vertices of $\mathcal{P}$ or arbitrary points on $\mathcal{P}$. In any case, the path accuracy can be adjusted to be ϵ-approximate by replacing ϵ by some ϵ_1 which is ϵ/C, for some constant C which varies according to the cost metric (i.e., Euclidean or weighted) as well as the location of s and t.

It was shown that if s and/or t is chosen to lie arbitrarily on $\mathcal{P}$ that this path was also bounded but the first and last segments of these paths were not in G. When constructing G for the case in which s and t are given ahead of time, we merely extend G to contain additional graph edges from s and t to vertices of G corresponding to Steiner points on their face boundaries.

Our scheme can also be applied to the situation in which either or both of s and t are queried provided that we allow a preprocessing stage. In the case of a fixed source we can compute a shortest path map $\mathcal{M}_s$ from s to all vertices of G. If t is a vertex of $\mathcal{P}$, we can merely locate t in G and report the shortest path cost from $\mathcal{M}_s$ in constant time or the path itself in time $O(k)$, for a k-link path. An efficient implementation will allow vertex t to be located in constant time. If t is not a vertex of $\mathcal{P}$ then we could check all the Steiner points along the borders of the face containing t and then report the minimum path to t through one of these Steiner points in $O(m+k)$ time. For the case when both s and t are queries, a similar approach can be taken by

constructing a forest of shortest path trees using an all-vertex-pairs approach. Once again, a brute force approach can be taken by examining all $O(m^2)$ pairs of shortest paths between Steiner points of the faces containing s and t.

A different approach can also be used to solve this two-point query problem with a more efficient query time. An approach which followed from our work was presented by Aleksandrov et al. [9] which is based on using separators to divide the polyhedron into mn/r parts, where r is an adjustable constant. The polyhedron is preprocessed in $O(nmr \log r + nm^2 r + \frac{(mn)^2}{r} \log \frac{mn}{\sqrt{r}} + \frac{(mn)^2}{\sqrt{r}})$ time where $0 < r < n$. Queries can then be answered in $O(m + r)$ time.

We are currently generalizing the techniques used here in two ways. First, we are considering the problem of computing ϵ-approximate weighted shortest paths between two points among multiple polyhedra. The problem is as follows. We are given k disjoint polyhedral surfaces $\mathcal{P}_1, \mathcal{P}_2, ...\mathcal{P}_k$ and two points s and t lying on $\mathcal{P}_s$ and $\mathcal{P}_t$, respectively. Each polyhedron is triangulated and each face is given a fixed positive and non-zero real weight representing the cost of travel on that face. In addition, the free space between all polyhedra is given a fixed positive and non-zero real weight. We wish to compute a path $\Pi'(s,t)$ from s to t that consists of segments lying on the surfaces of one or more polyhedra as well as straight line segments in 3-Space that join two points of different polyhedra. Moreover, the path is an ϵ-approximation of an actual weighted shortest path $\Pi(s,t)$. The problem is essentially the weighted version of the problem previously studied by Papadimitriou [100], Choi et al. [26] and Clarkson [29] (see Chapter 1). The basic idea of this algorithm is the same as that which is presented in Chapter 3, except that additional Steiner points are placed interior to the faces of each polyhedron.

Our second generalization is to compute weighted shortest paths in 3D as follows. Consider a tetrahedralized polyhedral universe $\mathcal{U}$ where each tetrahedron is given

a fixed positive weight that represents the cost of travel through that tetrahedron. Given two points s and t that lie within two different tetrahedra, we would like to determine an ϵ-approximation to a weighted shortest path between them. To our knowledge, this would be the first algorithm to computed ϵ-approximations of weighted shortest paths in 3-space. Once again, the ideas are similar in that we now place Steiner points within the tetrahedra as well as on their faces and edges.

To investigate a more realistic model of computation, we investigated the problem of computing shortest anisotropic paths on the terrain. Due to the success of our initial schemes in Chapter 2, we decided to apply the same approach to solving this minimal energy problem. We supply two schemes for creating a graph, the first being more practical, while the second is more theoretical. We provide a theoretical analysis of these two schemes and show that ϵ-approximations are possible. Through experimental analysis, we have show that the first of these algorithms has similar convergence behaviour (with respect to path accuracy) as our results in Chapter 2.

One obvious extension to the metric which was used in our shortest energy path algorithm is that of restricting either the maximum allowable degree of each turn or the number of turns allowed in our approximation. The current approximations allow zig-zag paths which typically require sharp turns (possibly close to 180 degrees). For most vehicles, this is not possible. It would be more practical to produce paths that do not require these "sharp" turns. This would provide a more realistic solution that can be applied to real-world vehicles.

Some problems do arise, however, with this new constraint. With the previous model (which ignored turns), there was a possibility that for a given pair of points on $\mathcal{P}$, a valid path between them may not exist. With the added constraint on the turn angles, the probability of there being no valid path increases. The example of Figure 6.1 shows a valid switchback path between two points on a single face. The example

shows the face with very steep slopes on the adjacent faces which may not allow traversal. Hence, if sharp turn angles are disallowed, no valid path exists between the two points shown.

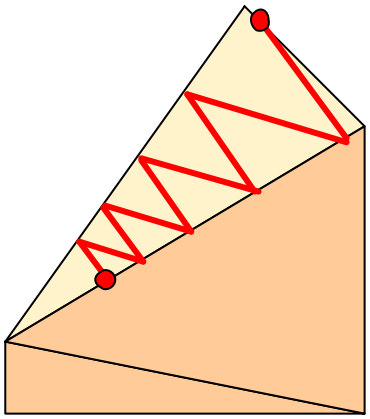

Figure 6.1: Example showing a switchback path between two points on a face. If the switchback path produces turns that are too sharp, there may be no valid path.

Assume now that given two points s and t on $\mathcal{P}$ that a valid path exists between them such that the turning constraint is also satisfied. In a sense, we are disallowing switchback paths within steep faces. This constraint produces a problem with our first algorithm since our approximated path segments are not of the same type as their corresponding shortest path segments. Hence, the existence of $\Pi(s,t)$ does not ensure the existence of a valid $\Pi'(s,t)$. In our second algorithm, we ensure that our approximated path segments are of the same type as the actual path segment which they approximate. However, our additional s'' segments that are used to join segments from consecutive faces may cause sharp turns. Therefore, the algorithm as stated, may not produce valid paths either.

Intuitively, if we add more and more Steiner points to the edges of $\mathcal{P}$, then the

chance of there being a valid turn path between s and t increases. However, it is non-trivial to determine the number of additional Steiner points that are required to ensure a path. It remains an open problem as to whether or not an efficient approximation can be produced for all valid $\Pi(s,t)$ paths when this additional turning constraint is added.

In Chapter 5 we present a parallel algorithm to compute approximations of shortest paths (Euclidean or weighted) between a source and target vertex on the surface of a polyhedron $\mathcal{P}$ [3]. The algorithm also applies to the one-to-all and few-to-all weighted shortest path problems as well. The algorithm is based on our previous sequential algorithms in that it constructs a graph on the surface of $\mathcal{P}$. The parallelization involves partitioning the terrain among processors and running Dijktra's algorithm on each processor, where each processor maintains only a local queue. Communication across partition boundaries occurs through message passing.

The data partitioning used in our experiments was that of the Multilevel Fixed Partition trees (see Nussbaum [97]). The MFP trees are a general form of data partitioning that recursively splits dense regions into levels according to a fixed (user-defined) threshold size. The partitioning incorporates an implicit mapping scheme which helps to minimize processor idle time and also reduce causality errors.

Through experimental analysis, we examined algorithmic-related performance factors such as varying the number of Steiner points which were placed on the edges of $\mathcal{P}$ as well as the computational intensity of the cost function (Euclidean or weighted). The tests have shown that the performance was not significantly affected by the variance in the number of Steiner points but that the more computationally-intensive cost functions do provide better speedup. It would be interesting to further investigate the effects that other cost functions have on the performance of the algorithm.

[3] The implementation was for the special case in which $\mathcal{P}$ is a terrain.

We have also determined that data-related factors such as relative source/target locations and data partitioning have a significant affect on the performance. The one-to-all variation of the algorithm has been shown to provide better performance than the one-to-one sessions. We obtained efficiencies of up to 27% on a 4x4 set of processors with the one-to-all tests whereas the one-to-one tests provided an average efficiency of about 15%. These speedups are consistent with results obtained from Träff [118]. In the case of few-to-all computations with weighted sources, we were able to improve upon the speedup to reach efficiencies of 64%. Through our tests, we were able to make the conclusion that the parallel algorithm is more worthwhile for large terrains since cost of communication is lower compared to the amount of computation to be done.

Our tests here have shown that the tile size (i.e., number of partition levels) plays an important role in determining the best partitioning strategy. For data that is uniformly distributed (non-clustered), the single level of partitioning has shown to provide performance comparable to the multi-level partitioning. An improvement in speedup with the multi-level partitioning was most noticeable on the terrain that was most clustered. The tile size that achieves "best" performance depends on the size and clusterization of the terrain. We have not determined the optimal tile size for our test terrains. As stated earlier, this value will most likely need to be determined through exhaustive experimentation since it depends highly on the data. However, from our experimental results we were able to conclude that the true benefits of the multi-level partitioning become apparent with a larger number of processors and for larger terrains with clustered areas.

One interesting topic for further study would be to try and formulate some kind of measure as to the density and/or sparseness of the data and propose a formula for determining an optimal (or near-optimal) tile size for that data. Also, as of yet,

the application has not been tested using other partitioning techniques. It would be interesting to contrast and compare other partitioning schemes with the MFP scheme used here.

Bibliography

[1] P. Adamson and E. Tick, "Greedy Partitioned Algorithms for the Shortest-Path Problem", *International Journal of Parallel Programming*, Vol. 20, No. 4, 1991, pp. 271-298.

[2] P. Adamson and E. Tick, "Parallel Algorithms for the Single-Source Shortest-Path Problem", *International Conference on Parallel Processing*, 1992, pp. III-346-350.

[3] P.K. Agarwal, B. Aronov, J. O'Rourke and C.A. Schevon, "Star Unfolding of a Polytope with Applications", *Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory*, Lecture Notes in Computer Science, Vol. 447, Berlin, 1990, pp. 251-263.

[4] P.K. Agarwal, S. Har-Peled, M. Sharir, and K.R. Varadarajan, "Approximating Shortest Paths on a Convex Polytope in Three Dimensions", *Journal of the ACM*, Vol. 44, 1997, pp. 567-584.

[5] P.K. Agarwal, B. Aronov, J. O'Rourke, and C. Schevon, "Star Unfolding of a Polytope with Applications", *SIAM Journal of Computing*, **26**, 1997, pp. 1689-1713.

[6] P.K. Agarwal and K.R. Varadarajan, "Approximating Shortest Paths on a Nonconvex Polyhedron", *Proceedings of the 38th IEEE Symp. on Foundations of Computer Science*, 1997.

[7] Ravindra K. Ahuja, Kurt Mehlhorn, James B. Orlin, and Robert E. Tarjan, "Faster Algorithms for the Shortest Path Problem", *Technical Report 193*, MIT Operations Research Center, MIT, 1988.

[8] D. Aingworth, C. Chekur, P. Indyk, and R. Motwani, "Fast Estimation of Diameter and Shortest Paths (Without Matrix Multiplication)", *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, 1996, pp. 547-553.

[9] L. Aleksandrov, M. Lanthier, A. Maheshwari and J.-R. Sack, "An ϵ-Approximation Algorithm for Weighted Shortest Path Queries on Polyhedral Surfaces", *14th European Workshop on Computational Geometry*, Barcelona, Spain, 1998, pp. 19-21.

[10] L. Aleksandrov, M. Lanthier, A. Maheshwari and J.-R. Sack, "An ϵ-Approximation Algorithm for Weighted Shortest Paths on Polyhedral Surfaces", *6th Scandinavian Workshop on Algorithm Theory*, LNCS 1432, Stockholm, Sweden, 1998, pp. 11-22.

[11] N. Alon, Z. Galil, and O. Margalit, "On the Exponent of the All Pairs Shortest Path Problem", *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, 1991, pp. 569-575.

[12] N. Alon, Z. Galil, O. Margalit, and M. Naor, "Witnesses for Boolean Matrix Multiplication and for Shortest Paths", *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, 1992, pp. 417-426.

[13] B. Aronov and J. O'Rourke, "Nonoverlap of the Star Unfolding", *Discrete and Computational Geometry*, **8**, New York, 1992, pp. 219-250.

[14] M.J. Atallah and D.Z. Chen, "Parallel Rectilinear Shortest Paths With Rectangular Obstacles", *Computational Geometry: Theory and Applications*, **1**, 1991, pp. 79-113.

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

[16] A. Baltsan and M. Sharir, "On the Shortest Paths Between Two Convex Polyhedra", *Journal of the ACM*, **35**, January 1988, pp. 267-287.

[17] Richard Bellman, "On a Routing Problem", *Quarterly of Applied Mathematics*, Vol **16**, No. 1, 1958, pp. 87-90.

[18] D.P. Bertsekas, F. Guerriero, and R. Musmanno, "Parallel Asynchronous Label-Correcting Methods for Shortest Paths", *Journal of Optimization Theory and Applications*, Vol. 88, No. 2, 1996, pp. 297-320.

[19] J. Canny and J. H. Reif, "New Lower Bound Techniques for Robot Motion Planning Problems", *Proceedings of the 28th IEEE Symp. on Foundations of Computer Science*, 1987, pp. 49-60.

[20] B. Chazelle, "Triangulating a Simple Polygon in Linear Time", *Discrete Computational Geometry*, Vol. 6, 1991, pp. 485-524.

[21] J. Chen and Y. Han, "Shortest Paths on a Polyhedron", *International Journal of Computational Geometry and Applications*, Vol. 6, 1996, pp. 127-144.

[22] D.Z. Chen, K.S. Klenk and H.T. Tu, "Shortest Path Queries Among Weighted Obstacles in the Rectilinear Plane", *Eleventh Annual ACM Symposium on Computational Geometry*, Vancouver, Canada, 1995, pp. 370-379.

[23] B.V.Cherkassky, A.V. Goldberg, and C. Silverstein, "Buckets, Heaps, Lists, and Monotome Priority Queues", *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms*, 1997, pp. 83-92.

[24] L. Paul Chew, "There Are Planar Graphs Almost as Good as the Complete Graph", *Journal of Computer and System Sciences*, 39, 1989, pp. 205-219.

[25] Y.J. Chiang and J.S.B. Mitchell, "Two-Point Euclidean Shortest Path Queries in the Plane", *Proceedings of the 10th Annual ACM Symposium on Discrete Algorithms*, 1999, pp. 215-224.

[26] J. Choi, J. Sellen and C.K. Yap, "Approximate Euclidean Shortest Path in 3-Space", *International Journal of Computational Geometry and Applications*, Vol. 7, No. 4, 1997, pp. 271-295.

[27] J. Choi, J. Sellen and C.K. Yap, "Precision-Sensitive Euclidean Shortest Path in 3-Space", *Proc. 11th Annual Symp. on Computational Geometry*, Vancouver, BC, 1995, pp. 350-359.

[28] J. Choi and C.K. Yap, "Rectilinear Geodesics in 3-Space", *Eleventh Annual ACM Symposium on Computational Geometry*, Vancouver, Canada, 1995, pp. 380-389.

[29] K.L. Clarkson, "Approximation algorithms for shortest path motion planning", *Proc. 19th Annual ACM Symp. Theory of Computing*, 1987, pp. 56-65.

[30] K.L. Clarkson, S. Kapoor and P.M. Vaidya, "Rectilinear Shortest Paths Through Polygonal Obstacles in $O(n \log^{3/2} n)$ time", *Proceedings of the 3rd Annual Symposium on Computational Geometry*, Waterloo, Canada, June 1987, pp. 251-257.

[31] E. Cohen, "Efficient Parallel Shortest-Paths in Digraphs with a Separator Decomposition", *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1993, pp. 57-67.

[32] D. Coppersmith and S. Winograd, "Matrix Multiplication Via Arithmetic Progressions", *Journal of Symbolic Computation*, Vol. 9, 1990, pp. 251-280.

[33] G. Das and G. Narasimhan, "Short Cuts in Higher Dimensional Space", *Proceedings of the 7th Annual Canadian Conference on Computational Geometry*, Québec City, Québec, 1995, pp. 103-108.

[34] M. de Berg, M. van Kreveld, B.J. Nilsson and M.H. Overmars, "Finding Shortest Paths in the Presence of Orthogonal Obstacles using a Combined L_1 and Link Metric", *Proc. of the 2nd Scandinavian Workshop on Algorithm Theory*, 1990, pp. 213-224.

[35] M. de Berg, M. van Kreveld and B.J. Nilsson, "Shortest Path Queries in Rectilinear Worlds", *International Journal of Computational Geometry and Applications*, Vol. 2, No. 3, 1992, pp. 287-309.

[36] M. de Berg, M. Katz, A. Frank van der Stappen and J. Vleugels, "Realistic Input Models for Geometric Algorithms", *Proceedings of the 13th Annual Symposium on Computational Geometry*, Nice, 1997, pp. 294-303.

[37] P.J.de Rezende, D.T. Lee and Y.F. Wu, "Rectilinear Shortest Paths With Rectilinear Barriers", *Proc. of the 1st Annual ACM Symposium on Computational Geometry*, 1985, pp. 204-213.

[38] E.W. Dijkstra, "A Note on Two Problems in Connection with Graphs", *Numerical Mathematics 1*, 1959, pp. 269-271.

[39] J.R. Driscoll, H.N. Gabow, R. Shrairman, and R.E. Tarjan, "Relaxed Heaps: An Alternative to Fibonacci Heaps with Applications to Parallel Computation", *Communications of the ACM*, **31**(11), 1988, pp. 1343-1354.

[40] H. ElGindy and M. Goodrich, "Parallel Algorithms for Shortest Path Problems in Polygons", *The Visual Computer*, No. 3, 1988, pp. 371-378.

[41] R.W. Floyd, "Algorithm 97 (SHORTEST PATH)", *Communications of the ACM*, **5**(6), 1962, pp. 345.

[42] Lestor R. Ford, Jr., and D.R. Fulkerson, "Flows in Networks", Princeton University Press, 1962.

[43] G.N. Frederickson, "Fast Algorithms for Shortest Paths in Planar Graphs, with Applications", *SIAM Journal of Computing*, **16**, 1987, pp. 1004-1022.

[44] G.N. Frederickson, "Planar Graph Decomposition and All-Pairs-Shortest-Paths", *Journal of the ACM*, **38**, 1991, pp. 162-204.

[45] M.L. Fredman and R.E. Tarjan, "Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms", *Journal of the ACM*, **34**(3), 1987, pp. 596-615.

[46] M.L. Fredman and D.E. Willard, "Surpassing the Information Theoretic Bound with Fusion Trees", *Journal of Computer and Systems Sciences*, **47**, 1993, pp. 424-436.

[47] M.L. Fredman and D.E. Willard, "Trans-dichotomous Algorithms for Minimum Spanning Trees and Shortest Paths", *Journal of Computer and System Sciences*, **48**, 1994, pp. 553-551.

[48] Harold N. Gabow and Robert E. Tarjan, "Faster Scaling Algorithms for Network Problems", *SIAM Journal of Computing*, **18**(5), 1989, pp. 1013-1036.

[49] G. Gallo and S.Pallottino, "Shortest Path Methods: A Unified Approach", *Mathematical Programming Study*, Vol. 26, 1986, pp. 38-64.

[50] L. Gewali, A. Meng, J. Mitchell and S. Ntafos, "Path Planning in $0/1/\infty$ Weighted Regions With Applications", *Proc. of the 4th Annual Symposium on Computational Geometry*, 1988, pp. 266-278.

[51] S.K. Ghosh and D.M. Mount, "An Output-Sensitive Algorithm for Computing Visibility Graphs", *SIAM Journal of Computing*, **20**, 1991, pp. 888-910.

[52] A.V. Goldberg, "Scaling Algorithms for the Shortest Path Problem", *SIAM Journal of Computing*, 24, 1995, pp. 494-504.

[53] M. Goodrich, S. Shauck and S. Guha, "Parallel Methods for Visibility and Shortest-Path Problems in Simple Polygons", *Algorithmica*, **8**, 1992, pp. 461-486.

[54] L. Guibas, J. Hershberger, D. Leven, M. Sharir and R.E. Tarjan, "Linear Time Algorithms for Visibility and Shortest Path Problems Inside Triangulated Simple Polygons", *Algorithmica*, **2**, 1987, pp. 209-233.

[55] L. Guibas and J. Hershberger, "Optimal Shortest Path Queries in a Simple Polygons", *Proc. of the 3rd ACM Symposium on Computational Geometry*, 1987, pp. 50-63.

[56] P.E. Hart, N.J. Nilsson, B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", *IEEE Transactions on System Science and Cybernetics*, SSC-4(2), 1968, pp. 100-107.

[57] S. Har-Peled, M. Sharir, and K.R. Varadarajan, "Approximating Shortest Paths on a Convex Polytope in Three Dimensions", *Proc. 12th Annual Symp. on Computational Geometry*, Philadelphia, PA, 1996, pp. 329-338.

[58] S. Har-Peled, "Approximate Shortest Paths and Geodesic Diameters on Convex Polytopes in Three Dimensions", *in Discrete & Computational Geometry*, Vol. 21, 1999, pp. 217-231.

[59] S. Har-Peled, " Constructing Approximate Shortest Path Maps in Three Dimensions", *SIAM Journal of Computing*, Vol. 28 (4), 999, pp. 1182-1197.

[60] R.V. Helgason, and D. Stewart, "One-to-One Shortest Path Problem: An Emperical Analysis with the Two-Tree Dijkstra Algorithm", *Computational Optimization and Applications*, Vol. 2, 1993, pp. 47-75.

[61] M.R. Henzinger, P. Klein, S. Rao, and S. Subramanian, "Faster Shortest-Path Algorithms for Planar Graphs", *Journal of Computer and System Sciences*, **55**(1), 1997, pp. 3-23.

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

[63] J. Hershberger and S. Suri, "Practical Methods for Approximating Shortest Paths on a Convex Polytope in $\Re^3$", *Proc. of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1995, pp. 447-456.

[64] M. Hribar, V. Taylor and D. Boyce, "Choosing a Shortest Path Algorithm", *Technical Report CSE-95-004*, Northwestern University, 1995.

[65] M. Hribar, V. Taylor and D. Boyce, "Performance Study of Parallel Shortest Path Algorithms: Characteristics of Good Decompositions", *13th Annual Conference on Intel Supercomputers User Group*, Albuquerque, NM, 1997.

[66] M. Hribar, V. Taylor and D. Boyce, "Parallel Shortest Path Algorithms: Identifying the Factors that Affect Performance", *Technical Report CPDC-TR-9803-015*, Northwestern University, 1998.

[67] M. Hribar, V. Taylor and D. Boyce, "Reducing the Idle Time of Parallel Shortest Path Algorithms", *Technical Report CPDC-TR-9803-016*, Center for Parallel and Distributed Computing, Northwestern University, 1998.

[68] D. Hutchinson, M. Lanthier, A. Maheshwari, D. Nussbaum, D. Roytenberg, J.-R. Sack, "Parallel Neighbourhood Modeling", *Proc. of the 4th ACM Workshop on Advances in Geographic Information Systems*, Minnesota, 1996, pp. 25-34.

[69] D.B. Johnson, "Efficient Algorithms for Shortest Paths in Sparse Networks", *Journal of the ACM*, **24**(1), 1997, pp. 1-13.

[70] S. Kapoor and S.N. Maheshwari, "Efficient Algorithms for Euclidean Shortest Path and Visibility Problems With Polygonal Obstacles", *Proc. of the 4th Annual ACM Symposium on Computational Geometry*, 1988, pp. 172-182.

[71] S. Kapoor, "Efficient Computation of Geodesic Shortest Paths", *STOC '99*, 1999, pp. 770-779.

[72] C. Kenyon and R. Kenyon, "How To Take Short Cuts", *Discrete and Computational Geometry*, Vol. 8, No. 3, 1992, pp. 251-264.

[73] V. Kumar, A. Grama, A. Gupta, and George Karypis, "Introduction to Parallel Computing: Design and Analysis of Algorithms", Ch. 7, Benjamin Cummings, 1993.

[74] D.G. Kirkpatrick, "Optimal Search in Planar Subdivisions", *SIAM Journal of Computing*, Vol. 12, No. 1, 1983, pp. 28-35.

[75] M. Lanthier, A. Maheshwari, J.-R. Sack, "Approximating Weighted Shortest Paths on Polyhedral Surfaces", *Technical Report TR-96-32*, School of Computer Science, Carleton University, Ottawa, December 1996.

[76] M. Lanthier, A. Maheshwari and J.-R. Sack, "Approximating Weighted Shortest Paths on Polyhedral Surfaces", *Proceedings of the 13th Annual ACM Symposium on Computational Geometry*, Nice, France, 1997, pp. 274-283.

[77] M. Lanthier, A. Maheshwari and J.-R. Sack, "Approximating Weighted Shortest Paths on Polyhedral Surfaces", *6th Annual Video Review of Computational Geometry*, Nice, France, June 1997.

[78] M. Lanthier, A. Maheshwari and J.-R. Sack, "Shortest Anisotropic Paths on Terrains", *ICALP 99*, LNCS 1644, Prague, 1999, pp. 524-533.

[79] M. Lanthier, A. Maheshwari and J.-R. Sack, "Approximating Weighted Shortest Paths on Polyhedral Surfaces", to appear *Algorithmica*, 1999.

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

[81] D.T. Lee, and F.P. Preparata, "Euclidean Shortest Paths in the Presence of Rectilinear Barriers", *Networks*, **14**, 1984, pp. 393-410.

[82] D.T. Lee, C.D. Yang and T.H. Chen, "Shortest Rectilinear Paths Among Weighted Obstacles", *International Journal of Computational Geometry and Applications*, Vol. 1, No. 2, 1991, pp. 109-224.

[83] A. Lingas, A. Maheshwari and J.R. Sack, "Optimal Parallel Algorithms for Rectilinear Link Distance Problems", *Algorithmica*, **14**, 1995, pp. 261-289.

[84] R. Lipton, D. Rose, and R.E. Tarjan, "Generalized Nested Dissection", *SIAM Journal on Numerical Analysis*, **16**, 1979, pp. 346-358.

[85] T. Lozano-Perez and M.A. Wesley, "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles", *Communications of the ACM*, Vol. 22, No. 10, 1979, pp. 560-570

[86] C. Mata and J. Mitchell, "A New Algorithm for Computing Shortest Paths in Weighted Planar Subdivisions", *Proceedings of the 13th Annual ACM Symposium on Computational Geometry*, 1997, pp. 264-273.

[87] K. Mehlhorn and S. Näher, " LEDA: a platform for combinatorial and geometric computing", *Communications of the ACM*, **38**, 1995, pp. 96–102.

[88] J.S.B. Mitchell, D.M. Mount and C.H. Papadimitriou, "The Discrete Geodesic Problem", *SIAM Journal of Computing*, **16**, August 1987, pp. 647-668.

[89] J.S.B. Mitchell, "An Algorithmic Approach to Some Problems in Terrain Navigation", *Journal of Artificial Intelligence*, **37**, 1988, pp. 171-201.

[90] J.S.B. Mitchell and C.H. Papadimitriou, "The Weighted Region Problem: Finding Shortest Paths Through a Weighted Planar Subdivision", *Journal of the ACM*, **38**, January 1991, pp. 18-73.

[91] J.S.B. Mitchell, "L_1 Shortest Paths Among Polygonal Obstacles in the Plane", *Algorithmica*, **8**, 1992, pp. 55-88.

[92] J.S.B. Mitchell, "Shortest Paths and Networks", *Handbook of Discrete and Computational Geometry*, J. Goodman and J. O'Rourke Eds., CRC Press LLC, Chapter 24, 1997, pp. 445-466.

[93] J.S.B. Mitchell, "Geometric Shortest Paths and Network Optimization", *Handbook on Computational Geometry* in print, J.-R. Sack and J. Urrutia Eds., Elsevier Science B.V., 1999.

[94] D.M. Mount, "On Finding Shortest Paths on Convex Polyhedra", *Technical Report 1495*, Department of Computer Science, University of Maryland, Baltimore, 1985.

[95] D.M. Mount, "Storing the Subdivision of a Polyhedral Surface", *Proc. of the 2nd Annual ACM Symposium on Computational Geometry*, New York, 1986, pp. 150-158.

[96] D.M. Mount, "The Number of Shortest Paths on the Surface of a Polyhedron", *SIAM Journal of Computing*, **19**, 1990, pp. 593-611.

[97] D. Nussbaum, "Spatial Modeling in Parallel Environments", *Phd. Thesis* in progress, School of Computer Science, Carleton University, Ottawa, Canada, 1999.

[98] J. O'Rourke, S. Suri and H. Booth, "Shortest Paths on Polyhedral Surfaces", extended abstract, Dept. Electrical Engineering and Computer Science, Johns Hopkins University, Baltimore, Maryland, September 1984.

[99] G. Pantziou, P. Spirakis, and C. Zaroliagis, "Efficient Parallel Algorithms for Shortest Paths in Planar Digraphs", *Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory*, 1990, pp. 288-300.

[100] C.H. Papadimitriou, "An Algorithm for Shortest Path Motion in Three Dimensions", *Information Processing Letters*, **20**, 1985, pp. 259-263.

[101] M. Pellegrini, "On Point Location and Motion Planning among Simplices", *SIAM Journal of Computing*, Vol. 25, No. 5, 1996, pp. 1061-1081.

[102] L. Polymenakos, and D.P. Bertsekas, "Parallel Shortest Path Auction Algorithms", *Parallel Computing*, Vol. 20, 1994, pp. 1221-1247.

[103] F.P. Preparata, "A New Approach to Planar Point Location", *SIAM Journal of Computing*, Vol. 10, No. 3, 1981, pp. 473-482.

[104] F.P. Preparata and M.I. Shamos, "Computational Geometry: An Introduction", Springer-Verlag, New York, 1985.

[105] K.V.S. Ramarao, and S. Venkatesan, "On Finding and Updating Shortest Paths Distributively", *Journal of Algorithms*, Vol. 13, 1992, pp. 235-257.

[106] R. Raman, "Priority Queues: Small Monotone, and Trans-dichotomous", *Lecture Notes in Computer Science*, Vol. 1136, 1996, pp. 121-137.

[107] R. Raman, "Recent Results on the Single-Source Shortest Paths Problem", *SICACT News*, Vol. 28, No. 2, 1997, pp. 81-87.

[108] N.C. Rowe, and R.S. Ross, "Optimal Grid-Free Path Planning Across Arbitrarily Contoured Terrain with Anisotropic Friction and Gravity Effects", *IEEE Transactions on Robotics and Automation*, Vol. 6, No. 5, 1990, pp. 540-553.

[109] A.A. Rula and C.J. Nuttall, "An Analysis of Ground Mobility Models(ANAMOB)", *Technical Report M-71-4*, U.S. Army Engineer Waterways Experiment Station, Vicksburg, MS, July 1971.

[110] C. Schevon and J. O'Rourke, "An Algorithm to Compute Edge Sequences on a Convex Polytope", *Technical Report JHU-89-3*, Department of Computer Science, Johns Hopkins University, Baltimore, Maryland, 1989.

[111] M. Sharir and A. Schorr, "On Shortest Paths in Polyhedral Spaces", *SIAM Journal of Computing*, **15**, 1986, pp. 193-215.

[112] M. Sharir, "On Shortest Paths Amidst Convex Polyhedra", *SIAM Journal of Computing*, **16**, 1987, pp. 561-572.

[113] R.G. Seidel, "On the All-Pairs-Shortest-Paths Problem", *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, 1992, pp. 745-749.

[114] M. Smid, "Geometric Spanners: Approximating the Complete Euclidean Graph", book in preparation, Univeristy of Magdeburg, Germany, 1999.

[115] S. Suri, personal communication, 1996.

[116] M. Thorup, "On RAM Priority Queues", *Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms*, 1996, pp. 59-67.

[117] M. Thorup, "Undirected Single Source Shortest Paths in Linear Time", *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, 1997.

[118] J.L. Träff, "An Experimental Comparison of two Distributed Single-Source Shortest Path Algorithms", *Parallel Computing*, Vol. 21, 1995, pp. 1505-1532.

[119] J.W.J. Williams, "Heapsort", *Communications of the ACM*, Vol. 7, No. 5, 1964, pp. 347-348.

[120] Y.F. Wu, P. Widmayer, M.D.F. Schlag and C.W. Wong, "Rectilinear Shortest Paths and Minimum Spanning Trees in the Presence of Rectilinear Obstacles", *IEEE Transactions on Computing*, C-36, 1987, pp. 321-331.

[121] Paradigm Group Webpage, School of Computer Science, Carleton University, http://www.scs.carleton.ca/~ gis.

[122] C.D. Yang, T.H. Chen and D.T. Lee, "Shortest Rectilinear Paths Among Weighted Rectangles", *Journal of Information Processing*, Vol. 13, No. 4, 1990, pp. 456-462.

[123] C.D. Yang, D.T. Lee and C.K. Wong, "On Bends and Lengths of Rectilinear Paths: A Graph-Theoretic Approach", *International Journal of Computational Geometry and Applications*, Vol. 12, No. 1, 1992, pp. 61-74.

[124] S. Yasutome, T. Masuzawa, Y. Tsujino and N. Tokura , "Shortest Paths Among Weighted Obstacles in a Plane", *Systems and Computers in Japan*, Vol. 24, No. 14, 1993, pp. 12-19.