Constrained Shortest Paths in Terrains and Graphs

by

Mustaq Ahmed

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2009

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Finding a shortest path is one of the most well-studied optimization problems. In this thesis we focus on shortest paths in geometric and graph theoretic settings subject to different feasibility constraints that arise in practical applications of such paths.

One of the most fundamental problems in computational geometry is finding shortest paths in terrains, which has many applications in robotics, computer graphics and Geographic Information Systems (GISs). There are many variants of the problem in which the feasibility of a path is determined by some geometric property of the terrain. One such variant is the *shortest descending path (SDP)* problem, where the feasible paths are those that always go downhill. We need to compute an SDP, for example, for laying a canal of minimum length from the source of water at the top of a mountain to fields for irrigation purpose, and for skiing down a mountain along a shortest route. The complexity of finding SDPs is open. We give a full characterization of the bend angles of an SDP, showing that they follow a generalized form of Snell's law of refraction of light. We also reduce the SDP problem to the problem of finding an SDP through a given sequence of faces, by adapting the sequence tree approach of Chen and Han for our problem. Our results have two implications. First, we isolate the difficult aspect of SDPs. The difficulty is not in deciding which face sequence to use, but in finding the SDP through a given face sequence. Secondly, our results help us identify some classes of terrains for which the SDP problem is solvable in polynomial time. We give algorithms for two such classes.

The difficulty of finding an exact SDP motivates the study of approximation algorithms for the problem. We devise two approximation algorithms for SDPs in general terrains—these are the first two algorithms to handle the SDP problem in such terrains. The algorithms are robust and easy-to-implement. We also give two approximation algorithms for the case when a face sequence is given. The first one solves the problem by formulating it as a convex optimization problem. The second one uses binary search together with our characterization of the bend angles of an SDP to locate an approximate path.

We introduce a generalization of the SDP problem, called the *shortest gently descending path (SGDP)* problem, where a path descends but not too steeply. The additional constraint to disallow a very steep descent makes the paths more realistic in practice. For example, a vehicle cannot follow a too steep descent—this is why a mountain road has hairpin bends. We give two easy-to-implement approximation algorithms for SGDPs, both using the Steiner point approach. Between a pair of points there can be many SGDPs with different number of bends. In practice an SGDP with fewer bends or smaller total turn-angle is preferred. We show using a reduction from 3-SAT that finding an SGDP with a limited number of bends or a limited total turn-angle is hard. The hardness result applies to a generalization of the SGDP problem called the *shortest anisotropic path* problem, which is a well-studied computational geometry problem with many practical applications (e.g., robot motion planning), yet of unknown complexity.

Besides geometric shortest paths, we also study a variant of the shortest path problem in graphs: given a weighted graph G and vertices s and t, and given a set X of forbidden paths in G, find a shortest s-t path P such that no path in X is a subpath of P. Path P is allowed to repeat vertices and edges. We call each path in X an *exception*, and our desired path a *shortest exception avoiding path*. We formulate a new version of the problem where the algorithm has no

a priori knowledge of X, and finds out about an exception $x \in X$ only when a path containing x fails. This situation arises in computing shortest paths in optical networks. We give an easy-to-implement algorithm that finds a shortest exception avoiding path in time polynomial in $|G|$ and $|X|$. The algorithm handles a forbidden path using vertex replication, i.e., replicating vertices and judiciously deleting edges so as to remove the forbidden path but keep all of its subpaths. The main challenge is that vertex replication can result in an exponential number of copies of any forbidden path that overlaps the current one. The algorithm couples vertex replication with the "growth" of a shortest path tree in such a way that the extra copies of forbidden paths produced during vertex replication are immaterial.

Acknowledgments

All praise is due to Allah, the Lord of the Worlds, the Beneficent, the Merciful.

I express my gratitude to my supervisor, Anna Lubiw, for her encouragement and support for the thesis. Her insightful research ideas and invaluable feedback played a vital role in my successful completion of the work. It was a great pleasure working with her. I owe to her also for her generous help and advice on issues not related to research.

I am thankful to my thesis committee members, Timothy M. Chan, Alejandro López-Ortiz, Joseph Cheriyan and Mark Keil, for their invaluable comments that helped to improve the thesis. I would like to extend my thanks to Anil Maheshwari for his research ideas, to Erik Demaine for his useful suggestions on extending my work.

I am indebted to my parents for their sincere inspiration and unconditional support at every step in my life. I am grateful to my brothers and my sister for their guidance and their encouragement, my wife for her love and patience, and my son and my daughter for their soothing company.

I am also thankful to all my teachers, specially M. Kaykobad who introduced me to the world of research, and provided me with strong encouragement during my undergrad years.

Finally I thank everyone who has contributed to this work in any way.

Contents

List of Figures x

1 Introduction 1

1.1 Our results and organization of the thesis . . . 2

2 Background on Shortest Paths in Terrains 5

2.1 Preliminaries . . . 5

2.2 Basic approaches . . . 6

2.2.1 Continuous Dijkstra approach . . . 6

2.2.2 Sequence tree approach . . . 8

2.2.3 Steiner point approach . . . 10

2.3 Relevant shortest path problems in terrains . . . 11

2.3.1 The weighted region problem . . . 11

2.3.2 The shortest anisotropic path problem . . . 17

3 Shortest Descending Paths: Towards an Exact Algorithm 20

3.1 Introduction . . . 20

3.2 Terminology . . . 21

3.3 Characteristics of a shortest descending path . . . 22

3.3.1 Similarities with a geodesic path . . . 23

3.3.2 Uniqueness of an SDP through given faces . . . 28

3.3.3 An algorithm for SDPs through given faces using convex optimization . . . 31

3.3.4 Generalized Snell's Law . . . 32

3.3.5 Complete characterization . . . 37

3.3.6 An algorithm to trace an LSDP along a given initial direction . . . 43

3.3.7 Another algorithm for SDPs through given faces . . . 43

3.4 Sequence tree approach for SDPs . . . 47
3.4.1 Constructing a sequence tree . . . 49
3.4.2 Correctness of our construction . . . 51
3.4.3 Problems in approximating SDPs using sequence trees . . . 55
3.5 Polynomial time algorithms for special terrains . . . 57
3.5.1 Algorithm for pseudo-convex terrains . . . 57
3.5.2 Algorithm for pseudo-orthogonal terrains . . . 59
3.6 Conclusion . . . 60

4 Approximation Algorithms for Shortest Descending Paths 62
4.1 Introduction . . . 62
4.2 Placing the Steiner points for SDPs . . . 63
4.2.1 Problems in placing Steiner points independently . . . 64
4.2.2 Problems in placing Steiner points in geometric progression . . . 65
4.3 Using uniform Steiner points . . . 66
4.3.1 Algorithm . . . 66
4.3.2 Correctness and analysis . . . 68
4.4 Using Steiner points in geometric progression . . . 71
4.4.1 Algorithm . . . 71
4.4.2 Correctness and analysis . . . 73
4.5 Conclusion . . . 76

5 Shortest Gently Descending Paths 78
5.1 Introduction . . . 78
5.2 Terminology . . . 80
5.3 Properties of SGDPs . . . 82
5.4 Approximation using uniform Steiner points . . . 86
5.4.1 Algorithm . . . 86
5.4.2 Correctness and analysis . . . 87
5.5 Approximation using non-uniform Steiner points . . . 89
5.5.1 Correctness and analysis . . . 90
5.6 Hardness of SGDPs with few bends . . . 92
5.6.1 Overview . . . 92
5.6.2 Elementary gadgets . . . 94

5.6.3 The Path Bundle and its labeling . . . 95
5.6.4 Intermediate gadgets . . . 96
5.6.5 Main gadgets . . . 99
5.6.6 Correctness . . . 100
5.6.7 Constructing a terrain without vertical faces . . . 101
5.7 Hardness of SGDPs with limited total turn-angle . . . 102
5.8 Conclusion . . . 103

6 Shortest Paths avoiding Forbidden Subpaths 105
6.1 Introduction . . . 105
6.1.1 Motivation . . . 106
6.1.2 Preliminaries . . . 107
6.1.3 Relaxation . . . 108
6.1.4 Related work . . . 109
6.2 A generic algorithm for a shortest s-t path . . . 110
6.2.1 Modifying the graph . . . 111
6.2.2 Constructing the tree . . . 112
6.3 Correctness and analysis . . . 113
6.3.1 Justifying the graph modification . . . 113
6.3.2 Justifying the tree construction . . . 114
6.3.3 Analysis of timing . . . 115
6.3.4 Relaxing the edges efficiently . . . 116
6.4 Algorithms for specific graph classes . . . 117
6.5 Computing shortest paths to all vertices . . . 119
6.6 Handling a weaker oracle . . . 120
6.7 Conclusion . . . 122

7 Conclusion 124

References 133

List of Figures

2.1 Handling candidate intervals in the continuous Dijkstra approach. 7
2.2 One-angle one-split property for sequence tree for shortest Euclidean paths. . . 9
2.3 Placing Steiner points non-uniformly. 11
2.4 Complications with shortest paths in the weighted region problem. 12
2.5 Non-uniformly placed Steiner points. 14
2.6 Utilizing Snell's law with approximate paths. 15
2.7 Maintaining the list $I_{e,e'}$ in the Bushwhack algorithm. 16
2.8 Links between the Steiner points along two bisectors. 17
2.9 The model of Rowe and Ross for shortest anisotropic paths. 18

3.1 General legend for all figures in this section. 23
3.2 An LSDP visiting a face twice. 24
3.3 Entering and exiting angles. 25
3.4 Proving that $|pq'| + |q'r| < |pq| + |qr|$ for any q' is inside $\triangle pqr$. 25
3.5 Showing that an unfolded LSDP is not a straight line segment. 26
3.6 A terrain in which an SDP is very different from a shortest path. 27
3.7 Two descending paths that follow Lemma 3.4, but of different lengths. 28
3.8 Proving strict convexity of $\mathcal{L}(x_1, x_2, \ldots, x_k)$. 29
3.9 Expressing $\lambda_{i,i+1}$ in terms of ν_i and ν_{i+1}. 33
3.10 Path Q with all intermediate nodes at height $h(p_c)$. 34
3.11 Proving that two SDPs from s are diverging. 38
3.12 Defining δ for the continuity argument in Lemma 3.12. 39
3.13 Two LSDPs from s with $h(p_1) > h(q_1)$ and $h(p_2) < h(q_2)$. 40
3.14 Proving the existence of a segment that extends the LSDP P_0 in Lemma 3.13. . 40
3.15 Constructing the paths $P = P_0$, P_1, P_2, ..., $P_d = Q$ to prove that Q is longer than P in the proof of Theorem 3.2. 42

3.16 The SDP from p_{i-1} to t does not cross $P(p_{i-1}, v)$ when $v \neq w$. 46
3.17 The sequence tree for a part of the terrain. 48
3.18 Expanding an edge-node. 51
3.19 Proving that the face/vertex sequence $\sigma_1 f_2$ can be discarded when P_2 is shorter than P_1. 52
3.20 Figure 3.19 repeated for convenience. 56
3.21 LSDPs in a pseudo-convex terrain. 58
3.22 Impossible subpaths of an LSDP in a pseudo-orthogonal terrain. 59

4.1 Problems with independently placed Steiner points. 65
4.2 An SDP that comes close to a vertex $O(n)$ number of times. 66
4.3 Finding an SDP from s to an interior point v of a face and an edge. 67
4.4 Bounding $|p_i p_i'|$ for uniform Steiner points. 69
4.5 Vicinity of a vertex. 72
4.6 Bounding $|p_i p_i'|$ for non-uniform Steiner points. 74

5.1 Descending gently towards a steep direction. 78
5.2 The cone defined by the lines at an angle ψ with a vertical line at p, and a steep line segment pq. 81
5.3 Critical paths from a to b in face f. 84
5.4 A critical path from a vertex a that is locally sharp in $f = f_0$, but not in f_k. . 84
5.5 An SGDP between two locally sharp vertices a and b in f that goes through $\Theta(d)$ other faces. 85
5.6 Clause Filters along the Path Bundle. 93
5.7 A Splitter. 94
5.8 A Blocker. 95
5.9 The Path Bundle in the initial terrain ($n = 3$). 96
5.10 A Tripler and a Reverse Tripler. 97
5.11 A Shuffler. 98
5.12 The structure of the Literal Filter for $\bar{b_2}$ ($n = 4$). 99
5.13 Determining the values of ψ, and the slope of f_1 when f_1 is not vertical. 102

6.1 Shortest paths and shortest exception avoiding paths in a graph. 110
6.2 Modifying G_{i-1} to G_i. 112
6.3 The graph in Figure 6.2(c) without the incoming edges of the intermediate vertices of x_i. 117

Chapter 1

Introduction

The *shortest path problem* is one of the most well-studied optimization problems, where the goal is to find a path of smallest possible *length* among all the paths that go from one point to another in a given *domain*. The classical domain for the problem is a *graph* with a *weight* for each of the edges, and a well-studied definition of the length of a path in the graph is the total weight of the edges in the path. Shortest paths in graphs have many practical applications. One popular example is finding the fastest driving route between two points in a city, where the graph corresponds to the road map of the city, with nodes representing intersections, and the weight of an edge between two intersections representing the time needed to drive from one of the intersections to the other. Other applications of the graph shortest path problem include network routing (e.g., Wang et al. [100]) and VLSI design [19, Section 23.3]. The problem is important for a wide range of other applications that do not involve "physical" paths. Arbitrage (i.e., exploiting the discrepancies in currency exchange rates [34, page 615]), algorithmic game theory (e.g., determining pricing policy for resources in the Internet [56]), industrial automation (e.g., Mo et al. [72] and Wilhelm et al. [101]) and operations research (e.g., Hillier and Lieberman [58]) are examples of such applications.

The shortest path problem has also been studied in geometric settings, where the problem domain is specified by a set of geometric objects. For example, one of the simplest geometric shortest path problems is: given a set of obstacles in the plane, find a path of smallest Euclidean length that *avoids* all the obstacles. Unlike the case of a graph, a geometric domain is continuous in nature, i.e., we can modify a feasible path into another in a continuous manner. Many special cases of the geometric shortest path problem can be defined by choosing various parameters of the problem, like the dimension of the domain (e.g., 2D or 3D), the metric used for path length (e.g., Euclidean or L_1 distance), the presence and type of obstacles (static or moving), and so on. The survey by Mitchell [69] discusses many of these special cases. Finding shortest paths in *polyhedral terrains* is one such problem, which has been studied extensively in computational geometry for a long time. This problem has many practical applications such as robotics (e.g., Rowe and Ross [81]), computer graphics (e.g., Lee et al. [65] and Surazhsky et al. [93]), and geographic information systems (e.g., Floriani et al. [46] and Upchurch et al. [97]).

In this thesis we focus on shortest paths in terrains and graphs subject to different feasibility constraints arising in practical applications of such paths. An example such a constraint for the case of terrain shortest paths is the *steepness*: when going downward from a mountain,

a car cannot follow a steep direction for safety reasons. On a mountain road, hairpin bends are used so that a car can avoid a steep direction. For terrain shortest paths, we consider two geometric constraints regarding relative heights of points in a path. In graphs, we consider a combinatorial constraint faced by paths in an optical network.

Our goal here is to devise polynomial time algorithms that compute exact shortest paths if possible, and to devise approximation algorithms otherwise. The term "polynomial time" means polynomial in the size of input instance. One issue regarding shortest paths in a geometric setting is that Euclidean path lengths, which involve sums of square roots, cannot be computed exactly from point coordinates, and no bound is known on the number of bits that are needed to guarantee an accurate comparison between two path lengths. See Problem 33 in the Open Problems Project [40] for a discussion of this issue. Because of this, it is conventional to use the real RAM model of computation, where real numbers can be stored and operated on at unit cost. Details on this model can be found in Preparata and Shamos [78, Section 1.4].

Another issue regarding algorithms for computational geometry problems is that the running time of such algorithms often depend on various geometric parameters of the input instance (besides the size of the input), and in many cases these parameters reflect the inherent simplicity or difficulty of the input instance—see Afshani et al. [1] for a few examples and de Berg et al. [36] for a discussion. The geometric parameters that determine the running times of our algorithms for terrain shortest paths include the lengths and inclinations of the terrain edges, and the size of the "narrowest" terrain face.

For the case of an approximation algorithm for geometric shortest paths, we are interested in a $(1+\epsilon)$-approximation algorithm, which takes as input an instance of an optimization problem and a parameter $\epsilon > 0$, and returns, in time polynomial in the size of input, $\frac{1}{\epsilon}$ and some geometric parameters, a solution that is within a factor $(1+\epsilon)$ of the optimal solution. A $(1+\epsilon)$-approximation algorithm for a shortest path problem returns a path whose length (in appropriate path length metric) is at most $(1+\epsilon)$ times the length of a shortest path.

1.1 Our results and organization of the thesis

This thesis examines constrained shortest paths in terrains in Chapters 3 to 5, and constrained shortest paths in graphs in Chapter 6.

Before that, we provide in Chapter 2 some background on the terrain shortest path problem. More precisely, we present three elementary approaches that have been used for finding shortest paths in terrains, and discuss their application to two terrain shortest paths problems closely related to the problems we study in Chapters 3 to 5.

Note that the background needed for our graph shortest path problem in Chapter 6 appears in that same chapter. This is because this background is needed only in that chapter.

In Chapter 3 we study the the *shortest descending path (SDP)* problem, where the feasible paths are those that always go downhill. The problem was introduced by de Berg and van Kreveld [37]. We need to compute an SDP, for example, for laying a canal of minimum length from the source of water at the top of a mountain to fields for irrigation purpose, and for skiing down a mountain along a shortest route. There is no known polynomial time algorithm for the SDP problem, nor is the problem known to be NP-hard. Previously known exact algorithms

for the problem were for two special classes of terrains, and they provided very little insight on the "structure" of SDPs in a general terrain. It was a mystery why the problem is so easy for those two special classes of terrain yet unsolved for the general case. We explore some structural properties of an SDP, and show that the path obeys a generalized form of Snell's law of refraction of light. We also reduce the SDP problem to the apparently simpler problem of finding an SDP through a given sequence of faces. Our results have two implications. First, we isolate the difficult aspect of SDPs. We show that the difficulty is not in deciding which face sequence to use, but in finding the SDP through a given face sequence. Secondly, our results help us identify some classes of terrains for which the SDP problem is solvable in polynomial time. We give algorithms for two such classes.

Chapter 3 also gives two approximation algorithms for the case when a face sequence is given. The first one solves the problem by formulating it as a convex optimization problem. The second one utilizes the structural properties of an SDP in binary search to locate an approximate path.

Because finding an exact SDP in general terrains seems to be a difficulty problem, it is natural to look for approximation algorithms. In Chapter 4 we devise two approximation algorithms for SDPs in general terrains. These are the first two algorithms to handle the problem in such terrains. Both the algorithms solve the problem by transforming the geometric shortest path problem into a graph shortest path problem. The idea is to discretize the terrain by adding many points, called the Steiner points, along the terrain edges, and then construct a graph of those Steiner points in such a way that any path in the graph approximates nearby paths in the terrain. The resulting algorithms are robust and easy-to-implement. The running times of the two algorithms are not comparable because one is faster in terms of the edge inclinations of the terrains, while the other is faster in terms of other geometric parameter as well as n.

In Chapter 5 we introduce a generalization of the SDP problem, called the *shortest gently descending path (SGDP)* problem, where a path descends but not too steeply. The additional constraint to disallow a very steep descent makes the paths more realistic when, for example, a vehicle cannot follow a too steep descent on the slope of a mountain. We explore some properties of SGDPs, and give two easy-to-implement approximation algorithms for the problem. These algorithms transform the problem into a graph shortest path problem in the same manner as in Chapter 4.

There can be infinitely many SGDPs between two points, and they can have different number of bends. In practice, for example in robot motion planning, an SGDP with fewer bends is preferred because at every bend a robot will need extra time and energy to change its direction. We show in Chapter 5 using a reduction from 3-SAT that finding an SGDP with a limited number of bends is NP-hard. We give a similar result for the case when, instead of a limit on the number of bends, we have a limit on the total turn-angle, i.e., on the *amount of rotation* made by a robot at all bends. Our hardness results apply to the more general problem of finding a *shortest anisotropic path* in a terrain, a well-studied computational geometry problem for which neither a polynomial time algorithm nor a hardness result is known.

In Chapter 6 we study a variant of the shortest path problem in graphs: given a weighted graph G and vertices s and t, and given a set X of forbidden paths in G, find a shortest s-t path P such that no path in X is a subpath of P. Path P is allowed to repeat vertices

and edges. We call each path in X an *exception*, and our desired path a *shortest exception avoiding path*. We formulate a new version of the problem where the algorithm has no a priori knowledge of X, and finds out about an exception $x \in X$ only when a path containing x fails. This situation arises in computing shortest paths in optical networks. We give an easy-to-implement algorithm that finds a shortest exception avoiding path in time polynomial in $|G|$ and $|X|$. The algorithm handles a forbidden path by replicating vertices and judiciously deleting edges in such a way that the forbidden path is removed from the graph but all of its subpaths remain. The main challenge is that such replication of vertices can result in an exponential number of copies of any forbidden path that overlaps the current one. Our algorithm couples vertex replication with the "growth" of a shortest path tree in such a way that the extra copies of forbidden paths produced during vertex replication are immaterial. The algorithm depends on a (conventional) shortest path algorithm, and works for directed acyclic graphs, graphs with non-negative edge-weights, and graphs with negative edge-weights but no negative weight cycle.

Chapter 2

Background on Shortest Paths in Terrains

This chapter gives background on shortest paths in terrains. This will be helpful in understanding our results in Chapters 3 to 5. Note that the background on graph shortest paths can be found in Chapter 6.

One of the earliest works on shortest paths in polyhedral surfaces is the one by Sharir and Schorr [88], which finds a shortest Euclidean path on a convex polyhedron in $O(n^3 \log n)$ time. Since then, many papers have focused on exact and approximation algorithms for the problem and its variants [2, 3, 13, 14, 15, 27, 28, 31, 52, 53, 54, 55, 61, 63, 70, 71, 75, 87, 92, 98]. In this chapter we present three basic approaches used in finding different variants of shortest paths on terrains. The approaches are: the continuous Dijkstra approach [70], the sequence tree approach [28], and the Steiner point approach [76]. (There is another interesting idea called the wavefront propagation approach [61]. We do *not* cover this approach in this section, as the approach lacks certain details. See Schreiber and Sharir [87] for a brief discussion on this issue.) We then discuss the application of these approaches to two terrain shortest path problems: the weighted region problem and the shortest anisotropic path problem. We choose these two variants because they are closely related to the problems we study in Chapters 3 to 5.

We give definitions of elementary terms in Section 2.1, and then describe the basic approaches and their applications in Sections 2.2 and 2.3 respectively.

2.1 Preliminaries

A terrain is a 2D surface in 3D space with the property that every vertical line intersects it in a point [38]. We adapt a slightly generalized definition that every vertical line intersects a terrain in a *single line segment*, in order to allow a terrain with vertical edges and faces. We consider triangulated terrains because any terrain can be triangulated in $O(n)$ time [26]. For the terrain shortest path problem, we are given two points s and t on the terrain, and we want compute a shortest path from s to t. We make s and t vertices of the terrain in a trivial way.

Let n be the number of vertices in the terrain. By *Euler's formula* [38, Page 29], the terrain has at most $3n$ edges, and at most $2n$ faces.

We use "shortest paths" to denote "shortest Euclidean paths" in this chapter. A *geodesic path* (also called a *locally shortest path*) between two nodes is a path that cannot be shortened by slight perturbation of the intermediate nodes. It is known that a geodesic path becomes a straight line segment when the faces crossed by the path are unfolded onto a plane [70]. One important issue is that it is practically impossible to store the exact Euclidean length of an unfolded geodesic path—representing the lengths with a floating point number introduces errors, as does rotating a face around an edge for unfolding [2, 20, 21]. In fact, given two geodesic paths through two different face sequences, we do not know how to compute, with a polynomially bounded number of bits, which of the paths is shorter. It means that shortest path problems are not known to lie in NP. It also means that algorithms assume a model of computation, e.g., the real RAM model, in which square roots can be computed accurately in constant time.

For ease of discussion, we use the following convention in this chapter and also in the chapters on graph shortest paths (Chapters 3 to 5). The term "edge" denotes an edge of a (triangular) terrain face, and the term "segment" denotes a line segment of a path. Similarly, an endpoint of an edge is called a "vertex", while an endpoint of a segment is called a "node".

2.2 Basic approaches

2.2.1 Continuous Dijkstra approach

One popular approach used in finding a shortest path on a polyhedral surface or a shortest obstacle-avoiding path in 2D is the continuous Dijkstra approach. Although similar ideas were used before [74, 88], Mitchell, Mount and Papadimitriou [70] formalized the idea for the first time in their algorithm for the *discrete geodesic problem*, where the goal is to find a shortest path from s to any other vertex in a polyhedral surface. We will now give details of this approach because this will make it easy to understand the sequence tree approach (discussed in the next section), which we will use in Chapter 3.

As the name suggests, the continuous Dijkstra approach is similar to Dijkstra's algorithm [44] for shortest paths in graphs: intuitively, a "signal" starts propagating from s over the surface at a constant speed, so that the moment a point on the surface receives the signal for the first time determines its shortest distance from s. In the continuous Dijkstra approach, the edges of the surface behave like nodes of a graph, except that there is no unique distance from s to an edge. To keep track of the distances of the points in an edge, each edge is subdivided into *intervals of optimality* such that the shortest path to any point in an interval has the same discrete structure, passing through the same sequence of vertices and edges. These intervals are determined using a set of dynamic intervals, called the *candidate intervals*, each of which is a superset of some (possibly empty) interval of optimality. For each edge e, and each face f adjacent to e, the algorithm maintains the property that the candidate intervals for the shortest paths crossing f to reach e are non-overlapping. The candidate intervals are created in the order of their distances from s, starting with a candidate interval to cover each edge opposite to s. At each step, one candidate interval is propagated onward to the next

face to create (at most) two other intervals (Figure 2.1(a)). As the algorithm proceeds, the candidate intervals on edge e shrink when new candidate intervals on e are created, and they finally converge to the corresponding intervals of optimality in the order of their endpoints' distances from s. Dijkstra's algorithm on a graph works in a very similar way: an over-estimate of the nodes' distances from s are assigned to the nodes, which are gradually decreased to the actual distances in increasing order of the actual distances.

Before giving further details of the approach, we will mention two properties of a geodesic path that are utilized here. The first property is that a geodesic path becomes a straight line segment when the faces crossed by the path are unfolded onto a plane. This is obvious because otherwise the path could be made shorter by moving the bend-point in a certain direction. Another property is that two geodesic paths from s through two different face sequences cannot intersect each other at an interior point of a face (otherwise, if they intersect at an interior point p of a face, both paths could be made shorter by first swapping their prefixes at p and then taking shortcuts around p.)

The algorithm uses a few simple data structures as follows. A list L of all candidate intervals is maintained. Each candidate interval stores its left and right endpoints on the corresponding edge, the nearest point in it from s—called its *frontier point*, and the candidate interval that precedes it on the path from s. For each edge e, and each face f adjacent to e, a list of candidates intervals for the shortest paths that cross f to reach e is maintained; the intervals in this list are sorted in the order they appear along e (note that these candidate intervals are disjoint). There is a priority queue Q of points that are labeled with the points' best known distances from s. Only the endpoints and the frontier points of candidate intervals, and the vertices are pushed into the queue.

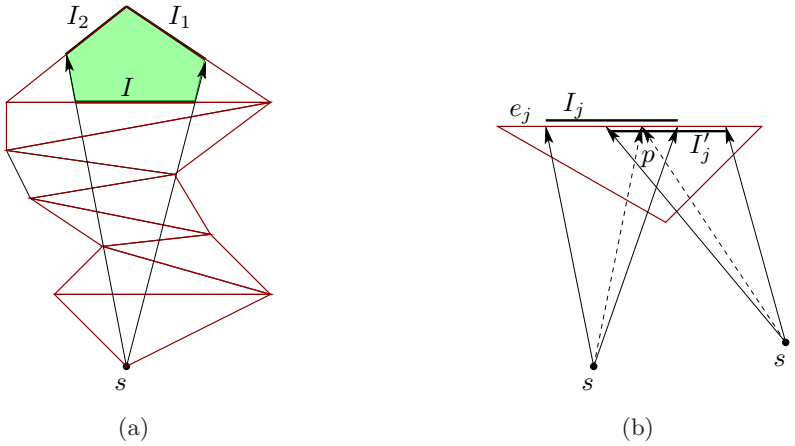


Figure 2.1: Handling candidate intervals in the continuous Dijkstra approach.

The algorithm is as follows. In the beginning, s is labeled with zero. For each edge opposite to s, a candidate interval covering the entire edge is created and stored in L. The frontier points and the endpoints of these intervals are pushed into Q. In the main loop, the point with smallest label is removed from Q, and if this is a frontier point of an interval I, the paths bounding I are traced forward into the next face to create two new intervals I_1 and I_2 on edges e_1 and e_2 respectively, as shown in Figure 2.1(a). Each new interval I_j, $j \in \{1, 2\}$, is then stored in L, and its frontier point and the endpoints are pushed into Q. However, if I_j overlaps with

existing candidate intervals on e_j, some of the involved intervals are trimmed in the following way before the mentioned points are pushed into Q. Using the location of frontier points of the intervals preceding the existing candidate intervals on e_j, the position of I_j with respect to existing candidate intervals I'_j on e_j is determined. Then if one or more intervals I'_j are dominated by I_j, they are deleted. Otherwise, a point $p \in I_j \cap I'_j$ is determined for which the shortest paths through the face sequences used by I_j and I'_j are of equal lengths, as shown in Figure 2.1(a). As per the figure, the part of I_j to the right of p and the part of I'_j to the left of p are then trimmed.

A geodesic path can bend at a non-convex vertex (i.e., at a vertex at which the total of the face angles is more than 2π), and propagating an interval of optimality along straight lines through such a vertex v does not cover all face angles around v. To handle this issue, new candidate intervals are created in the "uncovered" face angles around v, in the same way new candidate intervals are created for s. However, unlike s the distance of a non-convex vertices is positive, and this distance is determined by considering the non-convex vertex as an endpoint of some interval of optimality (as in the case of other vertices).

The algorithm takes $O(n^2 \log n)$ time and $O(n^2)$ space to build a tree of intervals of optimality because there are $O(n^2)$ intervals of optimality (and candidate intervals). Using this tree the shortest path from s to any point on the surface can be determined in $O(k + \log n)$ time, where k is the length of the path.

2.2.2 Sequence tree approach

Chen and Han [28] introduced a new approach to find a shortest path on a polyhedral surface. They construct a tree, called a *sequence tree*, which captures all the possible edge/face sequences used by the shortest paths from s. Each node in a sequence tree represents either a vertex of the polyhedron or a portion of an edge. We will call the portion of an edge represented by a node of the tree an *interval* because it is very similar to the intervals of optimality used by Mitchell, Mount and Papadimitriou [70]: all points in an interval are reachable from s using geodesic paths through a common sequence of faces and vertices. However, unlike the intervals of optimality, the intervals in the sequence tree are not only for shortest paths—some of them are for non-shortest geodesic paths and they *may* get deleted as the algorithm proceeds. Another difference is that the intervals lying on an edge are not disjoint from one another.

The tree is rooted at a node representing s, and the path from the root to a node v in the tree corresponds to a sequence σ of intervals that can be traversed by a geodesic path from s to any point in the the interval I represented by v. In other words, after unfolding all the faces involved in σ into a plane, the straight line segment connecting s to any endpoint of I stabs all the intervals in I. It is straightforward to construct a sequence tree that stores edge sequences of *all* geodesic paths: to grow the tree at a node for interval I on e, take the wedge of geodesic paths arriving at I and extend it via straight lines into the unfolded face on the other side of e, adding new tree nodes for the (one or two) intervals thus created (Figure 2.1(a) shows the same scenario, but for the intervals of optimality). The tree may be truncated at depth $2n$ because any shortest path traverses at most $2n$ faces, but even so, the tree can be exponentially large because each node can have two children. Chen and Han give a property of geodesic paths, called the *one-angle one-split* property, that can be used to prune the tree to size $O(n^2)$: for any two intervals I_1 and I_2 lying on edge e in face f, if the wedges of both the

intervals split at vertex $v \in f$ opposite to e (Figure 2.2), one of the four split intervals cannot lead to a shortest path. More precisely, if the distance of v from s through the face sequence determined by I_1 is more than its distance from s through the face sequence determined by I_2, then no shortest path crosses both I_1 and I_{12} in Figure 2.2, because the longer geodesic path cannot yield shorter paths on both sides of v. This implies that it is sufficient to store one split for each face angle. The number of leaf nodes in the tree then becomes $O(n)$ because each of the $O(n)$ face angles contributes at most one extra "branch" in the tree. Since there are $O(n)$ levels, the total number of nodes becomes $O(n^2)$.

Note that extending a wedge through a non-convex vertex is not easy, since a geodesic path can bend at a non-convex vertex. Chen and Han handle this issue by treating the non-convex vertices as a pseudo-source: they make a new node in the sequence tree each time a non-convex vertex splits a wedge, and then they start a new set of wedges of geodesic paths around that vertex. The new node represents the face angle encountered by the "arriving" wedge, and stores the vertex's distance from s through the corresponding face/vertex sequence.

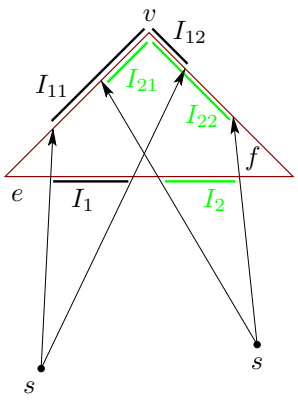


Figure 2.2: One-angle one-split property for sequence tree for shortest Euclidean paths.

The number of nodes can be decreased further to $O(n)$ by deleting the nodes with only one child, i.e., by removing the intervals for which there is only one "child interval". This is done during the time the tree is being expanded level-wise, resulting in an $O(n^2)$ time and $O(n)$ space algorithm to build the tree.

After the tree is constructed, the shortest path to any vertex can be determined in $O(n)$ time as follows. There can be at most $O(n)$ nodes in the tree for a particular vertex v—one for each face angle at v. After locating among them the node with the smallest distance from s, the sequence of faces/vertices is determined by traversing the tree upward to the root. The shortest path is then computed from the sequence in a trivial manner.

The preprocessing phase of the sequence tree approach is faster than that of the continuous Dijkstra approach. The query phase of the sequence tree approach is slower, and is limited to the vertices of the surface while for the continuous Dijkstra approach the query point can be *any* point on the surface. The sequence tree approach is easier to implement, and less vulnerable to inaccuracies in calculations involving floating point numbers because it does not require a priority queue, and does not require splitting an interval at a point equidistant from s through two face sequences. However, the sequence tree approach needs to compare two path

lengths to apply the one-angle one-split property, and it is practically impossible to represent these lengths accurately. Kaneva and O'Rourke [60] implemented the sequence tree approach successfully using some heuristics, but no error analysis is known to support this experimental result.

2.2.3 Steiner point approach

Papadimitriou [76] first introduced the idea of discretizing space by adding Steiner points and approximating a shortest path through the space by a shortest path in the graph of Steiner points. The problem he studied is to find a shortest obstacle-avoiding path in 3D. More precisely, given a polyhedral scene, i.e., a set of polyhedra given in terms of the coordinates of the vertices, and two points s and t, the goal is to find the shortest path from s to t that avoids all the polyhedra in the scene. Here the word "avoids" means that the path is disjoint from the interiors of all the polyhedra. For this problem computing an exact solution is NP-hard [24]. Note that although the algorithm of Papadimitriou does not deal with shortest paths on terrains, we are still discussing it because it was adopted for shortest paths on surfaces later on (See Sections 2.3.1 and 2.3.2).

To determine a $(1+\epsilon)$-approximate shortest path, Papadimitriou first discretizes the scene by subdividing each edge into a number of small parts. A graph G is then constructed as follows. Graph G contains a node for each part of an edge. Two nodes in G are connected by a link if and only if there exists an obstacle-avoiding straight-line path connecting the corresponding parts of the edges. The weight of a link is the distance between the midpoints of the corresponding parts of the edges. Note that because a node in G represents *a part of an edge* rather than a single point on that edge, the existence of a link does not necessarily mean that the midpoints of the corresponding parts of the edges are "visible" from each other. After a shortest s-t path P in the graph is constructed using Dijkstra's algorithm [44], the desired path is obtained by adjusting each segment of P individually to avoid obstacles if necessary, and then connecting each pair of consecutive segments that have been disconnected (by the adjustment) with a small segment along the corresponding edge.

The main trick here is to ensure that the parts of the edges are so small that the distance between any two points in two different parts can be considered constant, and the resulting relative error in the calculation is bounded by ϵ. This is achieved by placing a sequence $(\ldots, x^e_{-2}, x^e_{-1}, x^e_0, x^e_1, x^e_2, \ldots)$ of Steiner points on each edge e as follows (Figure 2.3). Let d_e be the distance of e from s. Point x^e_0 is the nearest point in e from s; x^e_1 is a points such that $|x^e_1 x^e_0| = \frac{\epsilon}{4n} d_e$; and for $i > 1$, x^e_i is a point on the same side of x^e_0 as x^e_1 such that $|x^e_i x^e_0| = (1 + \frac{\epsilon}{4n})|x^e_{i-1} x^e_0|$. The points $x^e_{-1}, x^e_{-2}, \ldots$ are placed in the same manner on the other side of x^e_0 on e. Such placement of Steiner points guarantees that there are at most $O(\frac{n^2}{\epsilon}(L + \log(n/\epsilon)))$ parts in each edge, and that the length of a part is at most $\frac{\epsilon}{4n}$ times the straight-line distance of the part from s, where L is the number of bits needed to represent the coordinates of the scene. Using these bounds, the running time of the algorithm can be shown to be $O(\frac{n^4}{\epsilon^2}(L + \log(n/\epsilon))^2)$.

Papadimitriou [76] also gave a faster $O(\frac{n^3}{\epsilon}(L + \log(n/\epsilon))^2)$-time algorithm, which is rather complicated. The idea is that in this algorithm the distance between any two points in two different parts of the edges is approximated by a linear function of the points' coordinates, unlike the previous algorithm which approximates the distance with a constant.

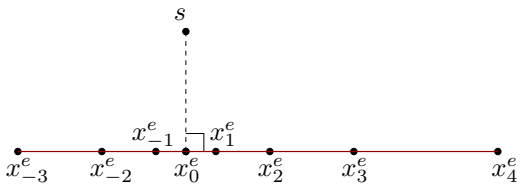


Figure 2.3: Placing Steiner points non-uniformly.

Choi, Sellen and Yap [32] later on showed that the result in Papadimitriou [76] is in the bit model, yet some critical part of his analysis are based on the real RAM model. Choi, Sellen and Yap filled in the details in the bit model.

2.3 Relevant shortest path problems in terrains

2.3.1 The weighted region problem

The weighted region problem [on terrains] is a generalization of the shortest path problem on polyhedral surfaces. In this problem, a set of constant weights is used to model the difference in costs of travel in different regions on the surface, and the goal is to minimize the weighted length of a path. The weighted length of a path is the sum of the weighted lengths of its segments, where the weighted length of a segment is defined as the Euclidean length of the segment times the weight of the face containing the segment. The complexity of the weighted region problem is still unknown. Mitchell and Papadimitriou [71] first studied the problem, and gave an approximation algorithm utilizing certain properties of a locally shortest path in weighted regions. Several faster approximation schemes [13, 14, 15, 63, 92] have been devised later on, all using the Steiner point approach. A comparison between these algorithms can be found in Aleksandrov et al. [15]. In this section we will focus on how the approaches mentioned in Section 2.2 have been employed in these algorithms.

Note that although Mitchell and Papadimitriou [71] defined the weighted region problem in 2D (i.e., for a planar subdivision), all the later papers focused on the problem in terrains. In this thesis we will use "weighted region problem" to denote this more general version of the problem.

Mitchell and Papadimitriou [71]

Mitchell and Papadimitriou [71] used the continuous Dijkstra approach to solve the weighted region problem in the plane. Although the algorithm works in the same manner as the continuous Dijkstra approach for shortest paths by Mitchell, Mount and Papadimitriou [70], the details are trickier. The main reason for this difference is that while a geodesic path in the discrete geodesic problem lies along a straight line, this is not the case in the weighted region problem. In this case, if a locally shortest path P crosses an edge e at point x to travel from a face f_1 of weight w_1 into a face f_2 of weight w_2, P bends at x, and the angles P makes with the perpendicular on e at x obeys Snell's law of refraction of light. More precisely, if ψ_1 and ψ_2 are the angles P makes with the perpendicular on e at x in f_1 and f_2 respectively (Figure 2.4(a)),

then $w_1 \sin\psi_1 = w_2 \sin\psi_2$. Besides the bends along P, there are several other issues that make the algorithm complicated:

(i) Assuming $w_1 > w_2$, angle ψ_2 can be as large as $\frac{\pi}{2}$, implying that a locally shortest path can contain a segment that is a part of an edge. This is impossible in the discrete geodesic problem.

(ii) For the same reason, if $w_1 < w_2$ and $\psi_1 = \frac{\pi}{2}$, path P can leave e at *any* position of x, implying that there is no unique way to trace P onward (Figure 2.4(b)).

(iii) A shortest path can cross a single region many times, as shown in Figure 2.4(c), where the shaded regions have weights significantly higher than the weight of the white region. This cannot happen in the discrete geodesic problem, although geodesic paths in both the problems may cross a single face multiple times (e.g., a path spiraling down from the apex of a pyramid to its base can be locally shortest in both the problems).

These issues make the number of events handled in this algorithm $O(n^4)$, which is significantly more than $O(n^2)$ events in the discrete geodesic problem.

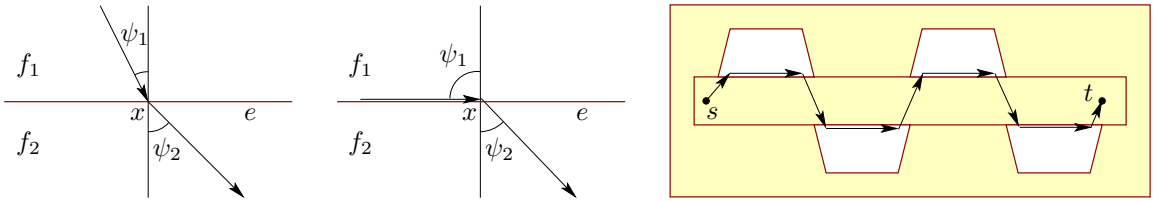


Figure 2.4: Complications with shortest paths in the weighted region problem.

There is another issue that makes this algorithm even slower. During the trimming of a candidate interval in the continuous Dijkstra approach, we need to solve the following sub-problem: given the sequence of edges crossed by a geodesic path, determine the length of the path. For the discrete geodesic problem, this sub-problem can be solved in constant time because the unfolded path lies along a straight line. For the weighted region problem, determining the path length requires solving a set of equations involving sines of angles, which is not easy—we will briefly discuss the issue in Section 3.6. Mitchell and Papadimitriou used binary search to solve this sub-problem approximately, which results in a $O\left(n^8 \log\left(\frac{n}{\epsilon}\right)\right)$ time approximation algorithm for the weighted region problem.

Lanthier, Maheshwari and Sack [63]

Lanthier, Maheshwari and Sack [63] gave three approximation schemes for the weighted region problem. The schemes are simple and practical, but the returned paths have additive errors.

Their first scheme (called the "fixed scheme") places $m = n^2$ Steiner points evenly on every edge of the terrain, and then constructs a weighted graph G as follows. For each face f_i, a graph G_i is formed, which has all the vertices and Steiner points in f_i as nodes, and has a link between nodes x and y if and only if either x and y lie on different edges of f_i, or x and y are

adjacent on a common edge of f_i. The weight of each link of G_i is the weighted length of the corresponding segment in f_i. Graph G is the union of G_i over all faces f_i of the terrain. An approximate path is formed by running Dijkstra's algorithm on G. The length of the returned path is at most WL plus the length of the shortest path, where W is the largest face weight, and L is the length of the longest edge. The space and time requirements are $O(n^3)$ and $O(n^5)$ respectively.

The second scheme of Lanthier et al. (called the "interval scheme") places just enough Steiner points on every edge to make the distance between adjacent Steiner points at most $\frac{L}{n^2}$. Since the long edges of the terrain contain the same number of Steiner points as in their first scheme, the worst case bounds for approximation error, space requirement and time requirement remain the same. However, the second scheme performs better when many edges in the terrain have lengths much less than L.

The third scheme (called the "spanner scheme") eliminates certain links in G to form a smaller graph G' which approximates any path in G within a constant factor $\beta > 1$. In graph terminology, G' is called a β-spanner of G. Lanthier et al. construct G' by following the approach of Clarkson [33]: for every node x, the plane is first decomposed into a constant number of cones with apex at x, and then within each cone all the links emanating from x are removed except the one having the smallest weighted length. If the number of cones at a node is $k > 4$, then $\beta = \frac{1}{\cos(\pi/k) - \sin(\pi/k)}$. Graph G' thus formed has a constant number of links at every node, which improves the running time to $O(n^3 \log n)$, although the approximation error becomes much greater: the length of the returned path is now β times the length of the approximate path returned by the other two schemes.

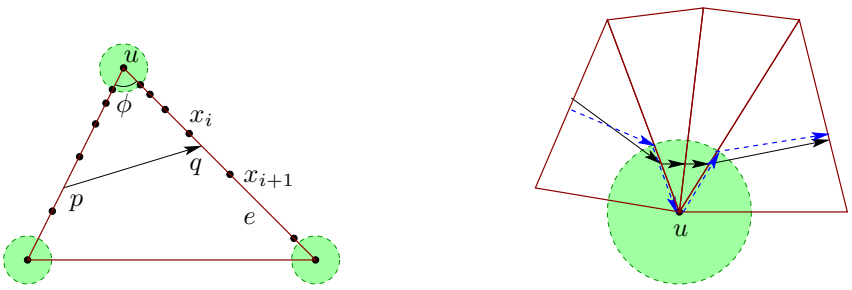
Lanthier et al. also gave experimental results that suggest that placing a small (hence constant) number of Steiner points per edge results in a good approximation for a "typical" terrain.

Aleksandrov et al. [13]

The main difference of the algorithm of Aleksandrov et al. [13] from that of Lanthier, Maheshwari and Sack [63] is that Aleksandrov et al. do not place Steiner points uniformly along the edges. Like Papadimitriou [76], they place the Steiner points in such a way that their distances from a particular end of the edge form a geometric progression. This approach significantly reduces the size of the graph, and has been used in several other terrain shortest path algorithms, including two algorithms in this thesis. We will now briefly explain this approach.

Let ϕ be the face angle of a triangular face at vertex u, and e be an edge adjacent to u in that face (Figure 2.5(a)). If x_i and x_{i+1} are two consecutive Steiner points on e such that $\frac{|ux_{i+1}|}{|ux_i|} = 1 + \epsilon \sin\phi$ where ϵ is a small constant, then it can be shown that for *any* segment pq that reaches a point $q \in x_i x_{i+1}$ from the other edge at u in the same face, $|x_i x_{i+1}| \leq \epsilon |pq|$. It then follows easily that the length of the link connecting any of the two Steiner points near p with any of the two Steiner point near q is bounded by $(1 + 2\epsilon)|pq|$. Thus every segment of a shortest path is approximated by a nearby segment connecting two Steiner points.

The main problem with such placement of Steiner points is that the points become infinitesimally close to one another near the vicinity of u, resulting in a very large number of Steiner



(a) (b)

Figure 2.5: (a) Non-uniformly placed Steiner points, and (b) an approximate path near a vertex u.

points on e. Aleksandrov et al. solved this problem by placing no Steiner points near u. More precisely, they place no Steiner points in the ball of radius ϵh centered at u, where h is the smallest 2D height of a triangle. As a result, the number of Steiner points on an edge becomes $O(\log_{1+\epsilon \sin\phi}(L/(\epsilon h)))$, where L is the length of the longest edge of the terrain. Note that every edge has two such sets of Steiner points, one set for each of the endpoints. Although there are no Steiner points near u, the small radius of the ball centered at u allows approximating a sequence of segments close to u with a sequence of two segments through u, as shown in Figure 2.5(b). In this figure, the dotted arrows form an approximation of the path drawn with solid arrows. Note that the "short detour through u" shown in the figure can be arbitrarily longer than the corresponding part of the shortest path. However, the part of the shortest path that lies before the ball at u and outside any other balls is longer than $(1-2\epsilon)h$, which bounds the approximation ratio of the whole path. In the weighted case, the approximation ratio becomes $1+2\epsilon+\frac{2W\epsilon}{(1-2\epsilon)w}$, where W and w are respectively the largest and the smallest weight. We are unable to verify their error analysis because their approach of approximating a segment very close to a vertex (in Claim 3.22) seems to guarantee a 2-approximation rather than a $(1+\epsilon)$-approximation.

The above scheme takes $O(nm \log nm + nm^2)$ time to compute a $(1+\epsilon)$-approximate shortest path, where $m = \log_{1+\epsilon \sin\phi}(L/(\epsilon h))$. Aleksandrov et al. claim that they can compute a $(1+\epsilon)$-approximate shortest path in $O(nm \log nm)$ time by first reducing the size of the graph used in the first scheme by eliminating certain edges. More precisely, if p is a Steiner point on edge e_1, and q_1, q_2 and q_3 are Steiner points on a different edge e_2 of a common face such that $|pq_1| \le |pq_2| \le |pq_3|$, then the new graph does *not* contain the edge pq_2 if $|q_1q_3| \le \epsilon|pq_1|$. This idea seems similar to the one used by Papadimitriou [76], but Aleksandrov et al. do not provide details on how they reduce the size of the graph in $O(nm \log nm)$ time. Ignoring geometric parameters, the running time of their algorithm can be simplified to $O(\frac{n}{\epsilon^2} \log n \log \frac{1}{\epsilon})$ [15].

Aleksandrov, Maheshwari and Sack [14]

Aleksandrov, Maheshwari and Sack [14] try to improve the above algorithm by using a modified version of Dijkstra's algorithm. Unlike the paper by Aleksandrov et al. [13] in which the shortest path tree in the graph is constructed blindly, i.e., without considering any geometry, this approach considers Snell's law at every step of expanding the shortest path tree. When

generating the children of a Steiner point p in the tree, only the Steiner points (across the next face) that lie in a small cone defined by Snell's law are considered. More precisely, let p_0p be the last segment of the path to p, and pq_0 be a segment in the next face for which the path obeys Snell's law at p (Figure 2.6). A Steiner point q is considered as a possible child of p in the shortest path tree if $\angle qpq_0 \leq \delta$, where

$$\delta \;=\; \left(1 + \sqrt{\frac{\pi w_1}{2w_2}}\right) \pi\sqrt{\epsilon} \;,$$

and w_1 and w_2 are the weights of the faces containing p_0p and pq_0 respectively. The running time of the algorithm is $O(\frac{n}{\epsilon}\log\frac{1}{\epsilon}(\frac{1}{\sqrt{\epsilon}} + \log n))$ ignoring geometric parameters. This approach appears to suffer from the same issue as the previous one regarding segments very close to vertices.

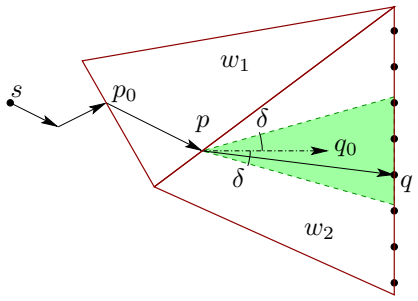


Figure 2.6: Utilizing Snell's law with approximate paths.

The Bushwhack algorithm of Sun and Reif [92]

Sun and Reif [92] use a discrete search algorithm, called Bushwhack [90], which modifies Dijkstra's algorithm for graphs representing approximate shortest paths on terrains. The Bushwhack algorithm improves the running time of Dijkstra's algorithm from $O(|V|\log|V| + |E|)$ to $O(|V|\log|V|)$ by utilizing certain geometric properties of the shortest paths in such a graph. Sun and Reif fill in the missing details in Aleksandrov, Maheshwari and Sack [14], and then apply the Bushwhack algorithm to devise a faster approximation algorithm for the weighted region problem, with running time $O(\frac{n}{\epsilon}\log\frac{n}{\epsilon}\log\frac{1}{\epsilon})$.

The Bushwhack algorithm has been used in other terrain shortest path algorithms, including our algorithms in Chapters 4 and 5. We will therefore explain this algorithm briefly.

The Buskwhack algorithm relies on a simple, yet important, property of shortest paths on terrains: two shortest paths through different face sequences do not intersect each other at an interior point of a face. As a result, for any two consecutive Steiner points u_1 and u_2 on edge e for which the distances from s are already known, the corresponding sets of "possible next nodes on the path" are disjoint, as shown using shading in Figure 2.7(a). This property makes it possible to consider only a subset of links at a Steiner point v when expanding the shortest path tree onwards from v using Dijkstra's algorithm. More precisely, Sun and Reif maintain a dynamic list of intervals $I_{e,e'}$ for every pair of edges e and e' of a common face. Each point in an interval is reachable from s using a shortest path through a common sequence of intermediate

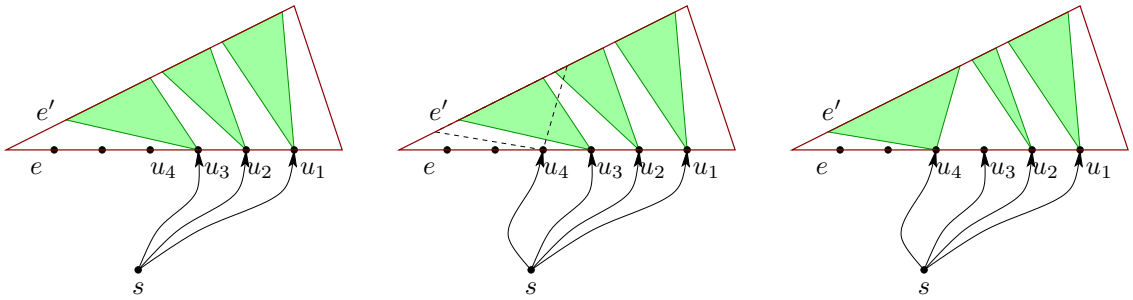


Figure 2.7: Maintaining the list $I_{e,e'}$ in the Bushwhack algorithm.

points. For every Steiner point v in e with known distance from s, $I_{e,e'}$ contains an interval of Steiner points on e' that are likely to become the next node in the path from s through v. The intervals in $I_{e,e'}$ are ordered in accordance with the ordering of the Steiner points v on e, which enables easy insertion of the interval for a Steiner point on e whose distance from s is yet unknown. For example, right after the distance of u_4 from s becomes known (i.e., right after u_4 gets dequeued in Dijkstra's algorithm) as shown in Figure 2.7(b), the Steiner points on e' that are closer to u_4 than to any other Steiner points on e with known distances from s can be located in time logarithmic in the number of Steiner points on e', using binary searches (Figure 2.7(c)). Within the interval for each Steiner point $u \in e$, only the Steiner point that is the nearest one from u is enqueued. Since the nearest Steiner point from u in its interval can be determined in constant time, each iteration of the modified Dijkstra's algorithm (i.e., the Bushwhack algorithm) takes $O(|V|)$ time, resulting in a total running time of $O(|V| \log |V|)$. (Note that "traditional" Dijkstra's algorithm completes each loop in $O(d(v) + \log |V|)$ time, where $d(v)$ is the degree of a vertex v.)

The intervals used by Sun and Reif are analogous to the intervals used in the continuous Dijkstra approach (discussed in Section 2.2). In both cases, an interval consists of points to which the shortest paths have the same combinatorial structure. The main difference is that the intervals used by Sun and Reif consist of discrete points, each of which is reachable from s using a shortest path through a common sequence of intermediate points while the intervals used in the continuous Dijkstra approach consist of a continuous range of points, each of which is reachable from s using a shortest path crossing a common sequence of edges.

Aleksandrov, Maheshwari and Sack [15]

Unlike all the previous Steiner point approaches for the weighted region problem, all of which place the Steiner points along the edges of the terrain, Aleksandrov, Maheshwari and Sack [15] place them along the bisectors of the face angles. Each link in graph G connects a pair of Steiner points p and q that lie on the bisectors of two face angles sharing a common edge e. The weight of the link connecting p and q is the weighted length of the shortest path from p to q that intersects e. The shortest path can have one of the following two forms. If p and q lie on two different faces with weights w_1 and w_2 respectively as shown in Figure 2.8(a), the path intersects e at a point x for which $w_1 \sin \psi_1 = w_2 \sin \psi_2$. On the other hand, if both p and q lie a common face of weight w_1, and the other face adjacent to e has weight $w_2 < w_1$ as shown in

Figure 2.8(b), the path intersects e at a segment xy for which $\sin\psi_1 = \sin\psi_2 = \frac{w_2}{w_1}$. Each of these paths can be determined in constant time. As in the paper by Aleksandrov, Maheshwari and Sack [14], the Steiner points are placed in a geometric progression, and no Steiner point is placed close to a vertex.

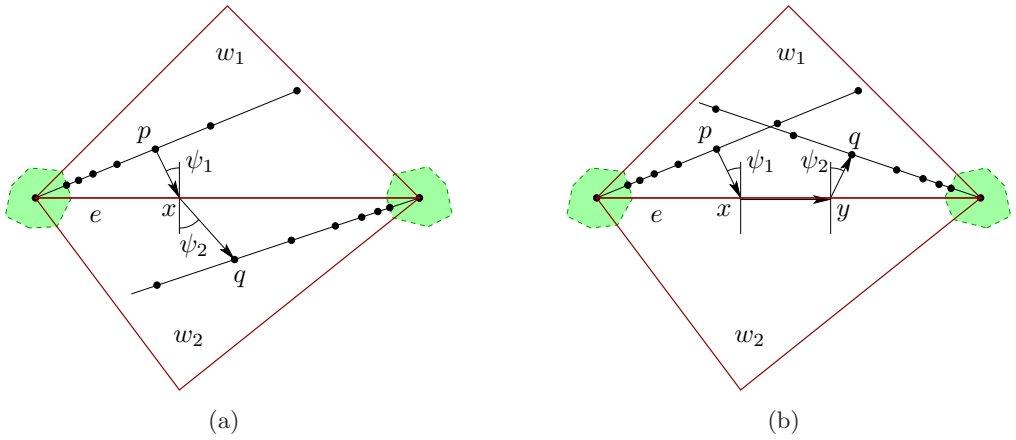


Figure 2.8: Links between the Steiner points along two bisectors.

As in the previous approaches, a shortest path in graph G is a $(1+\epsilon)$-approximation of a shortest path in the terrain. Applying the Bushwhack algorithm, a shortest path in G is determined in $O(\frac{n}{\sqrt{\epsilon}} \log \frac{n}{\epsilon} \log \frac{1}{\epsilon})$ time.

2.3.2 The shortest anisotropic path problem

The *shortest anisotropic path* problem is a generalization of the weighted region problem. In this problem, the goal is to minimize the weighted length of a path on a triangulated terrain, where the weight of a path segment ab depends both on the face containing ab and the direction of ab. In other words, the weight in a region is expressed as a function of the direction of travel.

The shortest anisotropic path problem has many practical applications, such as finding an energy-minimizing path for a robot [91], and minimizing travel time in presence of flows [80].

Neither a polynomial time algorithm nor a hardness proof is known for the shortest anisotropic path problem. In Chapters 3 to 5 of this thesis we will examine two special cases of this problem. Previous results on shortest anisotropic paths fail to solve our special cases because those results are based on two restricted anisotropic weight models, and neither of these models apply to our problems. Now we will discuss these two weight models, and mention how the Steiner point approach mentioned in Section 2.2.3 has been used for these models.

The anisotropic weight model of Rowe and Ross [81]

Rowe and Ross [81] introduced the shortest anisotropic path problem, and studied a model of weights that captures the effect of gravity and friction on a vehicle moving on a polyhedral terrain. We will briefly discuss various components of this model.

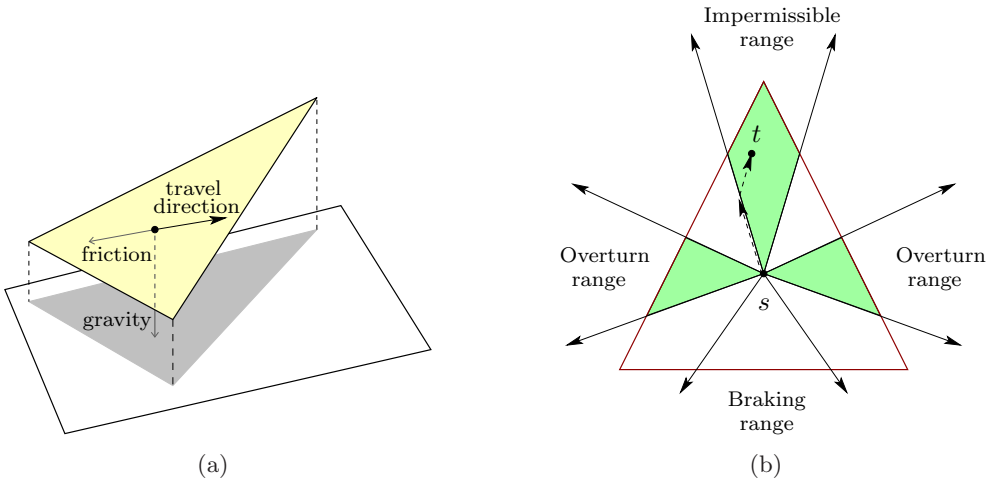


Figure 2.9: The model of Rowe and Ross for shortest anisotropic paths.

The effect of gravity on a vehicle moving on a non-level face depends on the direction of travel. For example, if the vehicle is moving upward [downward] in a face, gravity opposes [respectively supports] its motion (Figure 2.9(a)). The force of kinetic friction, on the other hand, always opposes the motion. As a result, depending on the slope of the face, a range of upward directions may be too steep for the vehicle, while a range of downward directions may necessitate braking to limit the speed of the vehicle. Similarly, a range of sideways direction may cause the vehicle to overturn. Based on such physical effects, the model of Rowe and Ross divides all the directions of travel into eight regions as shown in Figure 2.9(b). The shaded regions in the figure mark the forbidden directions. Each direction along a boundary of a region is called a *critical direction* of that region. Note that in this model the direction-dependent weight function in a face f is a constant function iff f is a level face.

The model assumes that any point in a forbidden direction can be reached by following a zigzag path switching between two critical directions, as shown by the dashed path from s to t in Figure 2.9(b). This is possible only when the angle between the two critical directions for any forbidden range is below π radians.

There are three papers [64, 89, 91] that address the shortest anisotropic path problem in this model. These papers present algorithms based on the Steiner point approach. There is one crucial property that shortest paths in the weighted region problem have, but shortest anisotropic paths do not have. A shortest path in the weighted region problem can come close to a given vertex only once—this follows from the characterization of bend angles in Mitchell and Papadimitriou [71]. However, a shortest anisotropic path may come close to a given vertex many times. We will give in Figure 4.2 an example of an SDP that shows this behavior. A similar shortest anisotropic path in the model of Rowe and Ross can be constructed when, for example, the angle defining the "impermissible range" in Figure 2.9(b) is very close to π, and the angles defining the "overturn ranges" are very close to zero. The analysis of approximation algorithms for the weighted region problem depends crucially on this property because the error close to a vertex is amortized against the "long" subpath leading to this vertex. This analysis fails for shortest anisotropic paths because repeated visits close to the same vertex need not be separated by long paths. We do not see how this is remedied in the above three papers on

shortest anisotropic paths. For our algorithms in Chapters 4 and 5, we analyze errors using a different argument.

Sun and Reif [91, Section V] relaxed the assumption in the model of Ross and Rowe (that any point in a forbidden direction can be reached by following a zigzag path), but they did so only in isolated faces. If we instead relax the assumption in any two adjacent faces f_1 and f_2, their approach can make the number of Steiner points in those faces exponentially large. This is because they place Steiner points in two phases, and every Steiner point placed in the edge $f_1 \cap f_2$ in the second phase for face f_1 [f_2] will generate more second phase Steiner points in f_2 [respectively f_1] (see Figure 11 in Sun and Reif [91]).

The anisotropic weight model of Cheng et al. [31]

Cheng et al. [31] introduced another anisotropic weight model which assumes that the (anisotropic) weight associated with a direction of travel is bounded by constants from both above and below. More precisely, they assume that there exists a constant $\rho \geq 1$ such that the weight in any direction lies in the interval $[1, \rho]$. They also assume a "convexity" property of the reciprocal of the direction-dependent weight function. The latter assumption makes a zigzag path in a face non-optimal. This is a more restricted model than the one by Rowe and Ross because, for example, forbidden directions are not allowed in this model. Another limitation of Cheng et al. is that their algorithm works only in 2D, i.e., on a subdivision of the plane, and cannot be used for a general terrain.

The algorithm of Cheng et al. is based on the Steiner point approach. The positions of the Steiner points are determined in a completely new way: the Steiner points are more dense in the part of the terrain close to *both* source s and destination t, and they become sparser in the part far away. More precisely, Cheng et al. first partition the terrain by "cutting" it along a set of ellipses each of which has s and t at its focal points. The bounds on direction-dependent weights play an important role in determining the shapes of the ellipses. Then for each partition, Cheng et al. place Steiner points uniformly along all the edges lying in that partition. The distance between consecutive Steiner points in a partition depends on how far the partition is from s and t—a nearer partition gets more Steiner points than the one further away.

Another paper by the same authors [30] presents a data structure that allows preprocessing the terrain in such a way that a $(1+\epsilon)$-approximate shortest anisotropic path can be computed in time linear in the size of the path.

Chapter 3

Shortest Descending Paths: Towards an Exact Algorithm

3.1 Introduction

Polynomial time algorithms are known for many geometric shortest path problems. In the plane, a shortest path among polygonal obstacles can be found using the continuous Dijkstra approach [70] where a wavefront expands over the surface from the source, deflecting around obstacles. The analogous problem in 3D is NP-hard [24], but the special case where the path must lie on a polyhedral surface is amenable to the same continuous Dijkstra approach, or to the sequence tree approach of Chen and Han [28], which grows shortest paths by unfolding the surface face by face, still in breadth-first order but not necessarily in path-length order. The excellent survey by Mitchell [69] discusses these and other shortest path problems. One variant of the continuous Dijkstra approach that has received recent attention utilizes a quad-tree like subdivision of the plane to achieve faster running time [55]. Schreiber and Sharir [87] have extended this variant for a 3D convex surface, which has later been applied to several realistic scenarios [86].

This chapter is about a variant of the shortest path problem for which no exact algorithm is known, the *shortest descending path (SDP) problem*: given a polyhedral terrain, and points s and t on the surface, find a shortest path on the surface from s to t such that, as a point travels along the path, its elevation, or z-coordinate, never increases. We need to compute a shortest descending path, for example, for laying a canal of minimum length from the source of water at the top of a mountain to fields for irrigation purpose, and for skiing down a mountain along a shortest route [9, 84].

The SDP problem was introduced by de Berg and van Kreveld [37], who gave an algorithm to preprocess a terrain in $O(n \log n)$ time and in $O(n)$ space so that it can be decided in $O(\log n)$ time if there exists a descending path between any pair of vertices. They did not consider the length of the path, and left open the problem of finding the shortest such path. In a subsequent paper, Roy, Das and Nandy [83, 84] studied the problem in two restricted settings. For convex (or concave) terrains, they use the *continuous Dijkstra approach* [70] to preprocess the terrain in $O(n^2 \log n)$ time and $O(n^2)$ space so that an SDP of size k can be determined

in $O(k + \log n)$ time. For a terrain consisting of edges parallel to one another, they find an SDP in $O(n \log n)$ time by transforming selected faces of the terrain in a way that makes the unfolded SDP a straight line segment. Roy [82] later improved this running time to $O(n)$, by replacing a sorting step in the previous algorithm with a divide-and-conquer technique. The SDP problem on general terrains has been addressed by Ahmed et al. [4]. The paper gives approximation schemes using the Steiner point approach—we will discuss them in Chapter 4. All these papers on the SDP problem provided very little insight on the "structure" of SDPs in a general terrain, and it remained a mystery why the problem is so easy for two special classes of terrain yet unsolved for the general case. In this chapter we explore the deeper structure of SDPs.

A main property of (unconstrained) shortest paths on terrains is that they unfold to straight lines. This makes two sub-problems trivial: (1) to extend a shortest path from one face into an adjacent face; and (2) to find a shortest path to a specified target point given the sequence of faces that the path traverses. Our first main result is an efficient algorithm to solve sub-problem (1) for SDPs. This is non-trivial because SDPs do not unfold to straight lines. We give a full characterization of the bend angles of an SDP, showing that the bend angles follow a generalized form of Snell's law of refraction of light. Our second main result is an adaptation of the sequence tree approach of Chen and Han (discussed in Section 2.2.2) to SDPs. This requires efficient solutions to sub-problems (1) and (2) plus a structural result that permits the sequence tree to be pruned. We prove the structural result for SDPs, so the only missing ingredient is an efficient solution to sub-problem (2). In other words, we reduce the SDP problem in polynomial time to sub-problem (2). Our results have two implications. First, we isolate the difficult aspect of SDPs. The difficulty is not in deciding which face sequence to use, but in finding the SDP through a given face sequence. This involves numerical issues similar to the ones faced by Mitchell and Papadimitriou while computing shortest paths in the weighted region problem [71, Section 8]. Secondly, our results help us identify some classes of terrains for which the SDP problem is solvable in polynomial time. We give algorithms for two such classes.[1]

The chapter is organized as follows. We define relevant terms in Section 3.2. We establish in Section 3.3 some properties of SDPs that lead to a complete characterization of the bend angles of an SDP. We also present in this section two approximation algorithms to find SDPs through given faces, one based on convex programming, and the other based on our full characterization of the bend angles and binary search. In Section 3.4 we adapt the sequence tree approach of Chen and Han for the SDP problem, and reduce the SDP problem to the case of a known face sequence. We give polynomial time algorithms for two classes of terrains in Section 3.5. Finally we conclude in Section 3.6 with a discussion on numerical issues with SDPs in general terrains.

3.2 Terminology

For any point p in the terrain, we use $h(p)$ to denote the height of p, i.e., the z-coordinate of p. A line or face in 3D is called *level* if all points on that line or face have the same height.

[1] A part of Section 3.3 appeared in Ahmed and Lubiw [9], and in a preliminary form earlier in Ahmed and Lubiw [5]. A preliminary version of the rest of this chapter was presented in FWCG 2007 [7].

A path P from s to t on the terrain is *descending* if the z-coordinate of a point p never increases while we move p along the path from s to t. A line segment of a descending path in face f is called a *free segment* if moving either of its endpoints by an arbitrarily small amount to a new position in f keeps the segment descending. Otherwise, the segment is called a *constrained segment*. All the points in a constrained segment are at the same height, though not all constant height segments are constrained. For example, a segment in a level face is free, although all its points are at the same height. Clearly, a constrained segment can only appear in a non-level face, and it is parallel to a level line in that face. A path consisting solely of free [constrained] segments is called a *free path* [respectively *constrained path*]. For any point p, we call the path (p, p) a *trivial constrained path*.

We use the following convention throughout this chapter. The term "edge" denotes an edge of a (triangular) terrain face, and the term "segment" denotes a line segment of a path. Similarly, an endpoint of an edge is called a "vertex", while an endpoint of a segment is called a "node". In Section 3.4 we use "node" and "link" to denote the corresponding entities in a tree. We assume that all paths in our discussion are directed unless stated otherwise.

We now define a *locally shortest descending path (LSDP)*, which is analogous to a *geodesic path* (i.e., a *locally shortest path*) [70]. An LSDP between two nodes is a descending path that cannot be shortened by slight perturbation of the intermediate nodes. Note that perturbing a single node in a descending path may make the path infeasible (i.e., not descending), and hence, we allow more than one node to be perturbed simultaneously. For example, if we change the height of a node p from h_1 to $h_2 > h_1$, all the points before p on the path must be moved to height at least h_2 to keep the path descending. Also note that a constrained path is an LSDP.

We prove in Section 3.3.2 the uniqueness of an LSDP through a given sequence of faces—a property that is used throughout the chapter. The proof relies on a certain function being strictly convex, which is defined as follows [22, Section 3.1.1]:

Definition 3.1 (Convexity and Strict convexity of a function)**.** *A function $f : \mathbb{R}^n \to \mathbb{R}$ is called* convex *if the domain of f is a convex set, and for all x, y in that domain and all $c \in [0, 1]$,*

$$f(cx + (1 - c)y) \leq cf(x) + (1 - c)f(y) \ . \tag{3.1}$$

The function f is called strictly convex *if strict inequality holds in Equation 3.1 whenever $x \neq y$ and $0 < c < 1$.*

The conventions used in the figures to mark various components related to a descending path are shown in Figure 3.1. In particular, an arrow with a solid, dark arrowhead denotes a path segment, and the arrow may be heavy to mark a level segment. In the figures where the direction of the edges are important, we again use arrows to mark the upward direction, but we make the arrowheads V-shaped ("open") in this case to differentiate the edges from the segments. Dotted lines are used to show level lines in a non-level face. The figures with face sequences show the faces after unfolding them onto a common plane.

3.3 Characteristics of a shortest descending path

In this section we discuss various characteristics of shortest descending paths (SDPs) and locally shortest descending paths (LSDPs). Our goal is to characterize the bend angles of an LSDP

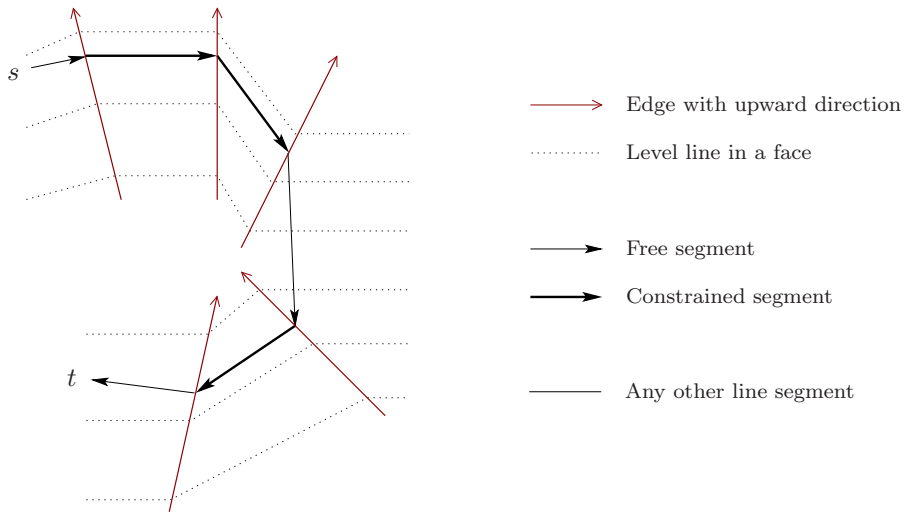


Figure 3.1: General legend for all figures in this section.

that does not go through any vertex of the terrain. This characterization gives us the ability to trace an LSDP onward through the terrain from a given ray leaving s. Unlike the case of a geodesic path [70], tracing an LSDP in this manner is a non-trivial task because, as we will soon see, an unfolded LSDP is not necessarily a straight line segment.

Throughout this section, we will use $\sigma = (f_0, f_1, \ldots, f_k)$ to denote a sequence of faces used by an LSDP from $s \in f_0 - f_1$ to $t \in f_k - f_{k-1}$. For each $i \in [1, k]$, f_{i-1} and f_i share an edge; let the edge $f_{i-1} \cap f_i$ be $a_i b_i$ with $h(a_i) \geq h(b_i)$. (Note that σ can also be defined as the edge sequence $(a_1 b_1, a_2 b_2, \ldots, a_k b_k)$.) Our discussion below considers this problem as a 2D problem, because by unfolding each pair of consecutive faces in the sequence appropriately, we can "align" all the faces in the sequence onto a plane, as is usually done for paths on polyhedral surfaces. There is an obvious one-to-one correspondence between points in this unfolded face sequence and on the original terrain. In order to avoid an extra layer of notation, we will, for point p in the unfolded face sequence, still use $h(p)$ to refer to the height of the corresponding point on the original terrain. Similarly, when we say that a line in the unfolded face sequence is level, we mean it has constant height on the original terrain.

3.3.1 Similarities with a geodesic path

An LSDP and a geodesic path over a terrain are similar in many respects. The following lemmas establish two properties of an LSDP that make an LSDP analogous to a geodesic path:

Lemma 3.1. *Any subpath of an LSDP is an LSDP.*

Proof. If a subpath P_1 of an LSDP P is not locally shortest, there exists an LSDP P_1' corresponding to P_1 such that the length of P_1' is less than that of P_1. In that case, we can modify P by replacing the subpath P_1 with P_1' and get an LSDP of length less than that of P. This leads to a contradiction. □

Lemma 3.2. *An LSDP consists of straight line segments, and bends only at the edges of the terrain.*

Proof. Suppose that an LSDP P bends at point p that is an interior point of some face f of the terrain. Let P_1 be the connected subpath of $P \cap f$ that contains p, and p_1 and p_2 be respectively the starting and ending points of P_1. Since P is a descending path, $h(p_1) \geq h(p_2)$. Therefore, the line segment from p_1 to p_2 is a descending path, and its length is less than that of P_1 because P_1 bends at p. That means, P_1 is not an LSDP. Then, by Lemma 3.1, P is not an LSDP, which is a contradiction. □

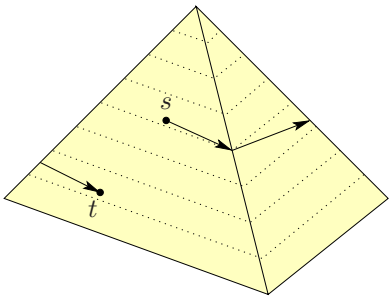


Figure 3.2: An LSDP visiting a face twice.

As in the case of a geodesic path [70], an LSDP may visit a single face more than once. For example, a string tightly wrapped around a pyramid as shown in Figure 3.2 is an LSDP from s to t, and it visits a face twice. However, like a shortest path, an SDP visits a face at most once. The following lemma proves this claim (Roy, Das and Nandy [84, Lemma 2] also prove the claim).

Lemma 3.3. *The intersection of an SDP P with a face of the terrain is either empty or a line segment.*

Proof. If P doesn't visit face f, then $P \cap f$ is empty. Otherwise, let p_1 and p_2 be the first and the last points of P in $P \cap f$. Then, $h(p_1) \geq h(p_2)$, and hence, the line segment from p_1 to p_2 is a descending path. If the part of P from p_1 to p_2 is not a line segment, replacing that part of P with the line segment p_1p_2 makes a descending path of shorter length, which is a contradiction. □

One important difference between an LSDP and a geodesic path is that unlike a geodesic path [70], two consecutive segments of an LSDP through an edge ab do not always become a straight line segment when the two faces of the terrain adjacent to ab are unfolded onto a plane. Before proving this claim, we define two angles at every edge intersected by an LSDP to quantify the angles at the bends along an LSDP.

Definition 3.2 (Entering and Exiting angles)**.** *Let $P = (p, q, r)$ be a path from an interior point p in face f_1 to an interior point r in face f_2 adjacent to f_1 such that P crosses edge $ab = f_1 \cap f_2$ at q where $h(a) \geq h(b)$ (Figure 3.3). Let $p' \in f_1$ and $q' \in f_2$ be two points such that both $p'q$ and qr' are perpendicular to ab at q. (Thus $p'qr'$ becomes a straight line segment*

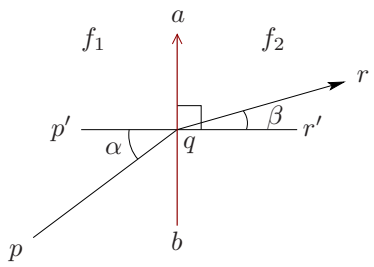


Figure 3.3: Entering and exiting angles.

after f_1 and f_2 are unfolded to lie in a plane.) The angle $\angle pqp'$ is called the entering angle of P at ab, *and is considered positive if and only if p and b are on the same side of line $p'q$. The angle $\angle rqr'$ is called the* exiting angle of P at ab, *and is considered positive if and only if r and a are on the same side of line qr'.*

In Figure 3.3, α and β are respectively the entering angle and the exiting angle of P at ab. In case ab is level, i.e., $h(a) = h(b)$, exchanging the labels a and b reverses the signs of both angles; our discussion is valid for any of these labeling choices.

Before discussing how an LSDP bends at its intermediate nodes, we observe the fact that any path consisting of two non-collinear line segments can be made shorter by perturbing the intermediate node of the path. More precisely:

Observation 3.1. *Let pq and qr be two non-collinear line segments, and q' be a point inside the smaller angle made by the two segments at q. If q' is inside $\triangle pqr$, then $|pq'|+|q'r| < |pq|+|qr|$.*

Proof. Let q'' be the intersection point of the lines pq' and qr (Figure 3.4). Since q' lies inside $\triangle pqr$, $|pq'| + |q'r| < |pq'| + |q'q''| + |q''r| = |pq''| + |q''r| < |pq| + |qq''| + |q''r| = |pq| + |qr|$.

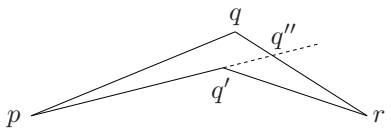


Figure 3.4: Proving that $|pq'| + |q'r| < |pq| + |qr|$ for any q' is inside $\triangle pqr$.

□

Using the above observation, Lemma 3.4 below proves the claim that an unfolded LSDP is not always a straight line segment:

Lemma 3.4. *The descending path $P = (p, q, r)$ is an LSDP if and only if one of the following holds:*

(i) $\alpha = \beta$;

(ii) $\alpha > \beta$, *and qr is constrained; or*

(iii) $\alpha < \beta$, and pq is constrained.

Proof. $\Leftarrow$:

(i) If $\alpha = \beta$, then for any point $q' \in ab$ such that $q' \neq q$, the length of the path (p, q', r) is more than that of P. Therefore, P is an LSDP.

(ii) If $\alpha > \beta$, and qr is constrained, then for any point $q' \in ab$ such that $h(q') > h(q)$, q lies inside $\triangle pq'r$ (Figure 3.5(a)). By Observation 3.1, the length of the path (p, q', r) is more than that of P. On the other hand, for any point $q' \in ab$ such that $h(q') < h(q)$, (p, q', r) is not a descending path because $h(r) = h(q) > h(q')$. Therefore, P is an LSDP.

(iii) If $\alpha < \beta$, and pq is constrained, the proof is similar to the one for Case (ii), except that the path (p, q', r) is longer than P when $h(q') < h(q)$, and is infeasible when $h(q') > h(q)$.

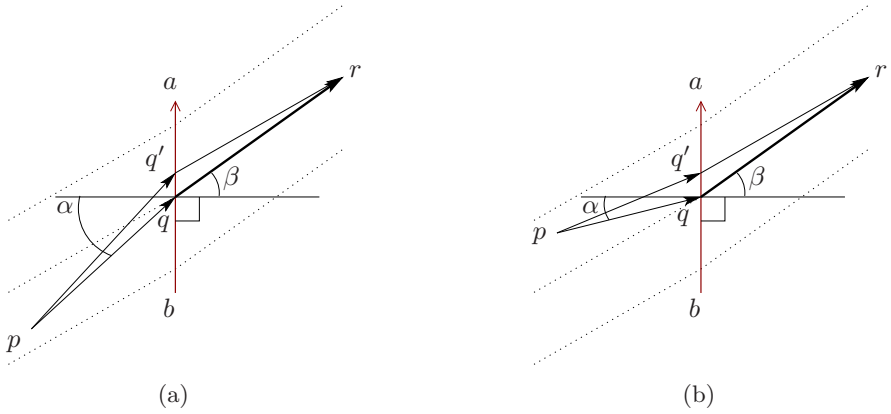


Figure 3.5: Showing that an unfolded LSDP is not a straight line segment.

$\Rightarrow$: It is sufficient to show that if either $\alpha < \beta$ and pq is free, or $\alpha > \beta$ and qr is free, P is not an LSDP.

If $\alpha < \beta$ and pq is free, let q' be a point on ab slightly above q, but with $h(q') \leq h(p)$ (Figure 3.5(b)). Such a point q' always exists because pq is a free segment and we can make q' arbitrarily close to q. We form a new path $P' = (p, q', r)$. When q' is very close to q, q' lies inside $\triangle pqr$, and the length of P' is smaller than that of P by Observation 3.1. Because qr is descending, the segment $q'r$ is also descending. The segment pq' is descending since $h(q') \leq h(p)$. Therefore, P' is a descending path, and is shorter than P. So, P is not an LSDP. We can similarly show that P is not an LSDP if $\alpha > \beta$ and qr is free. □

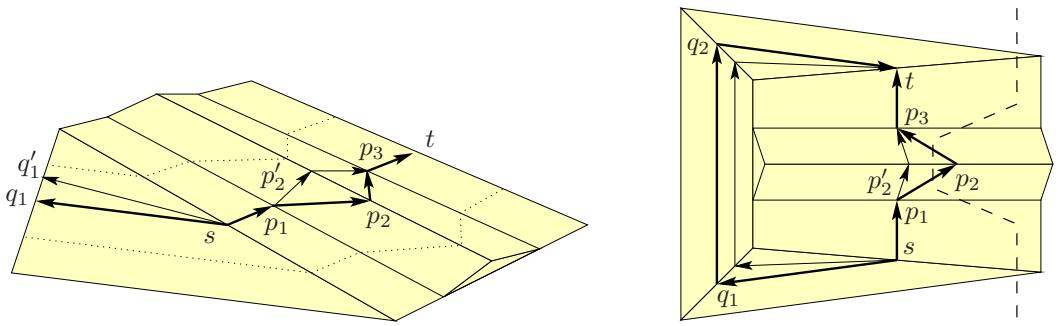
We can intuitively say that P can deflect downward if qr is constrained, and can deflect upward if pq is constrained. When ab is a level edge, this intuition may seem meaningless because "upward" and "downward" do not make any sense in this case. However, since a constrained segment does not intersect a level edge, P is a free path in this case and therefore must go straight (i.e., $\alpha = \beta$). The intuition thus holds in all cases.

Roy, Das and Nandy [83, Lemma 1] proved that an SDP does not bend downwards in a convex terrain. This property follows immediately from Lemma 3.4 because in such a terrain *the* angle formed by the level lines of f_1 and f_2 at q that contains a is at most π.

In spite of all the similarities between an LSDP and a geodesic path, an SDP and a shortest path can be very different from each other. The following lemma proves this claim.

Lemma 3.5. *Let P_T and P'_T denote respectively an SDP and a shortest path from s to t in terrain T. There exists a terrain T for which one (or more) of the following holds: (i) the ratio of the lengths of P_T and P'_T is arbitrarily large even when the paths pass through the same sequence of faces; (ii) P_T and P'_T pass through two different face sequences; and (iii) there is no descending path through the face sequence crossed by P'_T.*

Proof. The idea is that a shortest path may climb over a ridge while the SDP may need to travel a long way around. We use a slightly more elaborate example to capture all the given situations. Consider a polyhedron that has a perspective view and a top view as in Figure 3.6. Let s and t be two points of equal heights, and P be the constrained path (s, p_1, p_2, p_3, t), as shown in the figure.



(a) (b)

Figure 3.6: A terrain in which an SDP is very different from a shortest path.

Let T_1 be the terrain consisting of the faces crossed by P, and (s, p_1, p'_2, p_3, t) be the shortest path in T_1. Clearly, (s, p_1, p_2, p_3, t) is an SDP in T_1. Moreover, $p_1p'_2 \perp p_2p'_2$ (and $p_3p'_2 \perp p_2p'_2$ by symmetry). Now imagine rotating T_1 around the axis defined by the line through s and t. This rotation keeps the length of (s, p_1, p'_2, p_3, t) unchanged, but changes the length of (s, p_1, p_2, p_3, t). If we rotate T_1 until the face adjacent to s becomes almost horizontal, the length of (s, p_1, p_2, p_3, t) becomes arbitrarily large. This proves the first part of the lemma.

Let T_2 be the terrain consisting of the faces visible in the top view in Figure 3.6. It is not hard to see that from s to t there are exactly two LSDPs (s, p_1, p_2, p_3, t) and (s, q_1, q_2, t), and exactly two geodesic paths (s, p_1, p'_2, p_3, t) and (s, q'_1, q'_2, t) in T_2. In the figure, the path (s, p_1, p'_2, p_3, t) is shorter than the path (s, q'_1, q'_2, t). So, (s, p_1, p'_2, p_3, t) is the shortest path from s to t. We can make the length of (s, p_1, p_2, p_3, t) greater than that of (s, q_1, q_2, t) by rotating the faces crossed by (s, p_1, p_2, p_3, t) as in the first part of the proof, while keeping the slopes of other faces unchanged. This makes (s, q_1, q_2, t) an SDP in T_2. Clearly, the SDP and the shortest path in T_2 pass through disjoint sets of faces, which proves the second part.

If we modify T_2 by removing the part of it to the right of the dashed lines in Figure 3.6(b), it is no longer possible to construct any descending path through the face sequence crossed by the shortest path (s, p_1, p_2', p_3, t). This proves the third part. □

Roy, Das and Nandy [83, Section 5] suggest a heuristic for tracing an approximate SDP P from s to t, which follows a shortest path P' until reaching a point where P' is not descending, and follows constrained segments from that point until the traced path P either reaches t, or reunites with P' in which case P starts following P' again. They point out in their conclusion that the efficacy of this method depends on whether there is any relationship between the length of an SDP and the length of a descending path through the face sequence of a shortest path. Lemma 3.5 answers this negatively.

3.3.2 Uniqueness of an SDP through given faces

Our goal in the rest of Section 3.3 is to determine a necessary and sufficient condition for an LSDP through σ from s to t. The first step towards our goal is to show the uniqueness of an LSDP through σ from s to t, which is the topic of this section. The uniqueness of a geodesic path is evident from the fact that an unfolded geodesic path is a straight line segment. Since an unfolded LSDP is not a straight line segment, the uniqueness of an LSDP is not obvious. We prove this property by formulating the problem of computing an LSDP through σ as a convex optimization problem.

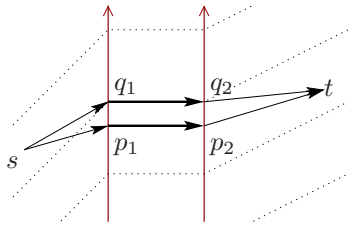


Figure 3.7: Two descending paths that follow Lemma 3.4, but of different lengths.

Note that the properties of LSDPs that we have mentioned so far do not imply the uniqueness of an LSDP through σ from s to t. In particular, even though Lemma 3.4 gives necessary and sufficient condition for an LSDP that crosses only one edge of the terrain, the condition is not sufficient for an LSDP that crosses at least two edges of the terrain. This is because Lemma 3.4 determines only the *direction* of deflection at an intermediate node of an LSDP, not the *amount* of deflection. It is possible to construct two descending paths of different lengths through a common face sequence such that both paths follow Lemma 3.4 at each intermediate node. Figure 3.7 shows two such paths, where it is not hard to see that (s, q_1, q_2, t) is longer than (s, p_1, p_2, t).

We now go into the details of our uniqueness proof. We will start with a *general* path from s to t in 3D such that the ith intermediate node of the path lies on the *line* through a_ib_i (thus the path does not necessarily lie on the terrain). More precisely, let $F(x_1, x_2, \ldots, x_k)$ denote the general path consisting of the line segments sp_1, p_1p_2, p_2p_3, $\ldots$, $p_{k-1}p_k$ and p_kt in this order, where for all $i \in [1, k]$, p_i is any point on the line through a_ib_i, and x_i is a parameter to denote the position of p_i on line a_ib_i. For all $i \in [1, k]$ such that a_ib_i is non-horizontal, the

height of p_i uniquely determines its position. So, in these cases, we use the height of p_i as parameter x_i. For each horizontal edge a_ib_i, we use as parameter x_i the signed distance of p_i from b_i. More precisely, $x_i = \overrightarrow{b_ip_i} \cdot \overrightarrow{b_ia_i}/|a_ib_i|$ in this case.

Note that parameter x_i can be defined in many ways, and our results below remain unchanged as long as x_i varies *linearly* with the position of p_i. However, choosing the height of p_i as our parameter x_i for non-horizontal edges makes the proof of Lemma 3.7 simple and intuitive, because the constraint that the path is descending is simply expressed as $x_i \geq x_{i+1}$. This is why we have chosen to define x_i in this manner.

Let $\mathcal{L}(x_1, x_2, \ldots, x_k)$ denote the length of the path $F(x_1, x_2, \ldots, x_k)$. In other words, $\mathcal{L}(x_1, x_2, \ldots, x_k) = \sum_{i=0}^{k} |p_ip_{i+1}|$, where $p_0 = s$ and $p_{k+1} = t$. The core of our uniqueness proof is the following property of $\mathcal{L}(x_1, x_2, \ldots, x_k)$:

Lemma 3.6. *$\mathcal{L}(x_1, x_2, \ldots, x_k)$ is a strictly convex function.*

A similar result was proved by Mitchell and Papadimitriou [71, Lemma 3.6] for the case of the weighted region problem. We include a proof for completeness and in order to be meticulous about faces where the function is convex but not strictly so. More precisely, our proof shows that $|p_ip_{i+1}|$ is *not* a strictly convex function of x_i and x_{i+1} for any $i \in [1, k-1]$, yet $\mathcal{L}(x_1, x_2, \ldots, x_k)$ is a strictly convex function.

Proof. Let $(u_1, u_2, \ldots, u_k)$ and $(v_1, v_2, \ldots, v_k)$ be two lists of constants such that $u_i \neq v_i$ for at least one $i \in [1, k]$. For any real constant $\kappa \in (0, 1)$, let $w_i = \kappa u_i + (1-\kappa)v_i$ for all $i \in [1, k]$. We show that

$$\mathcal{L}(w_1, w_2, \ldots, w_k) < \kappa\mathcal{L}(u_1, u_2, \ldots, u_k) + (1-\kappa)\mathcal{L}(v_1, v_2, \ldots, v_k),$$

which proves the lemma by the definition of strict convexity (Definition 3.1).

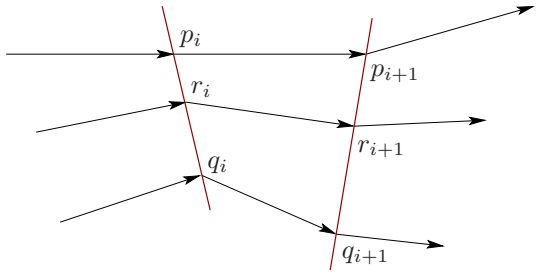


Figure 3.8: Proving strict convexity of $\mathcal{L}(x_1, x_2, \ldots, x_k)$.

Clearly, $F(u_1, u_2, \ldots, u_k)$ and $F(v_1, v_2, \ldots, v_k)$ denote two different paths. Let the ith segments of the three paths $F(u_1, u_2, \ldots, u_k)$, $F(v_1, v_2, \ldots, v_k)$ and $F(w_1, w_2, \ldots, w_k)$ be p_ip_{i+1}, q_iq_{i+1} and r_ir_{i+1} respectively (Figure 3.8). We can express the coordinates of r_i in terms of those of p_i and q_i as follows:

$$r_i = \kappa p_i + (1-\kappa)q_i.$$

Therefore,

$$\overrightarrow{r_ip_i} = p_i - r_i = p_i - \kappa p_i - (1-\kappa)q_i = (1-\kappa)(p_i - q_i) = (1-\kappa)\,\overrightarrow{q_ip_i},$$

and similarly, $\overrightarrow{r_iq_i} = \kappa\,\overrightarrow{p_iq_i}$. So, $\kappa\,\overrightarrow{r_ip_i} + (1-\kappa)\,\overrightarrow{r_iq_i} = 0$. For the same reason, $\kappa\,\overrightarrow{p_{i+1}r_{i+1}} + (1-\kappa)\,\overrightarrow{q_{i+1}r_{i+1}} = 0$. Using these two equations, we can express $\overrightarrow{r_ir_{i+1}}$ in terms of $\overrightarrow{p_ip_{i+1}}$ and $\overrightarrow{q_iq_{i+1}}$ as follows:

$$\begin{aligned}
\overrightarrow{r_ir_{i+1}} &= \kappa\,\overrightarrow{r_ir_{i+1}} + (1-\kappa)\,\overrightarrow{r_ir_{i+1}} \\
&= \kappa\,(\overrightarrow{r_ip_i} + \overrightarrow{p_ip_{i+1}} + \overrightarrow{p_{i+1}r_{i+1}}) + (1-\kappa)\,(\overrightarrow{r_iq_i} + \overrightarrow{q_iq_{i+1}} + \overrightarrow{q_{i+1}r_{i+1}}) \\
&= \kappa\,\overrightarrow{p_ip_{i+1}} + (1-\kappa)\,\overrightarrow{q_iq_{i+1}} + \\
&\qquad \kappa\,\overrightarrow{r_ip_i} + (1-\kappa)\,\overrightarrow{r_iq_i} \quad + \quad \kappa\,\overrightarrow{p_{i+1}r_{i+1}} + (1-\kappa)\,\overrightarrow{q_{i+1}r_{i+1}} \\
&= \kappa\,\overrightarrow{p_ip_{i+1}} + (1-\kappa)\,\overrightarrow{q_iq_{i+1}}.
\end{aligned}$$

Taking the lengths of $\overrightarrow{r_ir_{i+1}}$, $\overrightarrow{p_ip_{i+1}}$ and $\overrightarrow{q_iq_{i+1}}$, we get

$$|r_ir_{i+1}| \le \kappa|p_ip_{i+1}| + (1-\kappa)|q_iq_{i+1}|. \tag{3.2}$$

In Equation (3.2), the equality holds only when p_ip_{i+1} and q_iq_{i+1} are parallel to each other. Because $F(u_1, u_2, \ldots, u_k)$ and $F(v_1, v_2, \ldots, v_k)$ are different paths, and they both start at s and end at t, there are at least two $i \in [0, k]$ for which p_ip_{i+1} and q_iq_{i+1} are not parallel to each other. Considering this fact, and adding the lengths in Equation (3.2) over all $i \in [0, k]$, we get:

$$\begin{aligned}
&\sum_{i=0}^{k} |r_ir_{i+1}| < \kappa \sum_{i=0}^{k} |p_ip_{i+1}| + (1-\kappa) \sum_{i=0}^{k} |q_iq_{i+1}| \\
\Rightarrow\quad &\mathcal{L}(w_1, w_2, \ldots, w_k) < \kappa\mathcal{L}(u_1, u_2, \ldots, u_k) + (1-\kappa)\mathcal{L}(v_1, v_2, \ldots, v_k),
\end{aligned}$$

which completes the proof. □

We now determine the constraints on the variables x_i, $1 \le i \le k$, that ensure that $F(x_1, x_2, \ldots, x_k)$ is a descending path through σ. For all $i \in [1, k]$, the following constraints ensure that the intermediate nodes of $F(x_1, x_2, \ldots, x_k)$ are not outside the corresponding edges:

$$h(b_i) \le x_i \le h(a_i), \quad \text{when } h(a_i) \ne h(b_i), \tag{3.3}$$

$$\text{and} \quad 0 \le x_i \le |a_ib_i|, \quad \text{when } h(a_i) = h(b_i). \tag{3.4}$$

The constraints that ensure that $F(x_1, x_2, \ldots, x_k)$ is a descending path are: $h(p_i) \ge h(p_{i+1})$ for all $i \in [0, k]$. For each i such that a_ib_i is horizontal, $h(p_i)$ is a constant of value $H_i = h(a_i)$. Moreover, $h(p_0)$ and $h(p_k)$ are also constants of values $H_0 = h(s)$ and $H_{k+1} = h(t)$ respectively. For all other $i \in [1, k]$, $h(p_i) = x_i$. Therefore, for every $i \in [1, k]$, the corresponding height constraint expressed in terms of variables x_i's has one of the following forms:

$$x_i \ge x_{i+1} \tag{3.5}$$

$$H_i \ge x_{i+1} \tag{3.6}$$

$$x_i \ge H_{i+1} \tag{3.7}$$

$$H_i \ge H_{i+1} \tag{3.8}$$

Note that the constraint in Equation (3.8) is either always satisfied, or never satisfied. Clearly, the constraint is redundant in the former case, and there is no descending path through σ from s to t in the latter case.

Lemma 3.7. *There is a unique SDP through σ from s to t, provided that there is a descending path through σ from s to t.*

Proof. Any SDP through σ from s to t is an instance of $F(x_1, x_2, \ldots, x_k)$ because an SDP does not bend at an interior point of a face (Lemma 3.2). Moreover, the length of an SDP through σ from s to t corresponds to a local minimum of the length of $F(x_1, x_2, \ldots, x_k)$ subject to the constraints in Equations (3.3) to (3.8), i.e., a local minimum of $\mathcal{L}(x_1, x_2, \ldots, x_k)$ subject to those constraints. Now observe that the constraints in Equations (3.3) to (3.8) are linear, and therefore, the domain defined by them is convex. Since $\mathcal{L}(x_1, x_2, \ldots, x_k)$ is a strictly convex function (Lemma 3.6), it has at most one local minimum in this convex domain [22, Section 4.2.1]. Now, the existence of a descending path through σ from s to t implies that the domain is not an empty set. Moreover, the equalities in the constraints imply that the domain is a closed set. Function $\mathcal{L}(x_1, x_2, \ldots, x_k)$, therefore, has exactly one local minimum in the domain defined by the constraints. So, there is a a unique SDP through σ from s to t. □

Lemma 3.8. *There is at most one LSDP through σ from s to t.*

Proof. When there is no descending path through σ from s to t, there is clearly no LSDP through σ from s to t. Otherwise, the proof is the same as in Lemma 3.7 except that the Equations (3.3) and (3.4) are strict inequalities in this case, which implies that the domain defined by the constraints in Equations (3.3) to (3.8) is an open set, and hence $\mathcal{L}(x_1, x_2, \ldots, x_k)$ may not have a local minimum in the domain. □

Observation 3.2. *If the SDP P through σ from s to t does not go through a vertex, P is the LSDP through σ from s to t. Otherwise, there is no such LSDP.*

3.3.3 An algorithm for SDPs through given faces using convex optimization

It follows from Lemma 3.7 that we can determine an SDP through σ from s to t by solving the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \mathcal{L}(x_1, x_2, \ldots, x_k) = \sum_{i=0}^{k} |p_i p_{i+1}| \\ \text{subject to} \quad & \text{the constraints in Equations (3.3) to (3.8).} \end{aligned}$$

This is a convex optimization problem—a well-studied optimization problem where both the objective function and the constraints are convex functions of the variables involved. See Boyd and Vandenberghe [22] for extensive details on convex optimization problems and their applications.

To compute the SDP, we can solve the above optimization problem using any method available for convex optimization problems. Using the method used by Polishchuk and Mitchell [77], we get the following running-time:

Theorem 3.1. *Determining a $(1+\epsilon)$-approximate SDP through a sequence of k faces from s to t takes $O(k^{3.5} \log(\frac{1}{\epsilon}))$ time.*

Proof. We first convert the above convex optimization problem into the following equivalent problem on variables $x_1, x_2, \ldots, x_k, t_0, t_1, t_2, \ldots, t_k$:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=0}^{k} t_i \\ \text{subject to} \quad & |p_i p_{i+1}| \leq t_i, \quad \text{for } i \in [0,k], \\ & \text{and} \\ & \text{the constraints in Equations (3.3) to (3.8).} \end{aligned} \tag{3.9}$$

It is easy to show that the coordinates of p_i vary linearly with x_i for all $i \in [1,k]$. As a result, the constraint in Equation (3.9) can be written in the form $|A_i x_i + B_i x_{i+1} + C_i| \leq t_i$ for some scalar constants A_i, B_i and C_i for all $i \in [0,k]$. This makes the above optimization problem a Second-order Cone Program, for which finding a $(1+\epsilon)$-approximate solution takes $O(k^{3.5}\log(\frac{1}{\epsilon}))$ time [67, Section 1.4]. (A rigorous derivation of this running time appears in Algorithms and Theory of Computation Handbook [19, Section 33.6], which focuses on a more general class of problems called *Semidefinite programming*.) □

3.3.4 Generalized Snell's Law

In this section we will show that the direction of any free segment of an LSDP determines the direction of the next free segment on the path, even if there are constrained segments in between the two free segments. The equation relating these two directions is similar to Snell's law of refraction of light, i.e., $\mu_1 \sin\theta_1 = \mu_2 \sin\theta_2$, so we call it *Generalized Snell's Law*. In fact, Generalized Snell's Law has an important role in determining the direction of *any* segment, whether free or constrained, as we will see in Section 3.3.5.

We start by defining a few parameters of the terrain that will be used in Generalized Snell's Law. We first define a unit-height vector. For any non-level edge $a_i b_i$ in σ, we call the vector $\frac{1}{h(a_i)-h(b_i)}\overrightarrow{b_i a_i}$ a *unit-height vector along* $a_i b_i$, and denote it by ν_i.

Definition 3.3 (μ_i)**.** *For all $i \in [1,k]$, the constant μ_i is defined as follows:*

(i) if $a_i b_i$ is not level, μ_i is the length of vector ν_i, i.e., $\mu_i = \frac{1}{h(a_i)-h(b_i)}|a_i b_i|$; and

(ii) otherwise, $\mu_i = 1$.

Definition 3.4 ($\lambda_{i,j}$)**.** *For $i, j \in [1,k]$ such that $i \leq j$, the constant $\lambda_{i,j}$ is defined as follows:*

(i) $\lambda_{i,i} = 0$; and

(ii) if $i < j$, $\lambda_{i,j}$ is the rate at which the length of a constrained path from a point on edge $a_i b_i$ to a point on edge $a_j b_j$ changes as a function of height, provided such a path exists.

It follows immediately that $\lambda_{i,j} = \sum_{r=i}^{j-1} \lambda_{r,r+1}$, provided $\lambda_{r,r+1}$ is defined for all $r \in [i, j-1]$. We can express $\lambda_{i,j}$ in terms of ν_i and ν_j using the following lemma:

Lemma 3.9. *For any $i \in [1, k-1]$ such that there exists a level line from a point in edge $a_i b_i$ to a point in edge $a_{i+1} b_{i+1}$, $\lambda_{i,i+1} = |\nu_{i+1} - \nu_i|$ if $b_i = b_{i+1}$ and $\lambda_{i,i+1} = -|\nu_{i+1} - \nu_i|$ otherwise.*

Proof. Let z be a height such that there exists an interior point p_i in edge a_ib_i and an interior point p_{i+1} in edge $a_{i+1}b_{i+1}$ with $h(p_i) = h(p_{i+1}) = z$. The existence of a level line from a_ib_i to $a_{i+1}b_{i+1}$ implies that both of these edges are non-level. Therefore, $h(a_i) > z > h(b_i)$ and $h(a_{i+1}) > z > h(b_{i+1})$. These inequalities together with the fact that f_i is a triangle imply that either $b_i = b_{i+1}$ (Figure 3.9(a)), or $a_i = a_{i+1}$ (Figure 3.9(b)).

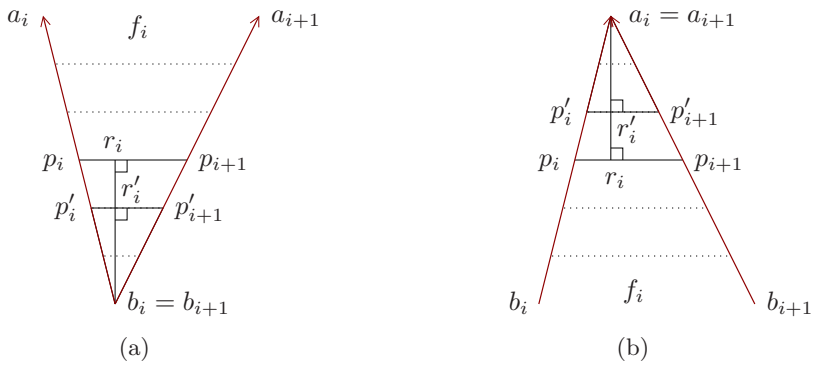


Figure 3.9: Expressing $\lambda_{i,i+1}$ in terms of ν_i and ν_{i+1}.

We now prove the lemma for each of the cases $b_i = b_{i+1}$ and $a_i = a_{i+1}$.

When $b_i = b_{i+1}$ (Figure 3.9(a)), let $\overrightarrow{b_ip'_i}$ and $\overrightarrow{b_ip'_{i+1}}$ be the vectors ν_i and ν_{i+1} respectively. Clearly, $h(p'_i) = h(p'_{i+1}) = h(b_i) + 1$ by the definition of a unit-height vector, and hence, p_ip_{i+1} and $p'_ip'_{i+1}$ are parallel to each other. So, $\triangle b_ip_ip_{i+1}$ and $\triangle b_ip'_ip'_{i+1}$ are similar triangles. It follows trivially that

$$\frac{|p_ip_{i+1}|}{|p'_ip'_{i+1}|} = \frac{|b_ir_i|}{|b_ir'_i|}, \tag{3.10}$$

where $r_i \in p_ip_{i+1}$ and $r'_i \in p'_ip'_{i+1}$ are two points such that $b_ir_i \perp p_ip_{i+1}$ and $b_ir'_i \perp p'_ip'_{i+1}$. Now, $|b_ir_i| = h(p_i) - h(b_i) = z - h(b_i)$, and $|b_ir'_i| = h(p'_i) - h(b_i) = 1$. Moreover, $\overrightarrow{p'_ip'_{i+1}} = \nu_{i+1} - \nu_i$. So, Equation (3.10) implies:

$$\begin{aligned} & \frac{|p_ip_{i+1}|}{|\nu_{i+1} - \nu_i|} = \frac{z - h(b_i)}{1} \\ \Rightarrow \quad & |p_ip_{i+1}| = |\nu_{i+1} - \nu_i| \cdot (z - h(b_i)) \\ \Rightarrow \quad & \frac{\mathrm{d}}{\mathrm{d}z}|p_ip_{i+1}| = |\nu_{i+1} - \nu_i| = \lambda_{i,i+1} \end{aligned}$$

by definition.

When $a_i = a_{i+1}$, (Figure 3.9(b)), let $\overrightarrow{p'_ia_i}$ and $\overrightarrow{p'_{i+1}a_i}$ be the vectors ν_i and ν_{i+1} respectively, and $r_i \in p_ip_{i+1}$ and $r'_i \in p'_ip'_{i+1}$ be two points such that $a_ir_i \perp p_ip_{i+1}$ and $a_ir'_i \perp p'_ip'_{i+1}$. As in the previous case, we have:

$$\begin{aligned} & \frac{|p_ip_{i+1}|}{|p'_ip'_{i+1}|} = \frac{|a_ir_i|}{|a_ir'_i|} \\ & \frac{|p_ip_{i+1}|}{|\nu_{i+1} - \nu_i|} = \frac{h(a_i) - z}{1} \\ \Rightarrow \quad & |p_ip_{i+1}| = |\nu_{i+1} - \nu_i| \cdot (h(a_i) - z) \end{aligned}$$

$$\Rightarrow \quad \frac{\mathrm{d}}{\mathrm{d}z}|p_i p_{i+1}| = -|\nu_{i+1} - \nu_i| = \lambda_{i,i+1}$$

by definition. □

We establish Generalized Snell's Law in the following lemma. The lemma focuses on a path that does not always obey the height constraint (i.e., the constraint of being descending). More precisely, the lemma is about a path in which all the segments are constrained except the first segment and the last segment which can be ascending or descending. As we have mentioned before, the significance of the lemma will become clear in Section 3.3.5 when we will combine this lemma with the height constraint of our paths in order to characterize the bend angles of an LSDP in Theorem 3.2. Recall that σ is the face sequence $(f_0, f_1, \ldots, f_k)$, where for all $i \in [1, k]$, $f_{i-1} \cap f_i$ is the edge $a_i b_i$ with $h(a_i) \geq h(b_i)$.

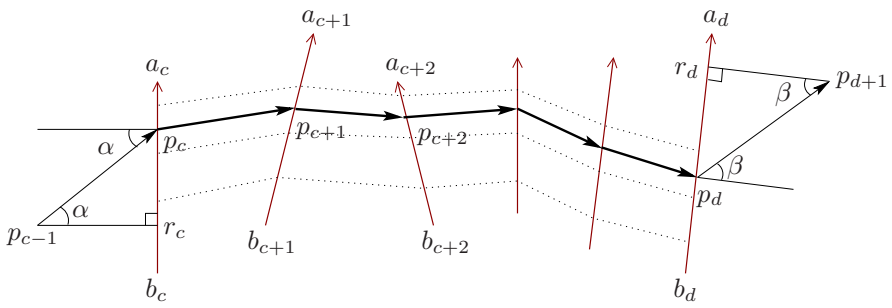


Figure 3.10: Path Q with all intermediate nodes at height $h(p_c)$.

Lemma 3.10. *Let c and d be two indices such that $1 \leq c \leq d \leq k$, and p_{c-1} and p_{d+1} be points interior to faces f_{c-1} and f_d respectively. Let $Q = (p_{c-1}, p_c, p_{c+1}, \ldots, p_{d-1}, p_d, p_{d+1})$ be a (not necessarily descending) path such that p_i is an interior point of edge $a_i b_i$ for all $i \in [c, d]$, and the subpath from p_c to p_d is constrained (hence, $h(p_i) = h(p_c)$ for all $i \in [c, d]$). Let α and β be respectively the entering angle at $a_c b_c$ and the exiting angle at $a_d b_d$. See Figure 3.10. Then:*

(i) $\mu_c \sin\alpha + \lambda_{c,d} = \mu_d \sin\beta$ *iff the length of Q is increased by moving the constrained subpath $(p_c, p_{c+1}, \ldots, p_d)$ upward (i.e., towards the a_i's) and downward (i.e., towards the b_i's);*

(ii) $\mu_c \sin\alpha + \lambda_{c,d} > \mu_d \sin\beta$ *iff the length of Q is increased by moving the constrained subpath $(p_c, p_{c+1}, \ldots, p_d)$ upward (i.e., towards the a_i's); and*

(iii) $\mu_c \sin\alpha + \lambda_{c,d} < \mu_d \sin\beta$ *iff the length of Q is increased by moving the constrained subpath $(p_c, p_{c+1}, \ldots, p_d)$ downward (i.e., towards the b_i's).*

Proof. Let x_i be the distance from b_i to p_i for all $i \in [c, d]$, and let $\mathcal{L}$ be the length of Q. As we move the constrained subpath up and down, Q, $\mathcal{L}$, α and β are functions of x_c. We will sometimes emphasize this using notation, e.g., $Q(x_c)$.

We prove the lemma by differentiating $\mathcal{L}$ with respect to x_c. More precisely, we first show that $\frac{\mathrm{d}}{\mathrm{d}x_c}\mathcal{L} = \sin\alpha + \frac{\lambda_{c,d}}{\mu_c} - \frac{\mu_d}{\mu_c}\sin\beta$, and then we prove each part of the lemma.

We have:

$$\mathcal{L} = |p_{c-1}p_c| + \sum_{i=c}^{d-1} |p_i p_{i+1}| + |p_d p_{d+1}|. \tag{3.11}$$

Since Q does not pass through the vertices of the terrain, we have for all $i \in [c, d]$, $x_i \in (0, |a_i b_i|)$. As a result, both $Q(x_c - \delta)$ and $Q(x_c + \delta)$ are defined for an arbitrarily small constant $\delta > 0$, and hence, $\mathcal{L}$ is differentiable at x_c. Let r_c and r_d be two points on respectively line $a_c b_c$ and line $a_d b_d$ such that $p_{c-1}r_c \perp a_c b_c$ and $p_{d+1}r_d \perp a_d b_d$ (Figure 3.10). Let l_c be the distance from b_c to r_c, and l_d be the distance from b_d to r_d.

The first term on the right hand side of Equation (3.11) can be written as follows:

$$\begin{aligned} |p_{c-1}p_c| &= \sqrt{|p_{c-1}r_c|^2 + |r_c p_c|^2} \\ &= \sqrt{|p_{c-1}r_c|^2 + (|b_c p_c| - |b_c r_c|)^2} \\ &= \sqrt{|p_{c-1}r_c|^2 + (x_c - l_c)^2}. \end{aligned}$$

Since the positions of p_{c-1} and r_c do not depend on x_c, differentiating the above term with respect to x_c yields:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}x_c}|p_{c-1}p_c| &= \frac{x_c - l_c}{\sqrt{|p_{c-1}r_c|^2 + (x_c - l_c)^2}} \cdot \frac{\mathrm{d}}{\mathrm{d}x_c}(x_c - l_c) \\ &= \sin\alpha \cdot \frac{\mathrm{d}}{\mathrm{d}x_c}(x_c - l_c) \\ &= \sin\alpha \ . \end{aligned}$$

The last term on the right hand side of Equation (3.11) can be written as follows:

$$\begin{aligned} |p_d p_{d+1}| &= \sqrt{|r_d p_{d+1}|^2 + |r_d p_d|^2} \\ &= \sqrt{|r_d p_{d+1}|^2 + (|b_d r_d| - |b_d p_d|)^2} \\ &= \sqrt{|r_d p_{d+1}|^2 + (l_d - x_d)^2}. \end{aligned}$$

Since the positions of p_{d+1} and r_d do not depend on x_d, differentiating the above term with respect to x_d yields:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}x_d}|p_d p_{d+1}| &= \frac{l_d - x_d}{\sqrt{|r_d p_{d+1}|^2 + (l_d - x_d)^2}} \cdot \frac{\mathrm{d}}{\mathrm{d}x_d}(l_d - x_d) \\ &= \sin\beta \cdot \frac{\mathrm{d}}{\mathrm{d}x_d}(l_d - x_d) \\ &= -\sin\beta \ . \end{aligned}$$

We now have two cases as follows. When $c = d$, $\sum_{i=c}^{d-1} |p_i p_{i+1}| = 0$. So, by differentiating Equation (3.11) with respect to $x_c = x_d$, we get:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}x_c}\mathcal{L} &= \frac{\mathrm{d}}{\mathrm{d}x_c}|p_{c-1}p_c| + 0 + \frac{\mathrm{d}}{\mathrm{d}x_d}|p_d p_{d+1}| \\ &= \sin\alpha - \sin\beta \\ &= \sin\alpha + \frac{\lambda_{c,d}}{\mu_c} - \frac{\mu_d}{\mu_c}\sin\beta, \end{aligned}$$

since $\mu_c = \mu_d$ and $\lambda_{c,d} = \lambda_{c,c} = 0$.

When $c < d$, we have $h(a_i) > h(b_i)$ for all $i \in [c, d]$ since a constrained segment does not intersect a level edge. Let $z = h(p_c)$. Clearly, for all i, we can express x_i in terms of z as follows:

$$x_i = \frac{z - h(b_i)}{h(a_i) - h(b_i)} |a_i b_i| = \mu_i (z - h(b_i)).$$

This implies that $\frac{\mathrm{d}x_i}{\mathrm{d}z} = \mu_i$. Moreover, because $h(a_i) > h(p_i) = z > h(b_i)$ for all $i \in [c, d]$, $\lambda_{i,i+1}$ is defined for all $i \in [c, d-1]$. By differentiating Equation (3.11) with respect to x_c, we have:

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}x_c}\mathcal{L} &= \frac{\mathrm{d}}{\mathrm{d}x_c}|p_{c-1}p_c| + \frac{\mathrm{d}}{\mathrm{d}x_c}\sum_{i=c}^{d-1}|p_i p_{i+1}| + \frac{\mathrm{d}}{\mathrm{d}x_c}|p_d p_{d+1}| \\
&= \sin\alpha + \left(\frac{\mathrm{d}}{\mathrm{d}z}\sum_{i=c}^{d-1}|p_i p_{i+1}|\right) \cdot \frac{\mathrm{d}z}{\mathrm{d}x_c} + \frac{\mathrm{d}}{\mathrm{d}x_d}|p_d p_{d+1}| \cdot \frac{\mathrm{d}x_d}{\mathrm{d}z} \cdot \frac{\mathrm{d}z}{\mathrm{d}x_c} \\
&= \sin\alpha + \left(\frac{\mathrm{d}}{\mathrm{d}z}\sum_{i=c}^{d-1}|p_i p_{i+1}|\right) \cdot \frac{1}{\mu_c} - \sin\beta \cdot \frac{\mu_d}{\mu_c} \\
&= \sin\alpha + \frac{\lambda_{c,d}}{\mu_c} - \frac{\mu_d}{\mu_c}\sin\beta,
\end{aligned}$$

by the definition of $\lambda_{c,d}$.

Therefore, in both cases, $\frac{\mathrm{d}}{\mathrm{d}x_c}\mathcal{L} = \sin\alpha + \frac{\lambda_{c,d}}{\mu_c} - \frac{\mu_d}{\mu_c}\sin\beta$. Each part of the lemma then follows immediately:

(i) Moving the constrained subpath $(p_c, p_{c+1}, \ldots, p_d)$ upward and downward increases the length of Q iff

$$\begin{aligned}
& \mathcal{L} \text{ is locally minimum at } x_c \\
\Leftrightarrow\ & \frac{\mathrm{d}}{\mathrm{d}x_c}\mathcal{L} = 0 \\
\Leftrightarrow\ & \mu_c \sin\alpha + \lambda_{c,d} = \mu_d \sin\beta.
\end{aligned}$$

(ii) Moving the constrained subpath $(p_c, p_{c+1}, \ldots, p_d)$ upward increases the length of Q iff

$$\begin{aligned}
& \mathcal{L} \text{ increases with a small increase in } x_c \\
\Leftrightarrow\ & \frac{\mathrm{d}}{\mathrm{d}x_c}\mathcal{L} > 0 \\
\Leftrightarrow\ & \mu_c \sin\alpha + \lambda_{c,d} > \mu_d \sin\beta.
\end{aligned}$$

(iii) Moving the constrained subpath $(p_c, p_{c+1}, \ldots, p_d)$ downward increases the length of Q iff

$$\begin{aligned}
& \mathcal{L} \text{ increases with a small decrease in } x_c \\
\Leftrightarrow\ & \frac{\mathrm{d}}{\mathrm{d}x_c}\mathcal{L} < 0 \\
\Leftrightarrow\ & \mu_c \sin\alpha + \lambda_{c,d} < \mu_d \sin\beta.
\end{aligned}$$

□

Note that when $c < d$, the edge $a_i b_i$ is non-level for all $i \in [c, d]$, since a constrained segment does not intersect a level edge. However, when $c = d$, the path from p_c to p_d is a trivial constrained path, and the edge $a_c b_c$ can be level. (In order to cover this latter case, the statement of the above lemma expresses Q as a function of p_c, even though it may appear more intuitive to express Q as a function of the height of the constrained subpath.)

3.3.5 Complete characterization

We will now define the notion of a path segment being *compatible* with the preceding part of the path to prove the main result of this section:

Theorem 3.2. *A descending path that does not pass through a vertex of the terrain is an LSDP iff each segment of the path is compatible with the part of the path preceding the segment.*

Viewed another way, compatibility allows us to extend an LSDP P into a new face by adding one more segment. Intuitively, if P contains at least one free segment then it extends via a free segment in the direction determined by Generalized Snell's Law, provided the direction is descending. If the direction is not descending, the path must bend downward to follow a level line. If P contains no free segment then it can be extended via a constrained segment, or via a free segment in a certain range of directions. More precisely:

Definition 3.5 (Compatible Segment)**.** *Let $P = (p_0, p_1, p_2, \ldots, p_k, p_{k+1})$ be a descending path through σ from a point $p_0 \in f_0 - f_1$ to a point $p_{k+1} \in f_k - f_{k-1}$ such that p_i is an interior point of $a_i b_i$ for all $i \in [1, k]$, and the part P_0 of P from p_0 to p_k is an LSDP. Let α_i and β_i be respectively the entering angle and the exiting angle of P at $a_i b_i$ for all $i \in [1, k]$.*

When P_0 contains no free segment, the segment $p_k p_{k+1}$ is said to be compatible *with P_0 if either:*

(i) $p_k p_{k+1}$ is a free segment and $\mu_i \sin \alpha_i + \lambda_{i,k} \leq \mu_k \sin \beta_k$ for all $i \in [1, k]$; or

(ii) $p_k p_{k+1}$ is a constrained segment.

When P_0 contains a free segment, let c be the largest index in $[1, k]$ such that $p_{c-1} p_c$ is a free segment. In this case, the segment $p_k p_{k+1}$ is said to be compatible *with P_0 if either:*

(i) $p_k p_{k+1}$ is a free segment and $\mu_c \sin \alpha_c + \lambda_{c,k} = \mu_k \sin \beta_k$, or

(ii) $p_k p_{k+1}$ is a constrained segment and $\mu_c \sin \alpha_c + \lambda_{c,k} \geq \mu_k \sin \beta_k$.

To prove Theorem 3.2, it is sufficient to show that a descending path P is an LSDP iff the last segment of P is compatible with the rest of the path, provided that the rest of the path is an LSDP. This is because we can then use induction on the number of edges on the path to prove Theorem 3.2. The forward direction of this "iff" statement follows quite directly from Lemma 3.10: we must show that if a path is an LSDP then its last segment is compatible. We

prove the contrapositive: if the last segment is not compatible then Lemma 3.10 allows us to perturb the path slightly to shorten it, which implies that the path was not an LSDP. For the other direction of the "iff" statement we must prove that if we add a compatible segment on to an LSDP then we get an LSDP. Our starting point is the property that an LSDP can always be extended (while remaining an LSDP) unless it arrives at a level edge and the next face goes uphill. The property seems intuitive, but depends on a continuity property that moving the endpoint of an SDP a small amount results in small movements of the intermediate nodes of the path. This continuity property is not obvious since a small change in the position of a node can abruptly change a free segment into a constrained one, or vice versa. As a result, perturbing a node can merge two sequences of constrained segments into one, or can split one such sequence into two. This implies that the two perturbed positions of the SDP may involve drastically different values of parameters μ_i's and $\lambda_{i,j}$'s. Despite such abrupt changes in parameters, the two paths still lie close to each other, which we will prove in Lemma 3.12 using the property that two SDPs from s through σ are diverging as we move away from s (Lemma 3.11). Note that Lemmas 3.11 and 3.12 deal with SDPs, not LSDPs. In other words, the paths in these two lemmas are allowed to go through a vertex while they pass through σ.

Lemma 3.11. *Let p_{k+1} and q_{k+1} be two points in $f_k - f_{k-1}$, and P and Q be the SDPs through σ from s to p_{k+1} and q_{k+1} respectively. Let $p_k = P \cap a_k b_k$ and $q_k = Q \cap a_k b_k$. If $p_k \neq q_k$, then there is no intersection between the ray that starts at p_k and goes through p_{k+1} and the ray that starts at q_k and goes through q_{k+1}.*

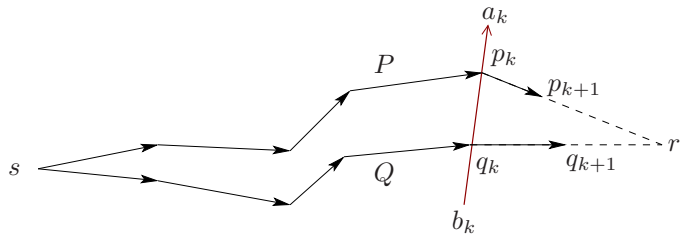


Figure 3.11: Proving that two SDPs from s are diverging.

Proof. Suppose for contradiction that the lemma is not true. Let r be the intersection point of the lines $p_k p_{k+1}$ and $q_k q_{k+1}$ (Figure 3.11). By our assumption, r and p_{k+1} lie on the same side of $a_k b_k$. We now have two cases as follows:

Case A: If $r \in f_k$, the two paths obtained by extending P and Q to point r are SDPs through σ from s to r, and they are different paths since $p_k \neq q_k$. This contradicts Lemma 3.7.

Case B: If $r \notin f_k$, we modify the (unfolded) terrain by making f_k bigger as follows. We keep the position of $a_k b_k$ unchanged, and shift each of the two other edges of f_k further away from p_k while keeping the edge parallel to its original position, possibly shrinking other faces into line segments. By making f_k sufficiently big, we make sure that r lies in the modified f_k. This modification keeps the part of P from s to p_{k+1} unchanged because there is a unique SDP through σ from s to p_{k+1} (Lemma 3.7). Therefore, extending P to point r yields an SDP through σ from s to r in the modified terrain. The same is true for the extension of Q to point r. Thus we have a contradiction as in Case A.

□

Using Lemma 3.11, we will now show in the following lemma that although perturbing a node of an SDP can abruptly change a free segment into a constrained one, or vice versa, its effect on the whole path is not abrupt. More precisely, moving the endpoint of an SDP in a continuous manner while maintaining the optimality of the path results in a continuous movement of each intermediate node.

Lemma 3.12. *Let ε be a small positive constant, P be the SDP through σ from s to a point $p_{k+1} \in f_k - f_{k-1}$, and $p_k = P \cap a_k b_k$. Then there exists a constant $\delta > 0$ such that for every point $q_{k+1} \in f_k - f_{k-1}$ lying within distance δ from p_{k+1} and on the right [left] side of $p_k p_{k+1}$, the SDP through σ from s to q_{k+1} intersects $a_k b_k$ at a point that lies within distance ε from p_k and on the right [respectively left] side of $p_k p_{k+1}$, provided that such an SDP exists.*

Proof. We prove the lemma in two steps. We first construct an SDP Q from s through σ that lies on the right [left] side of $p_k p_{k+1}$ in f_k, and intersects $a_k b_k$ within distance ε from p_k. We then determine the value of δ from P and Q.

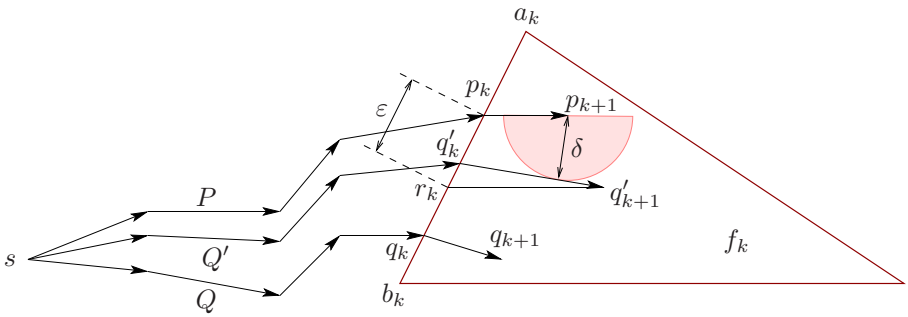


Figure 3.12: Defining δ for the continuity argument in Lemma 3.12.

For the first step, let q_{k+1} be any point in $f_k - f_{k-1}$ on the right [left] side of $p_k p_{k+1}$ such that the SDP Q through σ from s to q_{k+1} exists. Let $q_k = Q \cap a_k b_k$. If $|p_k q_k| \leq \varepsilon$, we are done with the first step. Otherwise we modify Q as follows to ensure that $|p_k q_k|$ decreases to at most ε. Let r_k be a point in $p_k q_k$ such that $|p_k r_k| = \varepsilon$, and q'_{k+1} be a point in $f_k - f_{k-1}$ such that $r_k q'_{k+1} \parallel p_k p_{k+1}$ (Figure 3.12). Let Q' be the SDP through σ from s to q'_{k+1}. Such a path exists by Lemma 3.7 as at least one of $p_k q'_{k+1}$ and $q_k q'_{k+1}$ is a descending segment that yields a descending path from s. Because the rays $p_k p_{k+1}$ and $q'_k q'_{k+1}$ are non-converging (Lemma 3.11), $|p_k q'_k| < |p_k r_k| = \varepsilon$. Path Q' is then the required path.

For the second step, let δ be the distance of p_{k+1} from the line through the last segment of the path constructed in the first step. It then follows from Lemma 3.11 that for any point $p \in f_k - f_{k-1}$ such that p lies on the right [respectively left] side of $p_k p_{k+1}$ and inside the ball defined by center p_{k+1} and radius δ, the SDP through σ from s to p must intersect $a_k b_k$ at a point in $p_k q'_k$, i.e., within distance ε from p_k. □

We will point out a minor issue that will be helpful for our algorithm in Section 3.3.7. It may appear from Lemma 3.12 that $h(p_k) > h(q_k)$ implies $h(p_{k+1}) > h(q_{k+1})$. However, this is

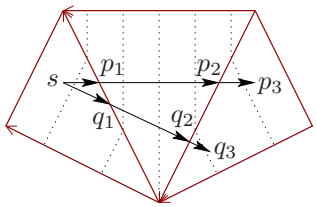


Figure 3.13: Two LSDPs from s with $h(p_1) > h(q_1)$ and $h(p_2) < h(q_2)$.

not always the case. For example, the two descending paths (s, p_1, p_2, p_3) and (s, q_1, q_2, q_3) in Figure 3.13 are SDPs, and they have $h(p_1) > h(q_1)$, but $h(p_2) < h(q_2)$ and $h(p_3) < h(q_3)$.

Our proof of Theorem 3.2 relies on Lemma 3.10 as well as the property that an LSDP can always be extended (while remaining an LSDP) unless it arrives at a level edge and the next face goes uphill. We give a careful proof of this property below.

Lemma 3.13. *Let P_0 be an LSDP through σ from s to an interior point p_k of $a_k b_k$. If an interior point of f_k is reachable from p_k following a descending segment, then there is an interior point p_{k+1} of f_k such that the path constructed by concatenating P_0 and $p_k p_{k+1}$ is an LSDP.*

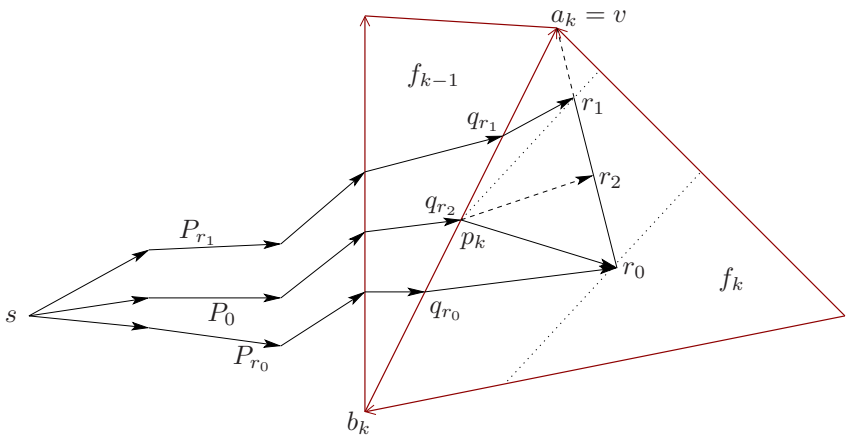


Figure 3.14: Proving the existence of a segment that extends the LSDP P_0 in Lemma 3.13.

Proof. For any point $r \in f_k$ that is reachable from s following a descending path through σ, we will use P_r to denote the SDP through σ from s to r, and q_r to denote the last intermediate node of P_r (i.e., $q_r = P_r \cap a_k b_k$). In Figure 3.14, P_{r_0}, P_{r_1} and P_{r_2} are three instances of P_r, and q_{r_0}, q_{r_1} and q_{r_2} are the corresponding instances of q_r.

Let r_0 be an interior point of f_k that is reachable from p_k following a descending segment. Lemma 3.7 implies that P_{r_0} exists. Now let v be the vertex of $a_k b_k$ such that v and q_{r_0} lie on the opposite sides of line $p_k r_0$, and let r_1 be the farthest point from r_0 in line segment $r_0 v$ that is reachable from p_k following a descending segment. Note that when $h(p_k) \geq h(v)$, $r_1 = v$; otherwise, r_1 is the intersection point of $r_0 v$ with the level line of f_k at p_k (Figure 3.14 shows the latter case). Every point in $r_0 r_1$ is reachable from p_k following a descending segment, and P_r is defined for all $r \in r_0 r_1$ by Lemma 3.7.

We first claim that q_{r_1} is a point in segment vp_k. This is because, when $h(p_k) \geq h(v)$ (which implies that $r_1 = v$), we have $q_{r_1} = r_1$ since $q_{r_1} \neq r_1$ would imply P_{r_1} contains a part of edge $a_k b_k$ which would contradict with Lemma 3.3 at face f_{k-1}. So, $q_{r_1} = r_1 = v \in vp_k$ in this case. On the other hand, when $h(p_k) < h(v)$, any point in $a_k b_k$ outside segment vp_k has a height less than $h(r_1) = h(p_k)$, which implies that $q_{r_1} \in vp_k$ as $q_{r_1} r_1$ is a descending segment. The claim thus holds in both the cases.

It then follows immediately that $p_k \in q_{r_0} q_{r_1}$. Now Lemma 3.12 implies that if we move r gradually from r_0 to r_1, q_r will move from q_{r_0} to q_{r_1} in a continuous manner. So, there is a point $r_2 \in r_0 r_1$ such that $q_{r_2} = p_k$. The part of P_{r_2} from s to p_k must be the same as P_0, which implies that P_{r_2} is an LSDP. Therefore, r_2 is the required point p_{k+1}. □

The main result of this section follows, which establishes that an LSDP can be extended onward by adding to its end a compatible segment.

Theorem (Theorem 3.2)**.** *A descending path that does not pass through a vertex of the terrain is an LSDP iff each segment of the path is compatible with the part of the path preceding the segment.*

Proof. We claim that a descending path P is an LSDP iff the last segment of P is compatible with the rest of the path provided that the rest of the path is an LSDP. The theorem follows immediately from the claim because we can then use induction on the number of edges on the path to prove the theorem.

To be precise about our claim, let $P = (s = p_0, p_1, p_2, \ldots, p_k, p_{k+1})$ be a descending path through σ from a point $p_0 \in f_0 - f_1$ to a point $p_{k+1} \in f_k - f_{k-1}$ such that p_i is an interior point of $a_i b_i$ for all $i \in [1, k]$, and the part P_0 of P from p_0 to p_k is an LSDP. We prove that P is an LSDP iff $p_k p_{k+1}$ is compatible with P_0.

Let α_i and β_i be respectively the entering angle and the exiting angle of P at $a_i b_i$ for all $i \in [1, k]$.

$\Rightarrow$: It is sufficient to show that if $p_k p_{k+1}$ is not compatible with P_0, then P is not an LSDP. This follows from Lemma 3.10, as we now show by considering the following two cases, depending on whether P_0 is a constrained path or not:

(i) Path P_0 contains no free segment: Since $p_k p_{k+1}$ is not compatible with P_0, $p_k p_{k+1}$ is a free segment and $\mu_i \sin \alpha_i + \lambda_{i,k} > \mu_k \sin \beta_k$ for some $i \in [1, k]$. Lemma 3.10 then implies that we can move the constrained path $(p_i, p_{i+1}, \ldots, p_k)$ downward by a small amount to make the part of the path from p_{i-1} to p_{k+1} shorter. All segments of P remain descending, and the only segment that changes its type (constrained or free) is $p_{i-1} p_i$, which becomes a free segment. Therefore, the modified path is descending.

(ii) Path P_0 contains a free segment $p_{c-1} p_c$: Since $p_k p_{k+1}$ is not compatible with P_0, either $p_k p_{k+1}$ is free and $\mu_c \sin \alpha_c + \lambda_{c,k} \neq \mu_k \sin \beta_k$, or $p_k p_{k+1}$ is constrained and $\mu_c \sin \alpha_c + \lambda_{c,k} < \mu_k \sin \beta_k$. In the first case, Lemma 3.10 implies that we can move the constrained path $(p_c, p_{c+1}, \ldots, p_k)$ upward or downward by a small amount to make the part of the path from p_{c-1} to p_{k+1} shorter. All segments of P remain descending, and none of the segments changes its type. In the second case, Lemma 3.10 implies that we can move the constrained

path $(p_c, p_{c+1}, \ldots, p_k)$ upward by a small amount to make the part of the path from p_{c-1} to p_{k+1} shorter. All segments of P remain descending as before, but in this case $p_k p_{k+1}$ becomes a free segment.

In both the cases, perturbing P decreases its length while maintaining the property of being descending, which implies that P is not an LSDP.

$\Leftarrow$: Since $p_k p_{k+1}$ is compatible with P_0, $p_k p_{k+1}$ is a descending segment. It then follows from Lemma 3.13 that there is an interior point p'_{k+1} in f_k such that $(s = p_0, p_1, p_2, \ldots, p_k, p'_{k+1})$ is an LSDP. By the forward direction of the proof, $p_k p'_{k+1}$ is compatible with P_0. In most situations, the direction of a compatible segment is unique, so $p_k p_{k+1}$ and $p_k p'_{k+1}$ are collinear, and therefore P is an LSDP. The only case where P_0 can have compatible segments in more than one direction is when P_0 is constrained. The remainder of the proof deals with this case.

We have two possibilities. When $p_k p_{k+1}$ is constrained, since P contains no free segments, any perturbation of its intermediate nodes makes the path infeasible. Therefore, P is an LSDP. When $p_k p_{k+1}$ is free, we show as follows that any descending path $Q \neq P$ through σ from s to p_{k+1} is longer than P provided that Q does not go through a vertex. This will complete the proof.

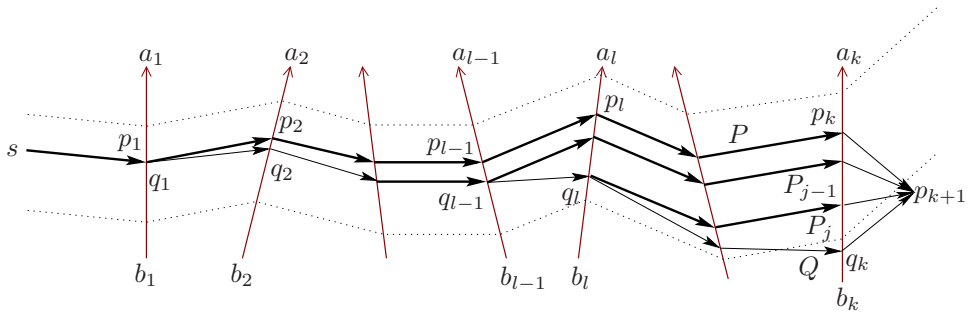


Figure 3.15: Constructing the paths $P = P_0$, P_1, P_2, $\ldots$, $P_d = Q$ to prove that Q is longer than P in the proof of Theorem 3.2.

To show that Q is longer than P, we "transform" P into Q step by step in such a way the length of the path gradually increases. We describe the construction first, and then prove in the next paragraph our claim about the length increase. Let Q be the descending path $(s = q_0, q_1, q_2, \ldots, q_k, q_{k+1} = p_{k+1})$, where q_i in an interior point of $a_i b_i$ for all $i \in [1, k]$ (Figure 3.15). Because Q is descending, and every intermediate node of P has height $h(s)$, we have $h(q_i) \leq h(p_i) = h(s)$ for all $i \in [1, k]$. We now construct as follows a sequence of descending paths $P = P_0$, P_1, P_2, $\ldots$, $P_d = Q$ such that each path goes through σ from s to p_{k+1}. To construct P_1 from P_0, let l be the smallest index in $[1, k]$ such that $q_l \neq p_l$. Note that l is always defined since $Q \neq P$, and that $h(q_l) < h(p_l)$. Construct P_1 by moving the constrained subpath $(p_l, p_{l+1}, \ldots, p_k)$ of P_0 downward to height $h(q_l)$. In general, to construct P_j from P_{j-1} for any $j > 0$, let l be the smallest index in $[1, k]$ such that $q_l \neq P_{j-1} \cap a_l b_l$. Take the constrained subpath of P_{j-1} from $a_l b_l$ to $a_k b_k$, and then move this subpath downward to height $h(q_l)$ to get P_j (Figure 3.15). After d steps, where d is the number of free segments in the part of Q from s to q_k, we will have $P_d = Q$.

It is now sufficient to show that for any $j \in [1, d]$, P_j is longer than P_{j-1}. Focus on a particular j, and (as in the last paragraph) let l be the smallest index in $[1, k]$ such that $P_{j-1} \cap a_l b_l \neq q_l$ $(= P_j \cap a_l b_l)$. Consider the entering angles of $P = P_0$, P_{j-1} and P_j at $a_l b_l$, and the exiting angles of these paths at $a_k b_k$. Let α' and α'' be the entering angles of respectively P_{j-1} and P_j at $a_l b_l$. Let β' and β'' be the exiting angles of respectively P_{j-1} and P_j at $a_k b_k$. By construction, the node of P_{j-1} at $a_l b_l$ is higher than the node of P_j at $a_l b_l$, and P_{j-1} and P_j share a common node (which is q_{l-1}) at $a_{l-1} b_{l-1}$. It then follows from Definition 3.2 that $\alpha' > \alpha''$. Because the incoming segments of P and P_{j-1} at $a_l b_l$ are both constrained, we have $\alpha_l = \alpha'$. Therefore, $\alpha_l > \alpha''$. Similarly, the node p_k of P is higher than the node of P_{j-1} at $a_k b_k$, and the latter node is higher than the node of P_j at $a_k b_k$. Since P, P_{j-1} and P_j share the node p_{k+1}, we have $\beta_k < \beta' < \beta''$. Now, the compatibility of $p_k p_{k+1}$ with P_0 implies:

$$\begin{aligned} \mu_l \sin \alpha_l + \lambda_{l,k} &\leq \mu_k \sin \beta_k \\ \Rightarrow \mu_l \sin \alpha'' + \lambda_{l,k} &< \mu_k \sin \beta'', \end{aligned}$$

because all the angles here lie in the interval $(-\frac{\pi}{2}, \frac{\pi}{2})$. It then follows from Lemma 3.10 that the part of P_j from q_{l-1} to p_{k+1} is longer that the part of P_{j-1} between the same points. In other words, P_j is longer than P_{j-1}.

Therefore Q is longer than P, and thus P is an LSDP. □

Note that the proof of Theorem 3.2 does not use Lemma 3.4. In fact, it is not hard to see that Lemma 3.4 is a special case of Theorem 3.2.

3.3.6 An algorithm to trace an LSDP along a given initial direction

Using Theorem 3.2 we can trace as follows an LSDP from a given source point s if we are given the starting direction for the path. We trace a segment from s in the given direction until the segment hits an edge. When the segment is free, Theorem 3.2 gives us a unique direction to follow in the next face along which we can extend the path further to another face. On the other hand, when the initial segment is not free, Theorem 3.2 gives us a range of possible directions to follow in the next face, and we can extend the path further along any direction in this range. In general, every time the path hits an edge, in the next face we follow a unique direction if the path traced so far has a free segment; otherwise the path traced so far is constrained, and we follow a direction chosen from a range of possible directions. By maintaining an index for the last free segment in the current path, we can extend the path to another face is constant time in the real RAM model. Thus we can trace an LSDP through k faces in $O(k)$ time.

3.3.7 Another algorithm for SDPs through given faces

In Section 3.3.3 we reduced the problem of finding an SDP through a given sequence of k faces to a convex optimization problem that can be solved via an approximation algorithm in time $O(k^{3.5} \log(\frac{1}{\epsilon}))$. In this section we give an alternative $(1+\epsilon)$-approximation algorithm based on the above characterization of LSDPs (Theorem 3.2). The algorithm here takes $O\left(k^2 \log\left(\frac{kL}{\epsilon h}\right)\right)$ time, where L is the length of the longest edge of the terrain, and h is the smallest distance

of a vertex of the terrain from a non-adjacent edge (i.e., the smallest 2D height of a triangle). The new algorithm is faster in terms of k. However, the previous algorithm is independent of the geometry of the terrain, which is not the case for this algorithm.

Since we will consider only paths through face sequence σ in this section, we will omit "through σ" for ease of discussion. We assume that there exists a descending path from s to t because the required SDP is non-existent otherwise. (Our algorithm can trivially detect the latter case—the **for** loop in Line 2 fails in this case.)

As we have mentioned in Section 3.3.6, we can trace an LSDP from s using Theorem 3.2 if we are given the starting direction for the path. However, we cannot trace an LSDP from s that reaches a particular destination point in f_k. Our algorithm uses binary search to determine the starting direction of the path. In fact, the algorithm uses binary search to locate each intermediate node of the path, one at a time and from s to t, in order to keep the approximation error small, and to ensure that the algorithm works for the more general case that the optimal path from s to t is *not* an LSDP, but an SDP that may go through a vertex.

Our algorithm is given below. In brief, the part of the algorithm from the beginning to Line 8 constructs two paths Q and R that mark the "boundary" of all descending paths from s to t. Path Q [and R] is the left [respectively, right] boundary. By "left" and "right" we mean that the ith node q_i of Q [and the ith node r_i of R] lies on the left [respectively, right] of us as we go from s to t through an interior point of a_ib_i. Note that paths Q and R do not "cross" each other. Also note that $h(q_i)$ may be less than $h(r_i)$ even when $h(q_{i-1})$ is more than $h(r_{i-1})$ because of the situation depicted in Figure 3.13.

The rest of the algorithm performs the main task, i.e., performs binary search to locate each node of an approximate SDP. More precisely, in the ith iteration of the **for** loop in Line 10, the algorithm locates the ith intermediate node of the approximate path. To locate the ith node, the algorithm performs binary search in the **while** loop in Line 17 to find a point v in the part of a_ib_i in between Q and R such that the LSDP from the $(i-1)$th intermediate node of approximate path to v, when extended onward, is very close to t. The binary search stops

when the current search range on a_ib_i becomes smaller than $\delta = \frac{\epsilon h}{5k}$.

let $q'_0 = s$ and $r'_{k+1} = t$;
for $i = 1 \dots k$ **do** let $q'_i \in a_ib_i$ be the highest point such that $h(q'_{i-1}) \geq h(q'_i)$;
for $i = k \dots 1$ **do** let $r'_i \in a_ib_i$ be the lowest point such that $h(r'_i) \geq h(r'_{i+1})$;
for $i = 1 \dots k$ **do**
    let q_i and r_i be respectively the left endpoint and the right endpoint of $q'_ir'_i$;
end
let Q be the path $(q_0 = s, q_1, q_2, \dots, q_k, q_{k+1} = t)$;
let R be the path $(r_0 = s, r_1, r_2, \dots, r_k, r_{k+1} = t)$;

let $p_0 = s$;
for $i = 1 \dots k$ **do**
    $q = q_i$, $r = r_i$;
    if $h(r) > h(p_{i-1})$ **then**
        let r be the point at height $h(p_{i-1})$ on q_ir_i;
    else if $h(q) > h(p_{i-1})$ **then**
        let q be the point at height $h(p_{i-1})$ on q_ir_i;
    end
    while $|qr| \geq \delta$ **do**
        let v be the midpoint of qr;
        trace an LSDP $P(p_{i-1}, v)$ from p_{i-1} in the direction $p_{i-1}v$, until the path intersects Q or R;
        if $P(p_{i-1}, v)$ intersects Q **then** $q = v$ **else** $r = v$;
    end
    let p_i be the higher endpoint of qr;
end
return $P = (s, p_1, p_2, \dots, p_k, t)$;

Correctness and analysis

First of all, we have to make sure that the LSDP $P(p_{i-1}, v)$ traced in Line 19 is unique in the sense that we have no freedom in choosing the direction of any segment of the path. We prove in Lemma 3.15 that this is true for any interior point v of line segment qr. We use the following claim in that proof:

Lemma 3.14. *Inside the **while** loop in Line 17, all points in qr are reachable from p_{i-1} through a descending segment.*

Proof. By construction, for every point v in q_ir_i there is a point w in $q_{i+1}r_{i+1}$ such that vw is a descending segment. So, q and r are always defined before Line 17, and all points in qr are reachable from p_{i-1} through a descending segment inside the **while** loop. □

Lemma 3.15. *Inside the **while** loop in Line 17, $P(p_{i-1}, v)$ is a unique LSDP for each interior point v of line segment qr.*

Proof. Because two different constrained segments from p_{i-1} cannot intersect each other at any point other than p_{i-1}, and because all points in qr are reachable from p_{i-1} (Lemma 3.14), $p_{i-1}v$ is a free segment for each interior point v of qr. Theorem 3.2 then implies that at every node of $P(p_{i-1}, v)$, there is a unique direction along which the LSDP can be extended. □

We determine the approximation factor guaranteed by our algorithm in Lemma 3.17, using the property that P lies close to the SDP from p_{i-1} to t:

Lemma 3.16. *For all $i \in [1, k]$, the intersection point of a_ib_i and the SDP from p_{i-1} to t lies within distance δ from p_i.*

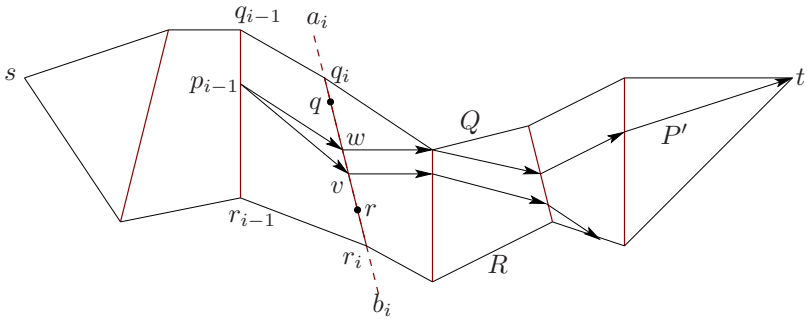


Figure 3.16: The SDP from p_{i-1} to t does not cross $P(p_{i-1}, v)$ when $v \neq w$.

Proof. Let P' denote the SDP from p_{i-1} to t, and w be the intersection point of a_ib_i and P' (Figure 3.16). Now focus on the binary search performed in the **while** loop in Line 17. It follows from Lemma 3.13 that for any interior point v of qr, $P(p_{i-1}, v)$ must intersect Q or R (or both, at t). By Lemma 3.11, $P(p_{i-1}, v)$ cannot cross P' when $v \neq w$. As a result, $P(p_{i-1}, v)$ intersects Q for any interior point v of qw, and $P(p_{i-1}, v)$ intersects R for any interior point v of wr. This is true even when P' intersects Q or R at a point before t. Therefore, the binary search maintains the invariant that w is a point in qr.

The lemma then follows since $|qr| < \delta$ in Line 22. □

Lemma 3.17. *The algorithm returns a $(1 + \epsilon)$-approximate SDP for any $\epsilon \in (0, 1]$.*

Proof. For all $i \in [0, k]$, let $l(i)$ denote the length of the path $(p_i, p_{i+1}, \ldots, p_k, t = p_{k+1})$, and $d(p)$ denote the length of the SDP through σ from any point p to t. We will first prove by induction on i that for all $i \in [0, k]$,

$$l(i) < (1 + \epsilon/2)d(p_i) + \frac{5}{2}(k - i)\delta. \tag{3.12}$$

Basis: When $i = k$, $l(k) = |p_kt| = d(p_k)$, and hence, the claim is trivially true.

Induction: Suppose that the claim is true for all $i > j$ for some $j \in [0, k-1]$ by induction. For $i = j$, we have:

$$l(j) \quad = \quad |p_jp_{j+1}| + l(j + 1)$$

$$\begin{aligned} &\leq |p_j p_{j+1}| + (1+\epsilon/2)d(p_{j+1}) + \frac{5}{2}(k-j-1)\delta \\ &\leq |p_j w| + |w p_{j+1}| + (1+\epsilon/2)\left(d(w) + |w p_{j+1}|\right) + \frac{5}{2}(k-j-1)\delta \end{aligned}$$

from the triangle inequality, where w is the intersection point of $a_{j+1}b_{j+1}$ with the SDP from p_j to t. Because $|wp_{j+1}| \leq \delta$ by Lemma 3.16, and $\epsilon \leq 1$, we have

$$\begin{aligned} l(j) &\leq |p_j w| + (2+\epsilon/2)|w p_{j+1}| + (1+\epsilon/2)d(w) + \frac{5}{2}(k-j-1)\delta \\ &\leq |p_j w| + (2+1/2)\delta + (1+\epsilon/2)d(w) + \frac{5}{2}(k-j-1)\delta \\ &= |p_j w| + (1+\epsilon/2)d(w) + \frac{5}{2}(k-j)\delta \\ &< (1+\epsilon/2)\left(|p_j w| + d(w)\right) + \frac{5}{2}(k-j)\delta \\ &= (1+\epsilon/2)d(p_j) + \frac{5}{2}(k-j)\delta. \end{aligned}$$

This establishes the relation in Equation (3.12).

Now, at $j = 0$, we have:

$$\begin{aligned} l(0) &< (1+\epsilon/2)d(p_0) + \frac{5}{2}k\delta \\ &= (1+\epsilon/2)d(s) + \frac{\epsilon h}{2} \\ &\leq (1+\epsilon)d(s) \ , \end{aligned}$$

since $h \leq d(s)$. Lemma 3.14 implies that P is a descending path, and hence P is a $(1+\epsilon)$-approximate SDP. □

Theorem 3.3. *Given a constant $\epsilon \in (0,1]$, we can determine a $(1+\epsilon)$-approximate SDP through σ from s to t in $O\left(k^2 \log\left(\frac{kL}{\epsilon h}\right)\right)$ time.*

Proof. In each iteration of the **for** loop (Line 10), the **while** of the binary search iterates $O(\log(L/\delta))$ times because the **while** divides an edge segment into halves until its length becomes less than $\delta = \frac{h\epsilon}{5k}$. In each iteration of the **while** loop, tracing LSDP $P(p_{i-1}, v)$ takes $O(k)$ time. Considering the outer loop, which iterates k times, the total time needed for the algorithm is $O\left(k^2 \log(L/\delta)\right)$, i.e., $O\left(k^2 \log\left(\frac{kL}{\epsilon h}\right)\right)$.

The theorem then follows from Lemma 3.17. □

3.4 Sequence tree approach for SDPs

So far we have only addressed the algorithmic problem of finding an LSDP through a given sequence of faces. In this section we address the problem of finding an SDP from s to t when the face sequence is not specified. Our discussion in this section follows the main framework of the algorithm by Chen and Han [28] that finds a shortest path (i.e., *not* a shortest descending

path). They construct a *sequence tree* rooted at s that captures all the possible edge/face sequences of shortest paths. The sequence tree can then be used to find the shortest s-t path for any t. Assuming for the moment that the terrain is convex, the root of the sequence tree corresponds to vertex s, and every other node of the tree corresponds to a portion of an edge of the terrain. Any path from the root s to a node v in the tree corresponds to a sequence of (portions of) edges that can be traversed by a geodesic path—i.e., edges whose unfolding can be stabbed by a *wedge* of straight lines emanating from s. Figure 3.17 shows the sequence tree for a part of a terrain, where the portions of the edges represented by the tree nodes are marked with dashed line segments and labeled with the labels of the corresponding tree nodes. The shaded path in Figure 3.17(b) corresponds to the sequence of edge portions traversed by the geodesic paths lying in the shaded wedge in Figure 3.17(a). It is straightforward to construct a sequence tree that stores edge sequences of *all* geodesic paths: to grow the tree at a node for edge portion e, take the wedge of geodesic paths arriving at e and extend it via straight lines into the unfolded face on the other side of e, adding new tree nodes for the portions of the one or two edges that are encountered. The tree is truncated at depth $2n$ (which is an upper bound on the number faces in the terrain) because a shortest path traverses each face at most once. But even so, the tree can be exponentially large because each node can have two children. Chen and Han give a method of pruning the tree. A node has two children if a wedge of geodesic paths splits at a vertex v of the terrain. They show in their *one-vertex one-split* property that it suffices to store just one split for each vertex-face pair because if two geodesic paths arrive at $v \in f$ by crossing f, the longer one cannot yield shorter paths on both sides of v.

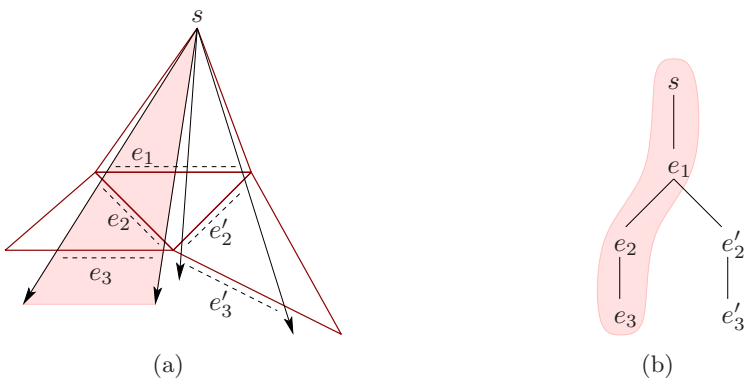


Figure 3.17: (a) A part of the terrain, and (b) the corresponding part of the sequence tree for shortest paths.

The added complication for non-convex terrains is that a geodesic path can bend at a non-convex vertex. Chen and Han make a new node in the sequence tree for each non-convex vertex encountered, starting a new set of wedges of geodesic paths. To differentiate such a node of the tree from the nodes mentioned in the previous paragraph, a tree node representing a portion of an edge is called an *edge-node*, and a tree node representing a vertex is called a *vertex-node*. A vertex v may correspond to many vertex-nodes in the tree, but the sub-tree of geodesic paths emanating from v is stored only once.

In our situation we want a sequence tree for shortest *descending* paths. There are three aspects of the construction of Chen and Han that require modification:

(i) extending an LSDP into a new face;

(ii) deciding which of two edge sequences yields a shorter path from s to a vertex v; and

(iii) proving the one-vertex one-split property.

We gave an algorithm for (i) in Section 3.3, and we prove (iii) below, but we do not have an algorithm for (ii). In other words, the main result of this section is to reduce the problem of finding SDPs to the problem finding SDPs through a given face sequence. We show that if there is an algorithm to find the SDP from a given source to a given target through a given sequence of k faces that runs in $R(k)$ time, then we can we can find an SDP from s to any vertex v in the terrain in $O(n^2R(n))$ time. In fact, the sub-problem we need to solve in $R(k)$ time in our reduction is easier than finding an SDP through a given face sequence between two points. To be more precise, we may assume that we are given not only the source vertex v_1, the target vertex v_2 and the face sequence σ, but also two LSDPs from v_1 through σ such that v_2 lies in between the two LSDPs. This additional information implies that the SDP from v_1 to v_2 through σ is an LSDP because the SDP must lie in between the two given LSDPs and hence cannot go through any other vertex. (Note that for the case of shortest paths, i.e., without the constraint of being descending, $R(n) = O(1)$.) We will show in Section 3.5 how to solve this sub-problem for two special classes of terrains. In the Conclusion we will discuss the difficulties of solving this sub-problem in a general terrain.

3.4.1 Constructing a sequence tree

Our ability to extend an LSDP onward using Theorem 3.2 makes it possible to construct a sequence tree for SDPs in the same manner as Chen and Han with one obvious modification: the paths defining an expanding wedge are LSDPs instead of geodesic paths. However, there are three aspects of the construction that are trivial for a geodesic path but complicated for our problem because of the bends along an unfolded LSDP. Before we go into the details, we clarify two relevant terms. For geodesic paths, as shown in Figure 3.17, an edge-node corresponds to an unfolded *wedge* of locally shortest (i.e., geodesic) paths traveling through the relevant face sequence. The wedge is bounded by the two geodesic paths arriving at the two ends of the portion of the edge. For LSDPs an edge-node still corresponds to a set of locally shortest paths traveling through the relevant face sequence. We will still call this a *wedge* although it is not wedge-shaped in general because of the bends along unfolded LSDPs. The two LSDPs arriving at the ends of the portion of the edge are called the *bracketing LSDPs* of the wedge. Because LSDPs do not intersect, a wedge is a polygon bounded by the bracketing LSDPs and the edge portion.

We will now focus on the three aspects of sequence trees that must be modified and generalized for SDPs.

The first issue is extending an LSDP on to the next face efficiently. We need to store enough information with each edge-node in the tree so that the two bracketing LSDPs of a wedge can be extended into the next face in an efficient manner. For each of these LSDPs, we store the index i of the last free segment as well as its direction, and the value of $\lambda_{i,j}$, where j is the index of the last node on the path. By definition, we can compute $\lambda_{i,j}$ in constant time from $\lambda_{i,j-1}$, which makes it possible to extend the path in constant time using Theorem 3.2.

The second issue arises when the wedge defined by the bracketing LSDPs splits at a vertex v of the terrain. At this point, we need to compute the LSDP from the source to v for two reasons. This path is one of the bracketing LSDPs for each side of the split, hence certain information on this path (discussed in the last paragraph) should be stored in the two children of the current node. Secondly, we need the length of this LSDP to apply the one-angle one-split property discussed later on. By assumption, we can compute this LSDP in $R(n)$ time because we know the sequence of faces to use, and we know two bracketing LSDPs through that face sequence. We add a vertex-node in the sequence tree to represent v, and store in this vertex-node the length and other parameters of the LSDP to v.

The vertex-node for v has another significance for our problem. The construction of Chen and Han adds a vertex-node in the above situation only if v is a non-convex vertex because a geodesic path does not bend at convex vertices [70]. An LSDP, on the other hand, can bend at *any* vertex of the terrain, and we don't yet know how to determine the bend angle. We solve this problem by treating *every* vertex in the manner Chen and Han treated the non-convex vertices. More precisely, after adding a vertex-node for v as mentioned in the last paragraph, we consider v as a pseudo-source, i.e., a source at a certain distance from s. Every vertex-node is labeled with its SDP distance from s, and we "trim" away all vertex-nodes for v except one that gives the minimum SDP distance to v.

The third issue is that the one-angle one-split property of a sequence tree for shortest paths [28, Lemma 1], which keeps the size of the tree polynomial, is not so obvious in our case because two SDPs from s to a vertex v through different face sequences can merge at an interior node, which is impossible for shortest paths. Lemma 3.18 establishes the one-angle one-split property for a sequence tree for descending paths.

Finally, the bends along LSDPs make another modification to the construction of Chen and Han necessary: we do not discard an edge-node if it has only one child, because otherwise it would be difficult to extend the LSDPs of a wedge onward in our case. This latter modification results in a tree of size $O(n^2)$, compared to the $O(n)$-sized tree of Chen and Han.

We now elaborate our construction. We start with a tree containing only a vertex-node representing s, and then expand the tree level by level up to depth $2n$ (which is an upper bound on the number faces in the terrain). For each vertex v we maintain the current minimum length of an SDP to v, and also, for each *face-angle* consisting of a vertex v and incident face f we maintain the minimum length of an SDP that reaches v through f. The former values are used to trim vertex-nodes from the tree, and the latter are used to trim edge-nodes using the one-vertex one-split property. We expand a node as follows:

Case 1: Expanding a vertex-node for vertex v: For each edge (v, w) such that $h(v) \geq h(w)$ we add a vertex-node child representing vertex w. We now trim as follows: If the length of the new SDP to w is greater than or equal to the current minimum, then we trim away the new vertex-node. Otherwise we trim the vertex-node for w that previously gave the minimum length. For each face f incident to v, let e be the edge of f opposite v. We add an edge-node child for the portion of e that has height at most $h(v)$.

Case 2: Expanding an edge-node for a portion of edge e arrived at through face f: Let f' be the other face incident to e, and let the other edges of face f' be e' and e'', with vertex w between them (Figure 3.18). We extend the wedge for the current edge-node into the

face f' and add an edge-node child representing the portions of e' and e'' covered by the wedge. There can be at most two such children, and when there are two, we also add a vertex-node child representing w.

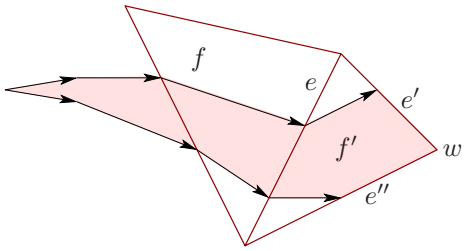


Figure 3.18: Expanding an edge-node.

In case we have added three children we now trim as follows. We have just found a new SDP to w through f'. Compare the length of this SDP to the current minimum for this face-angle (i.e., for the vertex-face pair (w, f')). The one-angle one-split property (Lemma 3.18) allows us to trim two of the three children (one edge-node and one vertex-node) either here or from the previous record-holder. Finally, we apply the rule for vertex-node trimming from Case 1.

If we must answer queries about actual SDPs, rather than their lengths, then at every vertex-node we must store information about the arriving LSDP—specifically the first or last free segment.

3.4.2 Correctness of our construction

As in Chen and Han, the most critical part of our construction is the guarantee that each face-angle contributes to at most one wedge split in the sequence tree. The following lemma provides this guarantee by establishing the one-angle one-split property for our sequence tree.

In the previous section we referred to the "face sequence" of a path because paths were assumed to not go through vertices. Because every pair of consecutive faces in a face sequence must share an edge, the sequence is in fact equivalent to an edge sequence. We must now generalize to the "face/vertex sequence" of a path, which lists all vertices and faces that the path traverses. Note that a vertex v lying in-between two faces f_1 and f_2 in a face/vertex sequence can be either the only common vertex of f_1 and f_2, or one of the two common vertices.

Lemma 3.18. *Let $f = \triangle vv_1v_2$ be any (counter-clockwise) face and let P_1 and P_2 be two SDPs from s to v that reach v through the interior of f (Figure 3.19). Suppose that P_1 [and P_2] uses face/vertex sequence σ_1 [respectively σ_2] ending with f. Let f_1 be the face across vv_1 and f_2 be the face across vv_2. (Thus we have four potential face/vertex sequences for SDPs: $\sigma_1 f_1$, $\sigma_1 f_2$, $\sigma_2 f_1$ and $\sigma_2 f_2$.)*

If the length of P_2 is less than or equal to the length of P_1, then one of the two face/vertex sequences $\sigma_1 f_1$ and $\sigma_1 f_2$ can be discarded without losing any SDPs.

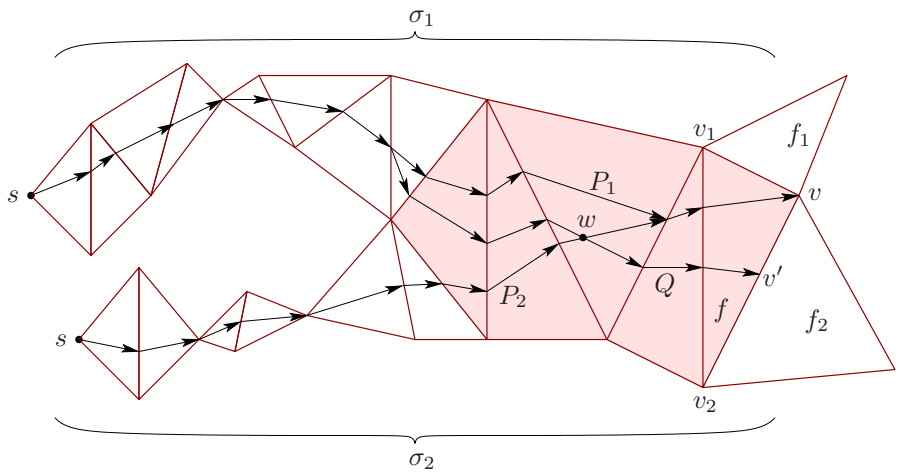


Figure 3.19: Proving that the face/vertex sequence $\sigma_1 f_2$ can be discarded when P_2 is shorter than P_1.

Proof. Let σ be the longest common suffix of σ_1 and σ_2. (Faces in σ are shaded in Figure 3.19.) Paths P_1 and P_2 enter the first face/vertex of σ from different faces/vertices, and may merge somewhere along σ before arriving together at v. Suppose without loss of generality that, traveling from v to s, the path P_2 enters σ to the left of P_1, as shown in the figure.

We will prove that the face/vertex sequence $\sigma_1 f_2$ can be discarded, i.e., that any descending path through this sequence can be replaced by a path that is at least as short and uses a different face/vertex sequence. We will use the following notation for convenience: let $\mathcal{L}(P)$ denote the length of a path P, and for two points a and b in P, let $P(a, b)$ be the subpath of P from a to b.

Suppose there is a descending path Q that follows σ_1 to reach an interior point v' of edge vv_2. Such a path must cross P_2 inside σ, say at a point w. (It does not matter that the intersection of Q and P_2 may be more than a single point.) Consider the path P_1' from s to v that consists of $Q(s, w)$ followed by $P_2(w, v)$. Path P_1' is descending from s to v in σ_1. We claim that it is at least as long as P_1. This is because by Lemma 3.7 there is a unique SDP through any given face sequence; and the same is true for face/vertex sequences. Thus, since P_1 is the unique SDP from s to v through σ_1, we have $\mathcal{L}(P_1) \leq \mathcal{L}(P_1')$.

Combining this with our hypothesis that $\mathcal{L}(P_2) \leq \mathcal{L}(P_1)$ yields $\mathcal{L}(P_2) \leq \mathcal{L}(P_1')$. Now P_2 and P_1' have a common suffix $P_2(w, v)$. Taking this away gives $\mathcal{L}(P_2(s, w)) \leq \mathcal{L}(Q(s, w))$.

Now consider the path Q' consisting of $P_2(s, w)$ followed by $Q(w, v')$. Path Q' is descending from s to v' in σ_2. Comparing Q' and Q, they have a common suffix $Q(w, v')$ and their prefixes satisfy $\mathcal{L}(P_2(s, w)) \leq \mathcal{L}(Q(s, w))$. Thus $\mathcal{L}(Q') \leq \mathcal{L}(Q)$. Therefore we do not need to consider Q, or any path through $\sigma_1 f_2$ in our search for SDPs. □

Lemma 3.18 guarantees that the size of our sequence tree is polynomial in n. We will need a stronger claim to determine the time requirement in Lemma 3.20. We claim that the number of nodes is $O(kn)$ when our level-by-level construction has finished the kth level. To be precise, we first mark the levels with natural numbers as follows: the vertex-node for s lies at level 0, and all the children of the ith level nodes lie at level $i + 1$.

Lemma 3.19. *Right after we have constructed the kth level of the above sequence tree, the tree has at most $(12k+1)n$ nodes, and at most $13n$ of these nodes are at the kth level.*

Proof. We bound the size of the tree using an accounting scheme where we partition nodes of the tree into n isolated vertex-nodes and $O(n)$ paths of edge-nodes. Each of these paths is associated with a vertex or with a face-angle (i.e., vertex-face pair). We define the paths by cutting some links in the tree—remember that this is for accounting purposes, not for real. First cut every link that connects a pair of vertex-nodes. Then at a vertex-node for vertex v, cut off all the children (which are edge-nodes), and associate with v the paths starting at the orphaned children. If an edge-node has more than one child (this happens when its wedge expanded into the next face to include a face-angle), then it has three children (two edge-nodes and one vertex-node); cut off all but one edge-node child, and associate with the face-angle the path starting at the orphaned edge-node child. Note that this cutting process makes the vertex-nodes isolated, and partitions the edge-nodes into paths.

Now we count the number of paths. By Lemma 3.18 a face-angle will be assigned at most one path. Thus the total number of paths assigned to face-angles is at most the number of face-angles. The number of paths associated with the vertex-node for vertex v is the number of faces incident to v, i.e., the number of face-angles at v. Because of our trimming of vertex-nodes, each vertex v has at most one vertex-node, and hence the total number of paths assigned to vertex-nodes is the number of face-angles. There are at most $2n$ faces and therefore at most $6n$ face-angles. So the total number of paths is at most $(1+1) \times 6n = 12n$.

Now observe that each of those $12n$ paths has at most k nodes (because the tree has depth k), and that at most one node of each path lies at the kth level of the tree. Together with n vertex-nodes, the total number of nodes and the number of nodes at the kth level become at most $(12k+1)n$ and at most $13n$ respectively. □

Lemma 3.20. *Given a vertex s in the terrain, we can construct the above sequence tree in $O(n^2 R(n))$ time and $O(n^2)$ space.*

Proof. The space requirement follows from Lemma 3.19 because the tree has depth $2n$, and the amount of information we store in a tree node is constant.

To compute the time requirement, we first determine the time needed to generate and trim one node. The child(ren) of an edge-node is generated in $O(1)$ time if the wedge of the edge-node is not split by a vertex, and in $O(R(n))$ time (by assumption) otherwise. Each child of a vertex-node is generated in $O(1)$ time. Thus each child node in the tree is generated in at most $O(R(n))$ time. Trimming one node (and *not* its descendants in the tree) takes $O(1)$ time, but there is an operation done right before trimming that takes more than $O(1)$ time. The operation is to apply the one-angle one-split property (Lemma 3.18). More precisely, when the wedge of an edge-node is split by a vertex as depicted in Figure 3.18, we have to trace backwards along the common suffix of the two LSDPs in order to decide which face/vertex sequence is unnecessary. Considering this operation a part of trimming, the time needed to trim a node becomes $O(n)$, which is $O(R(n))$.

We now claim that for any $i > 0$, our construction generates $O(n)$ nodes in the ith level, including the ones that are trimmed later on. Note that the ith level nodes are generated after node generation and trimming is complete for the first $(i-1)$ levels. The claim follows from

Lemma 3.19 as follows. There are $O(n)$ nodes in the $(i-1)$th level of the tree by Lemma 3.19, and each edge-node among them generates at most three children. So the number of the ith level nodes generated from the edge-nodes is $O(n)$. A $(i-1)$th level vertex-node for vertex v generates to at most $2d_v$ children (at most d_v of them are vertex-nodes, and at most d_v of them are edge-nodes), where d_v is the number of edges incident to v in the terrain. Because of our trimming of vertex-nodes in the first $(i-1)$ levels, each vertex has at most one vertex-node in the $(i-1)$-th level (in the first $i-1$ levels to be precise). So the number of the ith level nodes generated from the vertex-nodes is at most $2\sum_v d_v$, which is at most four times the number of edges (since each edge is counted twice in the sum). Because the number of edges is $O(n)$, the total number of the ith level nodes generated by our construction is $O(n)$.

We will now prove by induction on i that after the $(i-1)$th level of our sequence tree has been constructed, completing the ith level construction takes $O(nR(n))$ time including the time needed for trimming. We will use the following accounting scheme to incorporate trimming times with the claim in the previous paragraph: we consider that the time needed to generate a node is twice the actual time needed. Since trimming a node takes asymptotically the same time as generating a node (both are $O(R(n))$), our accounting scheme makes the trimming of nodes effectively free of cost. Now the basis of the induction is obvious: at $i = 1$ we generate $O(n)$ nodes at the first level from the vertex-node for s in $O(nR(n))$ time, and perform no trimming. For the induction step at $i > 1$, the claim in the previous paragraph implies that we generate $O(n)$ nodes at the ith level, which takes $O(nR(n))$ time. We also perform trimming, which takes no time by our accounting scheme. This completes the proof by induction.

By adding the construction times for all of the $2n$ levels of the tree, the total time becomes $O(n^2R(n))$. □

For the proof of correctness, we first consider an *untrimmed* sequence tree, i.e., a sequence tree which is constructed without trimming any sub-trees at vertex-nodes (i.e., without using the one-angle one-split property), as is done by Chen and Han. Note that the untrimmed sequence tree, which is of exponential size, is not generated by our algorithm. We are using the tree only to simplify the proof.

Lemma 3.21. *Using the above sequence tree, we can determine an SDP from s to any vertex v in $O(n)$ time.*

Proof. The untrimmed tree contains an SDP from s to v. Our trimming of vertex-nodes and edge-nodes removes only paths that are not shortest. Therefore the tree contains an SDP from s to v.

The vertex-node for v contains the length of an SDP from s to v by construction. Traversing the tree from this vertex-node to the root to get the list of vertices in P takes $O(n)$ time. An SDP between every pair of consecutive vertices in this list can be traced in time linear in the length of the path using the information stored in the tree. Therefore, the total time needed to trace the path from s to t is $O(n)$. □

Theorem 3.4. *Given a vertex s in the terrain, we can construct in $O(n^2R(n))$ time and $O(n^2)$ space a sequence tree that allows us to determine an SDP from s to any vertex v in $O(n)$ time, where $R(k)$ is the time needed to compute the SDP through a given sequence of k faces from a given source point to a given target point.*

Proof. The theorem follows immediately from Lemmas 3.20 and 3.21. □

3.4.3 Problems in approximating SDPs using sequence trees

The one thing that is missing from our Chen and Han modification is an algorithm to find an SDP through a given face sequence, and we have assumed in Theorem 3.4 that we have an $R(k)$-time algorithm to find the SDP through a sequence of k faces. Earlier in Sections 3.3.3 and 3.3.7 we have given two approximation algorithms for SDPs through given faces. It may appear that replacing the $R(k)$-time exact algorithm in Theorem 3.4 with any of our approximation algorithms would yield an efficient approximation algorithm for the SDP problem. Unfortunately this is not the case. The reason is that the one-angle one-split property (Lemma 3.18) used for trimming the sequence tree depends on exact lengths of the paths. If we instead trim the sequence tree by comparing approximate lengths, the approximation error guaranteed by the tree seems to become very large.

To be precise, consider as in Section 3.4.2 that we have an untrimmed sequence tree, and we are trimming the tree by repeatedly applying the one-angle one-split property. If we trim a part of the tree by comparing two $(1+\epsilon)$-approximate lengths, the best error bound that we can prove for the remaining tree is larger than $(1+\epsilon)$:

Lemma 3.22. *Let $f = \triangle vv_1v_2$ be any (counter-clockwise) face and let P_1 and P_2 be two SDPs from s to v that reach v through the interior of f (Figure 3.20). Suppose that P_1 [and P_2] uses face/vertex sequence σ_1 [respectively σ_2] ending with f. Let f_1 be the face across vv_1 and f_2 be the face across vv_2. (Thus we have four potential face/vertex sequences for SDPs: $\sigma_1 f_1$, $\sigma_1 f_2$, $\sigma_2 f_1$ and $\sigma_2 f_2$.)*

If we have a $(1+\epsilon)$-approximation of P_1 and a $(1+\epsilon)$-approximation of P_2, and the length of the latter approximate path is less than or equal to the length of the former, then one of the two face/vertex sequences $\sigma_1 f_1$, and $\sigma_1 f_2$ can be discarded without losing any $(1+\epsilon')$-approximate SDPs, where $\epsilon' = \epsilon\left(1 + \frac{2nL}{h}\right)$.

Proof. As in the proof of Lemma 3.18, let σ be the longest common suffix of σ_1 and σ_2. (Faces in σ are shaded in Figure 3.20.) Paths P_1 and P_2 enter the first face/vertex of σ from different faces/vertices, and may merge somewhere along σ before arriving together at v. Suppose without loss of generality that, traveling from v to s, the path P_2 enters σ to the left of P_1, as shown in the figure. We will use the following notation (as in the proof of Lemma 3.18): let $\mathcal{L}(P)$ denote the length of a path P, and for two points a and b in P, let $P(a,b)$ be the subpath of P from a to b.

Suppose there is a descending path that follows σ_1 to reach an interior point v' of edge vv_2. Let Q be the SDP from s to v' through σ_1. Path Q must cross P_2 inside σ, say at a point w. (It does not matter that the intersection of Q and P_2 may be more than a single point.) Consider the path P_1' from s to v that consists of $Q(s,w)$ followed by $P_2(w,v)$. Path P_1' is descending from s to v in σ_1. We claim that it is at least as long as P_1. This is because by Lemma 3.7 there is a unique SDP through any given face sequence; and the same is true for face/vertex sequences. Thus, since P_1 is the unique SDP from s to v through σ_1, we have: $\mathcal{L}(P_1) \leq \mathcal{L}(P_1')$. Our hypothesis about the relative lengths of approximations of P_1 and P_2 yields:

$$\mathcal{L}(P_2) \quad \leq \quad (1+\epsilon)\mathcal{L}(P_1)$$

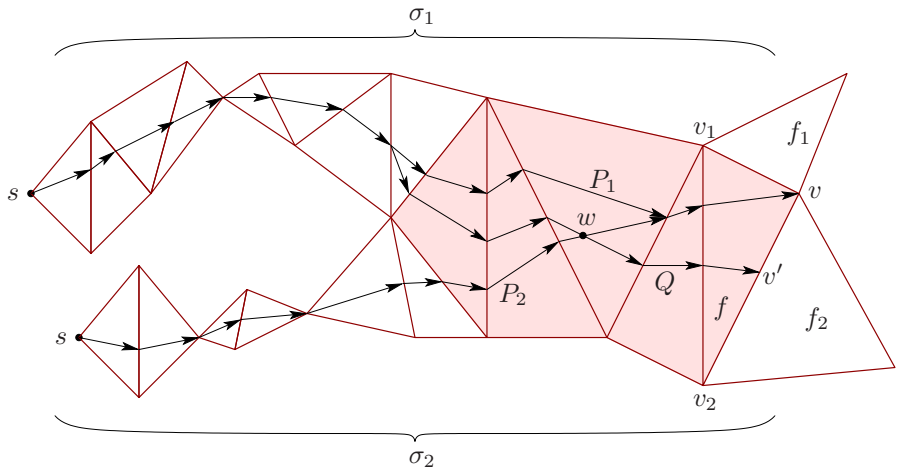


Figure 3.20: Discarding the face/vertex sequence $\sigma_1 f_2$ when an approximation of P_2 is shorter than an approximation of P_1 (Figure 3.19 repeated for convenience).

$$
\begin{aligned}
& & \leq\ & (1+\epsilon)\mathcal{L}(P_1') \\
\Rightarrow \mathcal{L}(P_2(s,w)) + \mathcal{L}(P_2(w,v)) & \leq\ & (1+\epsilon)(\mathcal{L}(Q(s,w)) + \mathcal{L}(P_2(w,v))) \\
\Rightarrow \mathcal{L}(P_2(s,w)) & \leq\ & (1+\epsilon)\mathcal{L}(Q(s,w)) \ +\ \epsilon\mathcal{L}(P_2(w,v)) \\
& =\ & \mathcal{L}(Q(s,w)) \ +\ \epsilon(\mathcal{L}(Q(s,w)) + \mathcal{L}(P_2(w,v))) \\
& =\ & \mathcal{L}(Q(s,w))\left(1+\epsilon\left(1+\frac{\mathcal{L}(P_2(w,v))}{\mathcal{L}(Q(s,w))}\right)\right) \\
& \leq\ & \mathcal{L}(Q(s,w))\left(1+\epsilon\left(1+\frac{2nL}{h}\right)\right) \\
& =\ & (1+\epsilon')\mathcal{L}(Q(s,w)) \ ,
\end{aligned}
$$

since $\mathcal{L}(Q(s,w)) \geq h$, and $\mathcal{L}(P_2(w,v)) \leq 2nL$.

Now, the length of the SDP from s to v' through σ_2 is at most:

$$
\begin{aligned}
& \mathcal{L}(P_2(s,w)) + \mathcal{L}(Q(w,v')) \\
\leq\ & (1+\epsilon')\mathcal{L}(Q(s,w)) \ +\ \mathcal{L}(Q(w,v')) \\
\leq\ & (1+\epsilon')(\mathcal{L}(Q(s,w)) + \mathcal{L}(Q(w,v'))) \\
=\ & (1+\epsilon')\mathcal{L}(Q) \ ,
\end{aligned}
$$

which implies that the face/vertex sequence $\sigma_1 f_2$ can be discarded without losing any $(1+\epsilon')$-approximate SDPs. □

Now consider the effect of repeated applications of "trimming using approximate lengths". Every time we trim a part of the sequence tree by comparing two approximate lengths, the approximation factor guaranteed by the paths in the (remaining) tree gets larger. The final approximation factor thus depends on the number of times we make such trimming. To be more precise, let P be an SDP from s to v, and $(1+\epsilon_0)$ be the approximation factor guaranteed by the algorithm for SDPs through given faces. Consider the untrimmed sequence tree we

mentioned before, i.e., the tree constructed without trimming any sub-trees at vertex-nodes. The untrimmed tree contains a path P_0 that is a $(1+\epsilon_0)$-approximation of P. Lemma 3.22 implies that after the first trimming event, there is a $\left(1+\epsilon_0\left(1+\frac{2nL}{h}\right)\right)$-approximation P_1 of P_0 in the sequence tree. Similarly, after the second trimming event, there is a $\left(1+\epsilon_0\left(1+\frac{2nL}{h}\right)^2\right)$-approximation P_2 of P_1 in the sequence tree. In general, after the ith trimming event, there is a $\left(1+\epsilon_0\left(1+\frac{2nL}{h}\right)^2\right)$-approximation P_i of P_{i-1} in the sequence tree. Since each vertex-node can result in at most one trimming event, and there are at most $12n$ such nodes (Lemma 3.19), the final tree contains a path P_k for some $k \leq 12n$ that is a $\left(1+\epsilon_0\left(1+\frac{2nL}{h}\right)^k\right)$-approximation of P_{k-1}. The ratio of the length of P_k to that of P is at most:

$$\prod_{i=0}^{12n}\left(1+\epsilon_0\left(1+\frac{2nL}{h}\right)^i\right) \leq \left(1+\epsilon_0\left(1+\frac{2nL}{h}\right)^{12n}\right)^{12n}. \tag{3.13}$$

This error bound is not tight, but this is the best we could prove. We have found that a $(1+\epsilon)$-approximation algorithm based on this error bound is much slower in terms of n than the approximation algorithms we will discuss in Chapter 4, all of which use the Steiner point approach instead. It is not clear if the blow-up in the approximation factor of an *approximate sequence tree* can be inhibited to yield an efficient approximation scheme.

3.5 Polynomial time algorithms for special terrains

We now use Theorem 3.4 to give algorithms for two special classes of terrains: pseudo-convex terrains and pseudo-orthogonal terrains.

3.5.1 Algorithm for pseudo-convex terrains

A terrain is called *pseudo-convex* if every concave edge in the terrain (i.e., an edge subtending a dihedral angle of less than π radians "above" the terrain) is either vertical or horizontal. In other words, in a pseudo-convex terrain every edge that is neither vertical nor horizontal is convex. For example, a many-tiered wedding cake is pseudo-convex, even if each layer is dome-shaped, and even after slices are cut from it. Clearly every convex terrain is a pseudo-convex terrain, but not vice versa.

The LSDPs in a pseudo-convex terrain have a simple structure, which makes it easy to compute an LSDP through a given face sequence. To elaborate, let $\sigma = (f_0, f_1, \ldots, f_k)$ be a sequence of faces in a pseudo-convex terrain, and for all $i \in [0,k]$ $a_ib_i = f_{i-1} \cap f_i$ with $h(a_i) \geq h(b_i)$. Let $P = (p_0, p_1, p_2, \ldots, p_k, p_{k+1})$ be an LSDP through σ such that $p_0 \in f_0 - f_1$, $p_{k+1} \in f_k - f_{k-1}$, and $p_i \in a_ib_i$ for all $i \in [1,k]$. Let α_i and β_i be respectively the entering angle and the exiting angle of P at p_i. We show below that P consists of a sequence of constrained segments followed by a sequence of free segments. As in Section 3.3, we first unfold the faces in σ onto a common plane.

Lemma 3.23. *There exists an index $c \in [0, k+1]$ such that $(p_0, p_1, p_2, \ldots, p_c)$ is a constrained path and $(p_c, p_{c+1}, p_{c+2}, \ldots, p_{k+1})$ is a free path.*

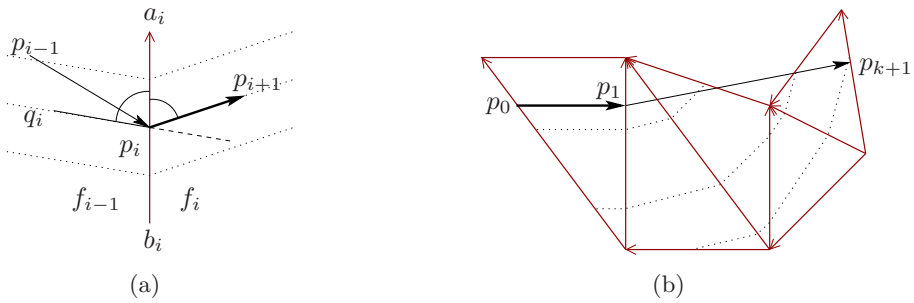


Figure 3.21: (a) Impossible subpaths of an LSDP in a pseudo-convex terrain. (b) A possible problem in tracing the LSDP.

Proof. It is sufficient to show that P does not have a free segment followed by a constrained segment. Suppose for contradiction that for some $i \in [1, k]$, $p_{i-1}p_i$ is a free segment and p_ip_{i+1} is a constrained segment (Figure 3.21(a)). Since a constrained segment does not intersect a level edge, a_ib_i is not a level edge. Let q_ip_i be a level line segment in f_{i-1} at p_i. Clearly, the level plane through p_i intersects $f_{i-1} \cup f_i$ along the path (q_i, p_i, p_{i+1}). When a_ib_i is a convex edge, $\angle a_ip_iq_i + \angle a_ip_ip_{i+1} \leq \pi$; otherwise a_ib_i is vertical and $\angle a_ip_iq_i + \angle a_ip_ip_{i+1} = \pi$. Thus $\angle a_ip_iq_i + \angle a_ip_ip_{i+1} \leq \pi$. As $p_{i-1}p_i$ is descending, p_{i-1} lies above q_ip_i, and thus P bends strictly upward at p_i. As a result, $\alpha_i < \beta_i$. This contradicts with Lemma 3.4. Therefore P does not have a free segment followed by a constrained segment, implying that P consists of a sequence of zero or more constrained segments followed by a sequence of zero or more free segments. □

Recall that it follows from Lemma 3.4 that a free path unfolds to a straight line segment. Because of this simple structure, we can find the LSDP P from p_0 to p_{k+1} through σ in $O(k)$ time as follows: trace a constrained path through σ starting at p_0, and at each node of this constrained path check if the straight line segment connecting the node to p_{k+1} is compatible with the path traced so far. By Lemma 3.8 only one bend will result in a compatible straight line segment, and this line segment together with the preceding constrained path is the required LSDP P, provided that P neither goes through a vertex nor leaves the faces in σ. It is possible that the second part of P (i.e., the straight line segment) is along a compatible direction at the last node of the constrained path, yet the line segment goes through a vertex or leaves the faces in σ, as shown in Figure 3.21(b). However, this is not an issue for our algorithm below. This is because our algorithm traces such a path while constructing a sequence tree as described in Section 3.4.1, where we have two bracketing LSDPs from p_0 through σ. The existence of these bracketing LSDPs rules out the possibility that the LSDP traced through σ goes through a vertex or leaves the faces in σ since the traced LSDP must lie in between the bracketing LSDPs.

Theorem 3.5. *Given a vertex s, we can construct in $O(n^3)$ time and $O(n^2)$ space a sequence tree that allows us to determine an SDP from s to any vertex v in $O(n)$ time.*

Proof. The proof follows immediately from Theorem 3.4, since $R(k) = O(k)$ in a pseudo-convex terrain. □

The above theorem applies immediately to a *pseudo-concave* terrain, which is defined similarly: a terrain is called pseudo-concave if every convex edge in the terrain is either vertical or horizontal.

3.5.2 Algorithm for pseudo-orthogonal terrains

A terrain is called *pseudo-orthogonal* if every face in the terrain is either vertical or horizontal. This is a generalization of an orthogonal terrain because a vertical face need not be parallel to the xz or yz plane. In other words, the class of pseudo-orthogonal terrains includes orthogonal terrains.

Every pseudo-orthogonal terrain is clearly a pseudo-convex terrain. Hence Theorem 3.5 applies directly to a pseudo-orthogonal terrain. But the LSDPs in a pseudo-orthogonal terrain are much simpler in structure than the LSDPs in a pseudo-convex terrain, and we can find an SDP in a pseudo-orthogonal terrain more easily. Although we have until now worked with triangulated terrains, for pseudo-orthogonal terrains it is better to discard the edges where the dihedral angle is flat. Then every edge of the terrain is either vertical or horizontal. Now let $P = (p_0, p_1, p_2, \ldots, p_k, p_{k+1})$ be an LSDP through face sequence σ, with σ, p_i, f_i, a_ib_i, α_i and β_i defined in the same manner as in Section 3.5.1.

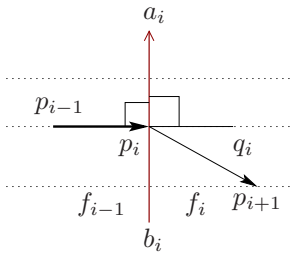


Figure 3.22: Impossible subpaths of an LSDP in a pseudo-orthogonal terrain.

Lemma 3.24. *Path P is either a constrained path or a free path.*

Proof. Because a pseudo-orthogonal terrain is a pseudo-convex terrain, Lemma 3.23 implies that P consists of a sequence of zero or more constrained segments followed by a sequence of zero or more free segments. It is therefore sufficient to show that P cannot have a constrained segment followed by a free segment. Suppose for contradiction that for some $i \in [1, k]$, $p_{i-1}p_i$ is a constrained segment and p_ip_{i+1} is a free segment (Figure 3.22). Since a constrained segment does not intersect a level edge, a_ib_i is a vertical edge. Let p_iq_i be a level line segment in f_i at p_i. Clearly both $\angle a_ip_ip_{i-1}$ and $\angle a_ip_iq_i$ are equal to $\frac{\pi}{2}$, and therefore, $p_{i-1}p_iq_i$ is a straight line segment. As p_ip_{i+1} is descending and free, p_{i+1} lies below p_iq_i, and thus P bends downward at p_i. As a result, $\alpha_i > \beta_i$, where α_i and β_i are respectively the entering angle and the exiting angle of P at p_i. This contradicts with Lemma 3.4. Therefore P is either a constrained path or a free path. □

When P is a constrained path, a_ib_i is a vertical edge for all $i \in [1, k]$ since a constrained segment does not intersect a level edge. Because both $p_{i-1}p_i$ and p_ip_{i+1} are level, $p_{i-1}p_i \perp a_ib_i$

and $p_ip_{i+1} \perp a_ib_i$, and thus $\alpha_i = \beta_i = 0$. On the other hand, when P is a free path, Lemma 3.4 implies $\alpha_i = \beta_i$ for all $i \in [1, k]$. In both cases, P unfolds to a straight line segment. As a result, we can find P in $O(1)$ time by simply connecting p_0 with p_{k+1} in the planar unfolding of the faces in σ. This argument is true even for the original (i.e., triangulated) terrain. This is because P does not bend at the additional edges of the triangulation, since each of these edges merely splits a face into two. Note that as in the case of pseudo-convex terrains, P in general may go through a vertex or leave the faces in σ, but this will not happen in our algorithm because of the existence of two bracketing LSDPs from p_0 through σ.

Theorem 3.6. *Given a vertex s, we can construct in $O(n^2)$ time and $O(n)$ space a sequence tree that allows us to determine in $O(n)$ time an SDP from s to any vertex v.*

Proof. The preprocessing time and the query time follow immediately from Theorem 3.4, since $R(k) = O(1)$ in a pseudo-orthogonal terrain. The $O(n)$ space requirement can be achieved by slightly modifying the algorithm in Theorem 3.4 as follows: during the construction of the sequence tree, we discard an edge-node if it has only one child, in the same manner as in Chen and Han [28]. This is possible because of the rectilinear nature of an LSDP in this terrain. The space requirement then follows directly from Theorem 8 of Chen and Han [28]. □

3.6 Conclusion

In this chapter we have given a full characterization of the bend angles of a shortest descending path (SDP), and have reduced the SDP problem to the problem of finding an SDP through a given sequence of faces. Our results imply that the difficulty with finding an SDP is not in deciding which face sequence to use, but in finding the SDP through a given face sequence. Our results also help us identify two classes of terrains for which the SDP problem is solvable in polynomial time. We have devised algorithms for these two classes of terrains.

One obvious open problem is the complexity of finding the SDP through a given sequence of faces from a given source point to a given target point. Using Theorem 3.2 we can trace an LSDP from a given source point if we are given the starting direction for the path, but we cannot solve the inverse problem of finding the direction from the source that will reach a given target point. The same issue was encountered by Mitchell and Papadimitriou while computing a shortest path in the weighted region problem [71, Section 8]. In brief, when they tried to calculate the positions of the nodes along the path that satisfies the local optimality criterion (i.e., Snell's law) at each node, they got k quartic equations with k unknowns, and solving for these unknowns using elimination yields a polynomial of degree doubly exponential in k. Our case is harder for the following reasons:

(i) As in the case of the weighted region problem, the equation for the local optimality criterion at each node of an SDP, when expressed in terms of the positions of the nodes along the path, contains square-roots. Because of the extra term $\lambda_{i,j}$ in the case of an SDP, simplifying the equation to "remove" the square-roots for the purpose of elimination yields an equation with unknowns of degree eight, as opposed to a quartic equation for the weighted region problem.

(ii) The local optimality criterion at a node of an SDP depends on the positions of many other nodes. More precisely, the directions of two segments of an SDP are related to each other even when the two segments are not adjacent, provided that all other segments in between them are constrained. In other words, for every pair of segments of the path we have an optimality criterion which is applicable only when all the nodes in-between are at a common height. Thus a local optimality criterion in our case can be *conditional*, which is not the case for the weighted region problem. It is not clear how to solve a set of conditional equations with high degree of unknowns.

(iii) A local optimality criterion in our case can be an equality or an inequality. The weighted region problem involves only equalities which are easier to solve.

Since finding the shortest path through a given face sequence in the weighted region problem remains an open problem 18 years after the numerical issues were first encountered, we believe that the problem of finding the SDP through a given face sequence may be intractable. So, a more interesting open question seems to be proving the NP-hardness of finding the SDP through a given face sequence.

We have shown in Section 3.5 two classes of terrains for which the SDP problem is easily solvable. Discovering other easily solvable classes of terrains is another interesting direction for further research.

Our error analysis in Section 3.4.3 seems to contradict a known experimental result [60] on the sequence tree approach for the Euclidean shortest path problem. To be precise, in Section 3.4.3 we have shown that in a sequence tree, trimming using approximate lengths decreases the accuracy to a large extent. The argument used there applies directly to unconstrained shortest paths in terrains, implying that the sequence tree approach should be vulnerable to floating point errors even for the Euclidean shortest path problem. This is because it is practically impossible to store exact Euclidean lengths of paths—representing the lengths with a floating point number introduces errors, as does rotating a face around an edge for unfolding. In fact, comparing the lengths of shortest paths between two points through different face sequences (which is done during one-angle one-split) requires exponentially many bits, because the algebraic numbers that describe the lengths may have exponential degrees [20, 21]. The first known implementation [102] of the sequence tree approach (for the Euclidean shortest path problem) had encountered floating point issues, but later on Kaneva and O'Rourke [60] were able to avoid any serious issue "within the range [they] were able to test" in their careful implementation. No error analysis is known to support this experimental result. We believe that the error analysis in Section 3.4.3 can be made more accurate, at least for the Euclidean shortest path problem. In particular, the "slack" in Inequality (3.13) can perhaps be made tighter, so can the upper bound used in that section on the number of times a path is affected by trimming. This demands further investigation.

Chapter 4

Approximation Algorithms for Shortest Descending Paths

4.1 Introduction

For many shortest path problems on terrains, algorithms to find a good approximation of a shortest path are more appealing than the ones to find exact paths. The main reason is that the terrain models used in these algorithms are often approximations of reality. An exact path in such a model is no better from a practical perspective than a good approximation of the path unless it is easy to compute an exact path. In practice, algorithms for exact shortest paths suffer from numerical issues in many cases. The weighted region problem is one such problem, and so is the shortest descending path (SDP) problem as we have discovered in Chapter 3. For both these problems, computation of an exact shortest path seems to involve complicated numerical issues even when we are given the sequence of faces traversed by the the path. Another issue is that any shortest path algorithm based on the continuous Dijkstra approach or the sequence tree approach involves unfolding a sequence of faces onto a common plane, and this operation may require an exponential number of bits to perform [2, 21]. On the other hand, approximation schemes based on the Steiner point approach are simple in design, hence easy to implement and attractive to practitioners [15].

Because of the difficulties in computing an exact SDP, it is logical to go for an approximate solution to the SDP problem. In this chapter we present two algorithms to find $(1 + \epsilon)$-approximation of SDPs in general terrains. Both of the algorithms are based on the Steiner point approach. More precisely, we model the SDP problem in a terrain as a shortest path problem in a weighted directed graph. The nodes in the graph correspond to Steiner points that are added along the edges of the terrain, with directed edges from higher to lower points in a common face. The weight of an edge corresponds to the Euclidean distance between the connected Steiner points. The algorithms are simple, robust and easy to implement. The two algorithms differ only in the placement of Steiner points—the first one places Steiner points evenly on the edges, whereas the second one places them in a geometric progression. The running times of these algorithms depend on the number n of vertices of the terrain, and the desired approximation factor ϵ. Moreover, because the geometry of the terrain determines the number of Steiner points required to achieve the desired approximation factor, the running

times depend also on the following geometric parameters: the length L of the longest edge, the smallest distance h of a vertex from a non-incident edge in the same face, and the largest acute angle θ between a non-level edge and a vertical line. The running times of our algorithms are as follows:

(i) Given a vertex s, the first algorithm places Steiner points evenly along terrain edges, so that after an $O\left(\frac{n^2 X}{\epsilon}\log\left(\frac{nX}{\epsilon}\right)\right)$-time preprocessing phase, we can determine a $(1+\epsilon)$-approximate SDP from s to any point v in $O(n)$ time if v is either a vertex of the terrain or a Steiner point, and in $O\left(\frac{nX}{\epsilon}\right)$ time otherwise, where $X = \frac{L}{h\cos\theta}$.

(ii) The second algorithm places Steiner points in a geometric progression along edges, and modifies the above preprocessing time and the two query times to $O\left(\frac{n^2 X'}{\epsilon}\log^2\left(\frac{nX'}{\epsilon}\right)\right)$, $O(n)$, and $O\left(\frac{nX'}{\epsilon}\log\left(\frac{nX'}{\epsilon}\right)\right)$, respectively, where $X' = \frac{L}{h}$.

The first algorithm is faster in terms of n, ϵ and $\frac{L}{h}$, but its running time depends on θ. On the other hand, the running time of the second algorithm does not depend at all on edge inclinations, and this algorithm is better for terrains with almost level edges. It is straightforward to follow a "hybrid" approach that first checks the edge inclinations of the input terrain, and then runs whichever of these two algorithms ensures a better running time for that particular terrain.[1]

The chapter is organized as follows. We discuss in Section 4.2 the uniqueness of our Steiner point approach, and mention why Steiner point algorithms for other terrain shortest path problems cannot solve the SDP problem. We give our two algorithms in Section 4.3 and Section 4.4. We conclude in Section 4.5 with a few open problems. See Section 3.2 for the terms used this chapter. The convention used here is similar to one in Chapter 3: "edge" means an edge of a (triangular) terrain face, "segment" means a line segment of a path, "vertex" means an endpoint of an edge, and "node" means an endpoint of a segment. "Node" and "link" mean the corresponding entities in a graph of Steiner points. We assume that all paths are directed. The conventions used in the figures are shown in Figure 3.1. The figures with face sequences show the faces after unfolding them onto a common plane.

4.2 Placing the Steiner points for SDPs

As we have mentioned before, our approximation algorithms work by first discretizing the terrain with many Steiner points along the edges, and then determining a shortest path in a directed graph in which each link connects a pair of vertices or Steiner points in a face of the terrain in the descending (more accurately, in the non-ascending) direction. Although the idea is similar to other Steiner point approaches discussed in Section 6.1.4, our approach is different from the previous Steiner point approaches in two ways, as discussed below.

[1]The first algorithm appeared in Ahmed and Lubiw [6], and a preliminary idea of the second appeared in Roy et al. [85]. A rigorous analysis of both the algorithms as well as their adaptation to a generalization of the SDP problem (to be discussed in Chapter 5) appears in our joint work [4] with Maheshwari, Roy and others.

4.2.1 Problems in placing Steiner points independently

While all the previous Steiner point approaches determine the positions of Steiner points in a face *independently from all other faces*, we cannot do the same for the SDP problem.

We have mentioned in Section 2.3 several Steiner point approximation algorithms for the weighted region problem [13, 14, 15, 92] and the shortest anisotropic path problem [31, 64, 89, 91]. All these algorithms assume that every face f is *totally traversable*, i.e., there is a feasible path from any point to any other point in f. To be precise, this assumption is without loss of generality for the weighted region problem because an infeasible path only occurs in a face with infinite weight, and such a face can simply be treated as an obstacle, i.e., not part of the terrain. For the shortest anisotropic path problem, the assumption does not hold in general, but all the papers on the problem make the assumption. Cheng et al. [31] assumed that the (anisotropic) weight associated with a direction of travel is bounded by constants from both above and below, thus any direction of travel is feasible. Moreover, the algorithm of Cheng et al. is for a subdivision of the plane, not for a terrain. The rest of the papers [64, 89, 91] used the anisotropic weight model of Rowe and Ross [81] which allows switchback paths to "cover" any direction in f. (Moreover, these three papers fail to approximate a path very close to a vertex, as we will see in Section 4.2.2.) The assumption that every face is totally traversable allows placing Steiner points in a face independently from all other faces. Sun and Reif [91, Section V] relaxed this assumption but only in isolated faces. Thus, they can still rely on independent placement of Steiner points in a face. If we relax the assumption in any two adjacent faces, their approach can make the number of Steiner points in those faces exponentially large.

For the SDP problem, ascending directions are unreachable in *every* face, which necessitates the use of a non-local strategy of placing Steiner points. In particular, we cannot place Steiner points in an edge without considering the heights of the Steiner points in other edges. To elaborate, for each Steiner point p in an edge, if there is no Steiner point with height $h(p)$ in other edges of the neighboring faces, it is possible that a descending path from s to v through Steiner points does not exist, or is arbitrarily longer than the SDP. For example, consider the SDP $P = (s, p_1, p_2, p_3, v)$ in Figure 4.1, where for each $i \in [1, 3]$, q_i, q_i' and q_i'' are three consecutive Steiner points with $h(q_i) > h(q_i') > h(q_i'')$ such that q_i is the nearest Steiner point above p_i. Note that in this figure the faces have been unfolded onto a plane, and that p_1 and q_1' are the same point. There is no descending path from s to v through the Steiner points: we must cross the first edge at q_1' or lower, then cross the second edge at q_2' or lower, and cross the third edge at q_3'' or lower, which puts us at a height below $h(v)$. Another important observation is that even if a descending path exists, it may not be a good approximation of P. In Figure 4.1, for example, if we want to reach a point v' slightly below v, $P' = (s, p_1, q_2', q_3'', v')$ would be a feasible path, but the last intermediate nodes of P and P' are not very close. We can easily extend this example to an SDP P going through many edges such that the "nearest" descending path P' gets further away from P at each step, and at one point, P' starts following a completely different sequence of edges. Clearly, we cannot ensure a good approximation by just making the Steiner points on an edge close to each other.

To guarantee the existence of a descending path through Steiner points that approximates an SDP from s to any vertex, we have to be able to go through the Steiner points in a sequence of faces without "losing height", i.e., along a level path. We achieve this by slicing the terrain with a set of horizontal planes, and then putting Steiner points where the planes intersect the

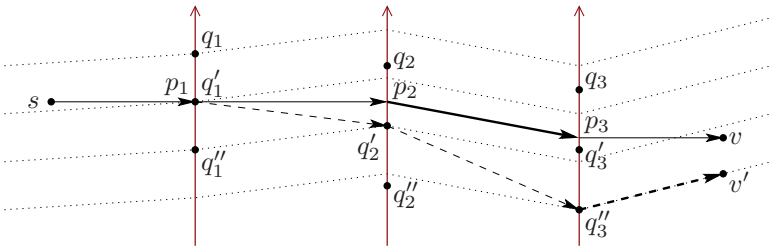


Figure 4.1: Problems with independently placed Steiner points.

edges. The set of horizontal planes includes one plane through each vertex of the terrain, and other planes in between them that are close enough to guarantee a good approximation ratio. This idea of horizontal slicing was introduced in our paper [6], which gave the algorithm we will discuss in Section 4.3.

4.2.2 Problems in placing Steiner points in geometric progression

There is another difference between our strategy of placing Steiner points and the strategies used in other papers. For the weighted region problem, the algorithms that place Steiner points in geometric progression run faster, in terms of n, than the algorithms that place Steiner points uniformly. For the SDP problem, we have the opposite situation. This is because SDPs very close to a vertex "behave" differently from shortest paths close to a vertex in the weighted region problem.

The previous approaches using Steiner points in geometric progression rely on the property that shortest paths in the weighted region problem cannot come very close to a particular vertex v more than once. As a result, approximating the part of a shortest path *near* to v with a path *through* v introduces an error which is much smaller than the length of the shortest path further away from v. This makes the approximation error analysis easy for the parts of the path near vertices. However, this property does not hold for shortest paths in the SDP problem. It is possible to construct a terrain where an SDP comes very close to a vertex v as many as $O(n)$ times, moving far away from v after every visit of the vicinity of v. Consider the terrain in Figure 4.2(a) which consists of the triangular faces of a pyramid with a star-shaped base. The points s and t have the same height, so the SDP P from s to t must consist of level segments. Moreover, P consists of $O(n)$ segments in the figure. Figure 4.2(b) shows the faces used by P after unfolding them onto a plane. By moving the convex vertices at the base away from the "center" of the base while keeping them on the same plane, we can make the points of P that are far away from v move even further away from v. Clearly it is possible to make P enter and leave a region close to v as many as $O(n)$ number of times. Because of such a possibility with an SDP, the approximation error analysis of our Steiner point approach is completely different from the previous approaches.

Note that since the SDP problem is a special case of the shortest anisotropic path problem, the above behavior of SDPs is also shown by shortest anisotropic paths. All the three papers on shortest anisotropic paths [64, 89, 91] that suggest the use of Steiner points in geometric progression fail to observe this issue.

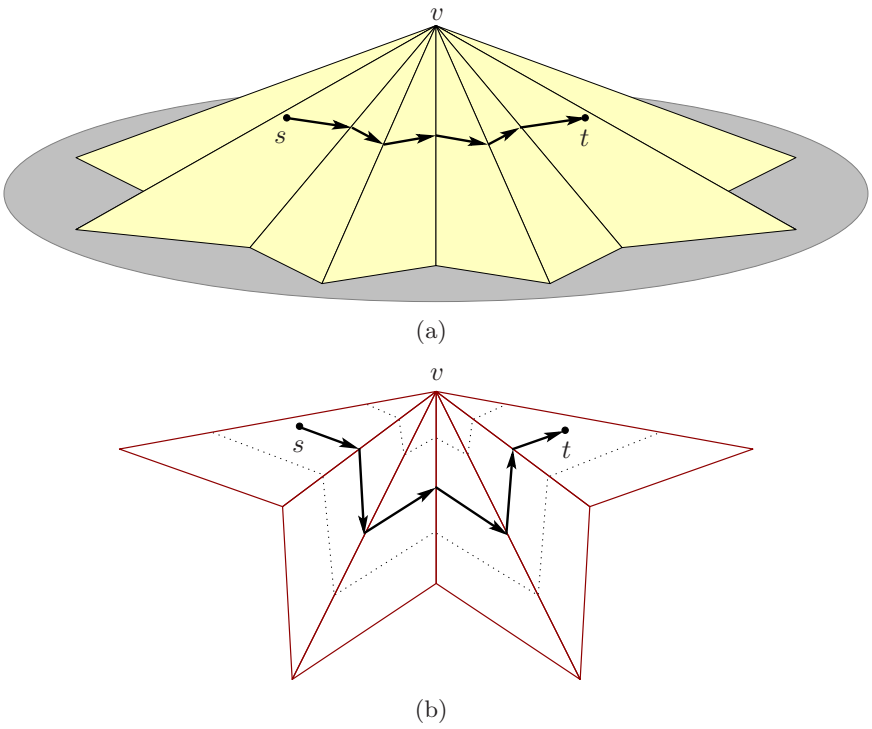


Figure 4.2: An SDP that comes close to a vertex $O(n)$ number of times.

4.3 Using uniform Steiner points

In our first algorithm the Steiner points on each edge are evenly spaced. To determine their positions, we first take a set of horizontal planes such that any two consecutive planes are within distance δ of each other, where δ is a small constant that depends on the approximation factor and terrain parameters. We then put a Steiner point at the intersection point of each of these planes with each of the terrain edges. One important observation is that this scheme makes the distance between consecutive Steiner points on an edge dependent on the slope of that edge. For instance, the distance between consecutive Steiner points is more for an almost-level edge than for an almost vertical edge. Since θ is the largest acute angle between a non-level edge and a vertical line, it can be shown that the distance between consecutive Steiner points on a non-level edge is at most $\delta \sec \theta$ (Lemma 4.3). Because of the situation depicted in Figure 4.1, we cannot place extra Steiner points *only* on the edges that are almost level. We guarantee a good approximation ratio by choosing δ appropriately. More precisely, we make sure that $\delta \sec \theta$ is small enough for the desired approximation ratio. Note that we can put Steiner points on a level edge without considering heights, since a level edge can never result in the situation depicted in Figure 4.1 (because all the points on such an edge have the same height).

4.3.1 Algorithm

Our algorithm runs in two phases. In the preprocessing phase, we place the Steiner points, and then construct a shortest path tree in the corresponding graph. During the query phase,

the shortest path tree gives an approximate SDP in a straightforward manner.

Preprocessing phase

Let $\delta = \frac{\epsilon h \cos\theta}{4n}$. We subdivide every non-level edge e of the terrain by placing Steiner points at the points where e intersects each of the following planes: $z = j\delta$ for all positive integers j, and $z = h(x)$ for all vertices x of the terrain. We subdivide every level edge e by putting enough Steiner points so that the length of each part of e is at most $\delta \sec\theta$. Let V be the set of all the vertices and Steiner points in the terrain. We then construct a weighted directed graph $G = (V, E)$ as follows, starting with $E = \emptyset$. For every pair (x, y) of points in V adjacent to a face f of the terrain, we add to E a directed link from x to y if and only if $h(x) \geq h(y)$ and xy is either an edge of the terrain or a segment through the interior of f. Note that we do *not* add a link between two points on the same edge unless both of them are vertices. Each link in E is assigned a weight equal to the length of the corresponding line segment in the terrain. Finally we construct a shortest path tree T rooted at s in G using the Bushwhack algorithm (discussed in page 15).

Note that we are mentioning set E only to make the discussion easy. In practice, we do not construct E explicitly because the neighbors of a node $x \in V$ in the graph are determined *during* the execution of the Bushwhack algorithm.

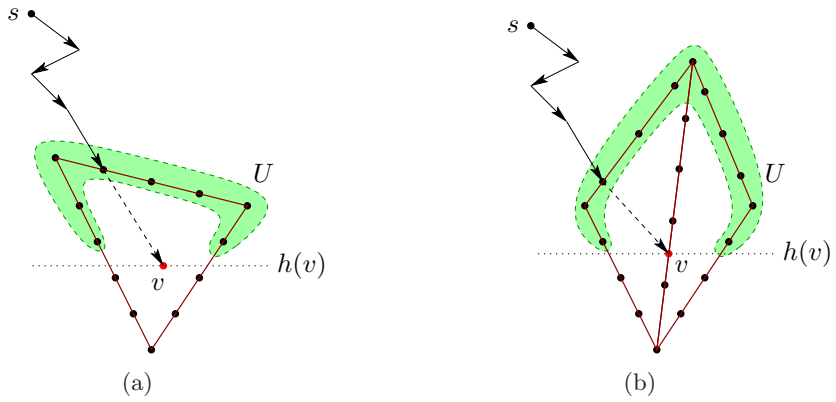


Figure 4.3: Finding an SDP from s to an interior point v of (a) a face and (b) an edge.

Query phase

When the query point v is a node of G, we return the path from s to v in T as an approximate SDP. Otherwise, we find the node u among those in V lying in the face(s) containing v such that $h(u) \geq h(v)$, and the sum of the length of the path from s to u in T and the length of the segment uv is minimum. We return the corresponding path from s to v as an approximate SDP in this case. To elaborate more on the latter case, let U be the set consisting of the nodes $u \in V$ with the following properties:

(i) u and v lie in a common face, and

(ii) $h(u) \geq h(v)$.

It is easy to see that if v is an interior point of a face, then all the nodes in U lie on at most three edges of that face (Figure 4.3(a)). Otherwise, v is an interior point of an edge, and there are at most four edges on which the nodes in U can lie (Figure 4.3(b)). Since we already know the length of an SDP from s to any $u \in U$, we can find in $|U|$ iterations the node $u \in U$ that minimizes the length of the path constructed by concatenating the segment uv at the end of the path from s to u in T. The corresponding path is returned as an approximate SDP.

4.3.2 Correctness and analysis

For the proof of correctness, we show that an SDP P from s to any point v in the terrain is approximated by a path P' from s to v in the *augmented graph* $G_v(V_v, E_v)$ constructed as follows. We first add v to graph G, and then add to this graph a link uv with weight $|uv|$ for every $u \in V$ such that u and v lie in a common face, and $h(u) \geq h(v)$. We construct path P' from P as follows. Note that P' might not be the path returned by our algorithm, but it provides an upper bound on the length of the returned path.

Let $P = (s = p_0, p_1, p_2, \ldots, p_k, v = p_{k+1})$ be an SDP from s to v such that p_i and p_{i+1} are two different boundary points of a common face for all $i \in [0, k-1]$, and p_k and p_{k+1} are two points of a common face. For ease of discussion, let e_i be an edge of the terrain through p_i for all $i \in [1, k]$ (e_i can be any edge through p_i if p_i is a vertex). Intuitively, we construct P' by moving each intermediate node of P upward to the nearest Steiner point. More precisely, we define a path $P' = (s = p'_0, p'_1, p'_2, \ldots, p'_k, v = p'_{k+1})$ as follows. For each $i \in [1, k]$, let $p'_i = p_i$ if p_i is a vertex of the terrain. Otherwise, let p'_i be the nearest point from p_i in $V \cap e_i$ such that $h(p'_i) \geq h(p_i)$. Such a point always exists in V because p_i is an interior point of e_i in this case, and it has two neighbors x and y in $V \cap e_i$ such that $h(x) \geq h(p_i) \geq h(y)$. Note that each node of P' except possibly the last one is either a vertex or a Steiner point.

Lemma 4.1. *For all $i \in [0, k]$, $h(p'_i) \geq h(p'_{i+1})$.*

Proof. We first claim that $h(p'_i) \geq h(p_{i+1})$. This claim follows from the facts that $h(p'_i) \geq h(p_i)$ by the definition of p'_i, and $h(p_i) \geq h(p_{i+1})$ as P is descending. Now consider the following two cases:

Case 1: $p'_{i+1} = p_{i+1}$ or e_{i+1} is a level edge. In this case, $h(p'_{i+1}) = h(p_{i+1})$. It follows from the inequality $h(p'_i) \geq h(p_{i+1})$ that $h(p'_i) \geq h(p'_{i+1})$.

Case 2: $p'_{i+1} \neq p_{i+1}$ and e_{i+1} is a non-level edge. In this case, there is either one or no point in e_{i+1} at any particular height. Let p''_{i+1} be the point in e_{i+1} such that $h(p''_{i+1}) = h(p'_i)$, or if no such point exists, let p''_{i+1} be the upper vertex of e_{i+1}. In the latter case, we can infer from the inequality $h(p'_i) \geq h(p_{i+1})$ that $h(p'_i) > h(p''_{i+1})$. Therefore we have $h(p'_i) \geq h(p''_{i+1})$ in both cases. Since $p''_{i+1} \in V \cap e_{i+1}$, the definition of p'_{i+1} implies that $h(p''_{i+1}) \geq h(p'_{i+1})$. So, $h(p'_i) \geq h(p'_{i+1})$.

Therefore, $h(p'_i) \geq h(p'_{i+1})$ for all $i \in [0, k]$. □

Lemma 4.2. *Path P' exists in G_v.*

Proof. It is sufficient to prove that $p'_i p'_{i+1} \in E_v$ for all $i \in [0, k-1]$, because by definition both p'_i and p'_{i+1} are in V_v, and $(p'_k, v) \in E_v$.

For all $i \in [0, k-1]$, p'_i and p'_{i+1} are boundary points of a common face by definition, and $h(p'_i) \geq h(p'_{i+1})$ by Lemma 4.1. It then follows from the construction that $p'_i p'_{i+1} \notin E$ only in the case that both of p'_i and p'_{i+1} lie on a common edge, and at most one of them is a vertex. We show as follows that this is impossible. When both p_i and p_{i+1} are vertices of the terrain, both p'_i and p'_{i+1} are vertices. When at least one of p_i and p_{i+1} is an interior point of an edge, they cannot lie on a common edge (Lemma 3.3); therefore, both of p'_i and p'_{i+1} cannot lie on a common edge unless both of p'_i and p'_{i+1} are vertices. So, it is impossible that both p'_i and p'_{i+1} lie on a common edge, and at most one of them is a vertex. Therefore, $p'_i p'_{i+1} \in E \subset E_v$. □

Lemma 4.3. *For all $i \in [1, k]$, $|p_i p'_i| \leq \frac{\epsilon h}{4n}$.*

Proof. It is sufficient to show that $|p_i p'_i| \leq \delta \sec\theta$ for all i, because $\delta \sec\theta = \frac{\epsilon h}{4n}$.

When $p_i = p'_i$, $|p_i p'_i| = 0 < \delta \sec\theta$. When $p_i \neq p'_i$, and e_i is a level edge, $|p_i p'_i| \leq \delta \sec\theta$ by construction. We will now focus on the case $p_i \neq p'_i$ and e_i is a non-level edge.

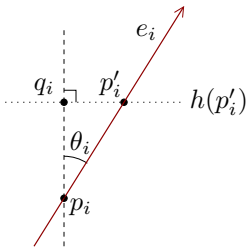


Figure 4.4: Bounding $|p_i p'_i|$ when $p_i \neq p'_i$ and e_i is a non-level edge.

Consider the vertical plane containing the edge e_i. Construct a line vertically upward from p_i to the point q_i where $h(q_i) = h(p'_i)$ (Figure 4.4). Let θ_i be the angle $\angle q_i p_i p'_i$. Since $h(q_i) = h(p'_i) > h(p_i)$, θ_i is an acute angle, and hence $\theta \geq \theta_i$, which implies: $\cos\theta \leq \cos\theta_i = \frac{|q_i p_i|}{|p_i p'_i|}$, and therefore, $|p_i p'_i| \leq |q_i p_i| \sec\theta$. As $q_i p_i$ is a vertical line segment, $|q_i p_i| = h(q_i) - h(p_i) = h(p'_i) - h(p_i) \leq \delta$ by construction, and therefore, $|p_i p'_i| \leq |q_i p_i| \sec\theta \leq \delta \sec\theta$. □

Lemma 4.4. *The algorithm returns a $(1+\epsilon)$-SDP.*

Proof. Let P and P' be respectively an SDP and a path in G_v as described above. Our algorithm finds a shortest path P'' in G_v, which provides a descending path of the same length. Since P' is a path in G_v (Lemma 4.2), the length of P'' is at most the length of P', and therefore it is sufficient to prove that the length of P' is at most $(1+\epsilon)$ times the length of P.

When $k = 0$, implying that P does not cross an edge of the terrain, we have $P = (s, v) = P'$, which proves the lemma trivially. When $k > 0$, the length of P' is equal to:

$$\begin{aligned}
\sum_{i=0}^{k} |p'_i p'_{i+1}| &\leq \sum_{i=0}^{k} \left(|p'_i p_i| + |p_i p_{i+1}| + |p_{i+1} p'_{i+1}| \right) \quad \text{(from triangle inequality)} \\
&= \sum_{i=0}^{k} |p_i p_{i+1}| + 2 \sum_{i=1}^{k} |p_i p'_i| \qquad \text{(since } p_0 = p'_0 \text{ and } p_{k+1} = p'_{k+1}\text{)} \\
&\leq \sum_{i=0}^{k} |p_i p_{i+1}| + \frac{\epsilon h k}{2n} \qquad \text{(Lemma 4.3)} \\
&< \sum_{i=0}^{k} |p_i p_{i+1}| + \epsilon h \ ,
\end{aligned}$$

because it follows from Lemma 3.3 that k is less than $2n$, the number of faces in the terrain. Now, since $k > 0$, p_1 lies on the edge opposite to p_0 in the face containing both p_0 and p_1, and therefore, $h \leq |p_0 p_1| \leq \sum_{i=0}^{k} |p_i p_{i+1}|$. So, $\sum_{i=0}^{k} |p'_i p'_{i+1}| < (1 + \epsilon) \sum_{i=0}^{k} |p_i p_{i+1}|$, and therefore, the length of P' is at most $(1 + \epsilon)$ times the length of P. □

Lemma 4.5. *Let $X = \left(\frac{L}{h}\right) \sec\theta$. Graph G has less than $\frac{15n^2X}{\epsilon}$ nodes and $O\left(\frac{n^3X^2}{\epsilon^2}\right)$ links. Moreover, it has less than $\frac{5nX}{\epsilon}$ nodes along any edge of the terrain.*

Proof. We will first prove the last part of the lemma. For each edge e of the terrain, the number of Steiner points corresponding to the planes $z = j\delta$ is at most $\frac{L}{\delta} - 1$, and the number of Steiner points corresponding to the planes $z = h(x)$ is at most $n - 2$. So,

$$\begin{aligned}
|V \cap e| &\leq \left(\frac{L}{\delta} - 1\right) + (n - 2) + 2 \\
&< \frac{L}{\delta} + n \\
&\leq 5n \left(\frac{L}{h}\right) \left(\frac{1}{\epsilon}\right) \sec\theta = \frac{5nX}{\epsilon}
\end{aligned}$$

because $\delta = \frac{\epsilon h \cos\theta}{4n}$ and $\left(\frac{L}{h}\right)\left(\frac{1}{\epsilon}\right)\sec\theta \geq 1$.

We will now compute $|V|$ and $|E|$. Let $c = \frac{5nX}{\epsilon}$ for ease of discussion. As the number of edges is at most $3n$, we have: $|V| < 3nc = \frac{15n^2X}{\epsilon}$.

For each face f of the terrain, there are less than $3c$ points in $V \cap f$, and each such point has less than $2c$ neighbors in f (more precisely, in the induced subgraph $G[V \cap f]$). So, the number of directed links in E contributed by f is less than $6c^2$, and this bound is tight for a level face. Because there are at most $2n$ faces, $|E| < 12nc^2 = O\left(\frac{n^3X^2}{\epsilon^2}\right)$. □

Theorem 4.1. *Let $X = \left(\frac{L}{h}\right) \sec\theta$, where L is the length of the longest edge, h is the smallest distance of a vertex from a non-incident edge in the same face, and θ is the largest acute angle between a non-level edge and a vertical line. Given a vertex s, and a constant $\epsilon \in (0, 1]$, we can discretize the terrain with $\frac{15n^2X}{\epsilon}$ Steiner points so that after a preprocessing phase that takes*

$O\left(\frac{n^2X}{\epsilon}\log\left(\frac{nX}{\epsilon}\right)\right)$ *time for a given vertex* s*, we can determine a* $(1+\epsilon)$*-approximate SDP from* s *to any point* v *in:*

(i) $O(n)$ *time if* v *is a vertex of the terrain or a Steiner point, and*

(ii) $O\left(\frac{nX}{\epsilon}\right)$ *time otherwise.*

Proof. The approximation factor follows from Lemma 4.4.

As we have mentioned before, we do not construct E explicitly because the neighbors of a node $x \in V$ in the graph are determined during the execution of the Bushwhack algorithm. As a result, the (implicit) construction of G takes $O(|V|)$ time. It follows from the running time of the Bushwhack algorithm that the preprocessing time of our algorithm is $O(|V|\log|V|) = O\left(\frac{n^2X}{\epsilon}\log\left(\frac{nX}{\epsilon}\right)\right)$ by Lemma 4.5.

During the query phase, if v is a vertex of the terrain or a Steiner point, the approximate path is in the tree T. Because the tree has height $O(n)$, it takes $O(n)$ time to trace the path. Otherwise, v is an interior point of a face or an edge of the terrain. The last intermediate node u on the path to v is a vertex or a Steiner point that lies on the boundary of a face containing v. If v is interior to a face [an edge], there are 3 [respectively 4] edges of the terrain on which u can lie. Thus there are $O\left(\frac{nX}{\epsilon}\right)$ choices for u by Lemma 4.5, and we try all of them to find the best approximate path, which takes $O\left(\frac{nX}{\epsilon}\right) + O(n) = O\left(\frac{nX}{\epsilon}\right)$ time. □

Corollary 4.1. *If the answer to a query is the length of an SDP (rather than the SDP itself), the query times for Cases (i) and (ii) of Theorem 4.1 become* $O(1)$ *and* $O\left(\frac{nX}{\epsilon}\right)$ *respectively.*

Note that the space requirement of our algorithm is $O(|V|) = O\left(\frac{n^2X}{\epsilon}\right)$ since we are not storing E explicitly. Also note that using Dijkstra's algorithm with a Fibonacci heap [47] instead of the Bushwhack algorithm yields an even simpler algorithm with a preprocessing time of $O(|V|\log|V| + |E|) = O\left(n^3\left(\frac{X}{\epsilon}\right)^2\right)$.

4.4 Using Steiner points in geometric progression

4.4.1 Algorithm

Unlike our first algorithm where the Steiner points on each edge are evenly spaced, our second algorithm places them non-uniformly along the edges. The Steiner points we use here are of two kinds. We first place Steiner points in "geometric progression" along the edges, as done by Aleksandrov et al. [13]. We call these points *primary Steiner points*. Then we place more Steiner points, called *isohypse Steiner points*, to guarantee that for every descending path in the terrain there exists a descending path through the Steiner points. Although the number of Steiner points used in this technique is more than in our first algorithm, the running time of the resulting algorithm no longer depends on the slope of the edges.

Preprocessing phase

The primary Steiner points are placed in such a way that for each vertex v of an edge e, there is a set of primary Steiner points whose distances from v form a geometric progression. Although the distance between a pair of consecutive Steiner points on e increases as we move away from v, we can still guarantee a good approximation ratio. This is because intuitively the length of a segment connecting two edges adjacent to v increases as we move the segment away from v—see Lemma 4.8 for a more precise statement. One observation is that if we want to maintain the geometric progression of the distances for the Steiner points very close to v, we would need infinitely many Steiner points near v. To avoid this problem, we do not put any primary Steiner points in a small region near v.

Before going into further details, we will define a few constants for ease of discussion. Let

$$\delta_1 = \frac{\epsilon h}{6n}$$

and

$$\delta_2 = \frac{\epsilon h}{6L} \ .$$

The constant δ_1 will define a region near v where we do not put any primary Steiner points, while δ_2 will determine the distances between consecutive primary Steiner points outside that region.

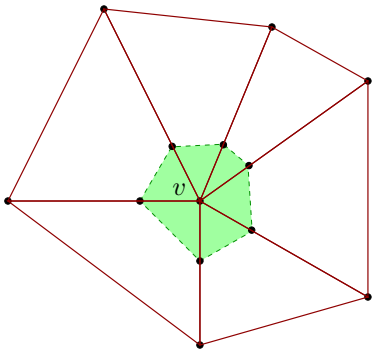


Figure 4.5: Vicinity of a vertex.

Definition 4.1 (Vicinity of a Vertex)**.** *In a face f incident to a vertex v, let p_1 and p_2 be two points lying on two different edges of f at v such that $|vp_1| = |vp_2| = \delta_1$. Clearly, $\triangle vp_1p_2$ is an isosceles triangle. The* vicinity of v *is defined to be the union of all such isosceles triangles around v (Figure 4.5).*

Note that the vicinities of any two vertices v_1 and v_2 are mutually disjoint because $\delta_1 < \frac{h}{2} < \frac{|v_1v_2|}{2}$.

In the preprocessing phase, we determine the positions of the Steiner points as follows. First, on every edge $e = v_1v_2$ we place primary Steiner points at points $p \in e$ such that $|pq| = \delta_1(1+\delta_2)^i$ for $q \in \{v_1, v_2\}$ and $i \in \{0, 1, 2, \ldots\}$. Then we add up to $3n$ isohypse Steiner points for each primary Steiner point and for each vertex, as follows. For every non-level edge

e, and every point p that is either a primary Steiner point or a vertex, we place an isohypse Steiner point at the point where e intersects the horizontal plane through p (i.e., the plane $z = h(p)$).

After placing the Steiner points, we construct a weighted directed graph $G = (V, E)$ and then construct a shortest path tree T rooted at s in G in the same way as in our first algorithm (Section 4.3.1).

Query phase

The queries are handled in exactly the same manner as in Section 4.3.1.

4.4.2 Correctness and analysis

For the proof of correctness, we follow the same approach used in Section 4.3.2: we show that an SDP P from s to any point v in the terrain is approximated by a path P' from s to v in the augmented graph $G_v(V_v, E_v)$. We construct path P' by moving each intermediate node of P upward to the nearest Steiner point. The correctness of our algorithm follows because the path returned by our algorithm is not longer than P'.

Let $P = (s = p_0, p_1, p_2, \ldots, p_k, v = p_{k+1})$ be an SDP from s to v such that p_i and p_{i+1} are two different boundary points of a common face for all $i \in [0, k-1]$, and p_k and p_{k+1} are two points of a common face. Let e_i be an edge of the terrain through p_i for all $i \in [1, k]$; e_i can be any edge through p_i if p_i is a vertex. Now define path $P' = (s = p'_0, p'_1, p'_2, \ldots, p'_k, v = p'_{k+1})$ as follows: for each $i \in [1, k]$, let $p'_i = p_i$ if p_i is a vertex of the terrain; otherwise, let p'_i be the nearest point from p_i in $V \cap e_i$ such that $h(p'_i) \geq h(p_i)$.

Lemma 4.6. *For all $i \in [0, k]$, $h(p'_i) \geq h(p'_{i+1})$.*

Proof. The proof is exactly the same as in Lemma 4.1. □

Lemma 4.7. *Path P' exists in G_v.*

Proof. The proof is exactly the same as in Lemma 4.2. □

Lemma 4.8. *For all $i \in [1, k]$ such that p_i is not inside a vertex vicinity, $|p_i p'_i| < \frac{\epsilon}{6} |p_{i-1} p_i|$.*

Proof. If p_i coincides with p'_i, the lemma follows trivially as $|p_i p'_i| = 0$. We will now focus on the case when these two points do not coincide. Since p_i is not inside a vertex vicinity, there is another Steiner point p''_i in e_i such that p'_i and p''_i lie on the opposite sides of p_i. Let v_i be the common vertex of e_{i-1} and e_i, w_i be the other vertex of e_i, and q_i and w'_i be two points in e_{i-1} such that $p_i q_i \perp e_{i-1}$ and $w_i w'_i \perp e_{i-1}$. Figure 4.6 depicts these vertices and points, for both the cases that the face angle at v_i is (a) acute and (b) obtuse.

We will first show that $|p'_i p''_i| \leq \delta_2 |v_i p_i|$. When $|v_i p''_i| < |v_i p'_i|$, we have $|v_i p'_i| \leq (1 + \delta_2) |v_i p''_i|$ by construction, and therefore, $|p'_i p''_i| = |v_i p'_i| - |v_i p''_i| \leq \delta_2 |v_i p''_i| \leq \delta_2 |v_i p_i|$. On the other

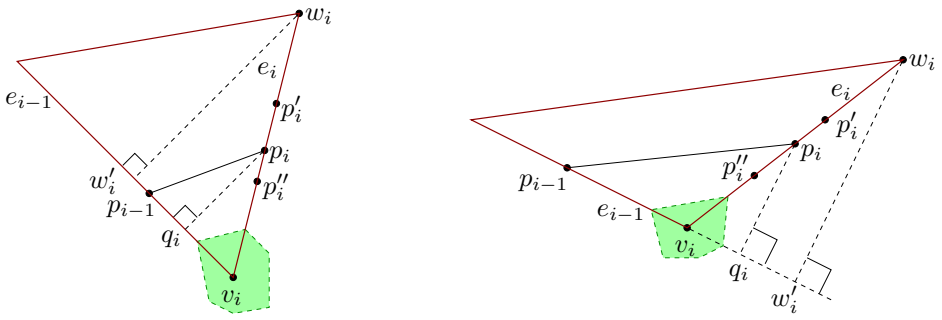


Figure 4.6: Bounding $|p_i p'_i|$ when the face angle at v_i is (a) acute and (b) obtuse.

hand, when $|v_i p''_i| > |v_i p'_i|$, then we have $|v_i p''_i| \leq (1+\delta_2)\,|v_i p'_i|$, which similarly implies that $|p'_i p''_i| = |v_i p''_i| - |v_i p'_i| \leq \delta_2\,|v_i p'_i| \leq \delta_2\,|v_i p_i|$.

Therefore, $|p_i p'_i| < |p'_i p''_i| \leq \delta_2\,|v_i p_i| = \delta_2\,|q_i p_i| \cdot \frac{|v_i p_i|}{|q_i p_i|} = \delta_2\,|q_i p_i| \cdot \frac{|v_i w_i|}{|w'_i w_i|}$. The lemma then follows from the inequalities $|q_i p_i| \leq |p_{i-1} p_i|$, $|v_i w_i| \leq L$, and $|w'_i w_i| \geq h$, as well as the definition of δ_2. □

Lemma 4.9. *For all $i \in [1, k]$ such that p_i is on or inside a vertex vicinity, $|p_i p'_i| \leq \frac{\epsilon h}{6n}$.*

Proof. If p_i is a vertex, the lemma follows trivially since $p'_i = p_i$ in this case. If p_i is not a vertex, let e_i be the edge containing p_i, and v_i be the vertex whose vicinity contains p_i. It is not hard to see that v_i is a vertex of e_i because δ_1 is strictly less than h. Let q_i be the primary Steiner point on e_i which lies at distance δ_1 from v_i. Clearly p_i lies in line segment $v_i q_i$. Now p'_i cannot be outside line segment $v_i q_i$ because otherwise we would have chosen either v_i or q_i as p'_i. As a result, p'_i also lies in line segment $v_i q_i$. Therefore, $|p_i p'_i| \leq |v_i q_i| = \delta_1 = \frac{\epsilon h}{6n}$. □

Lemma 4.10. *The algorithm returns a $(1+\epsilon)$-SDP.*

Proof. As in the proof of Lemma 4.4, it is sufficient to prove that the length of P' is at most $(1+\epsilon)$ times the length of P, where P and P' are respectively an SDP and a path in G_v as described above.

When $k = 0$, we have $P = (s, v) = P'$, which proves the lemma trivially. Otherwise, the length of P' is equal to:

$$\begin{aligned}
\sum_{i=0}^{k} |p'_i p'_{i+1}| &\leq \sum_{i=0}^{k} |p_i p_{i+1}| + 2\sum_{i=1}^{k} |p_i p'_i| && \text{(as in the proof of Lemma 4.4)} \\
&< \sum_{i=0}^{k} |p_i p_{i+1}| + 2\sum_{i=1}^{k} \left(\frac{\epsilon}{6}\,|p_{i-1} p_i| + \frac{\epsilon h}{6n}\right) && \text{(by Lemmas 4.8 and 4.9)} \\
&= \sum_{i=0}^{k} |p_i p_{i+1}| + \frac{\epsilon}{3}\sum_{i=1}^{k} |p_{i-1} p_i| + \frac{\epsilon h k}{3n}
\end{aligned}$$

$$\begin{aligned} &\leq \left(1+\frac{\epsilon}{3}\right)\sum_{i=0}^{k}|p_ip_{i+1}| \;+\; \frac{\epsilon hk}{3n} \\ &< \left(1+\frac{\epsilon}{3}\right)\sum_{i=0}^{k}|p_ip_{i+1}| \;+\; \frac{2\epsilon h}{3}\;, \end{aligned}$$

since $k < 2n$. Now, because $k > 0$, we have $h \leq |p_0p_1| \leq \sum_{i=0}^{k}|p_ip_{i+1}|$. So, $\sum_{i=0}^{k}|p'_ip'_{i+1}| < (1+\epsilon)\sum_{i=0}^{k}|p_ip_{i+1}|$, and therefore, the length of P' is at most $(1+\epsilon)$ times the length of P. □

Observation 4.1. *For any real number $x \in (0,1]$, $\log(1+x) > \frac{x\log e}{2}$.*

Proof. Since $\log(1+x)$ is a concave function, the definition of convcavity implies:

$$\log(1+x) = \log((1-x) \;+\; 2x) \geq (1-x)\log 1 + x\log 2 = x\;.$$

□

Lemma 4.11. *Graph G has less than $\frac{111n^2L}{\epsilon h}\log\left(\frac{6nL}{\epsilon h}\right)$ nodes and $O\left(\frac{n^3L^2}{\epsilon^2h^2}\log^2\left(\frac{nL}{\epsilon h}\right)\right)$ links. Moreover, it has less than $\frac{37nL}{\epsilon h}\log\left(\frac{6nL}{\epsilon h}\right)$ nodes along any edge of the terrain.*

Proof. We will first compute an upper bound on the number of primary Steiner points, which will then be used to prove the lemma.

Let n_e be the number of primary Steiner points on edge e. It is straightforward to see that n_e is at most $2j$, where j is the largest integer satisfying the inequality $\delta_1(1+\delta_2)^j < L$. Therefore,

$$\begin{aligned} n_e &\leq 2j \\ &< \frac{2\log\left(\frac{L}{\delta_1}\right)}{\log(1+\delta_2)} \\ &\geq \frac{2}{\delta_2}\log\left(\frac{L}{\delta_1}\right) \qquad \text{(by Observation 4.1)} \\ &= \frac{12L}{\epsilon h}\log\left(\frac{6nL}{\epsilon h}\right)\;. \end{aligned}$$

Since there are at most $3n$ edges in the terrain, the total number of primary Steiner points is at most $3nn_e$, which is less than $\frac{36nL}{\epsilon h}\log\left(\frac{6nL}{\epsilon h}\right)$.

We will now prove the last part of the lemma. For each point p that is either a primary Steiner point or a vertex, there is at most one node in $V \cap e$ for any edge e. This is obvious when p lies on e. On the other hand, if p does not lie on e, there is at most one isohypse Steiner point on e that corresponds to p. Using the above bound on the number of primary Steiner points, we have:

$$|V \cap e| \;<\; \frac{36nL}{\epsilon h}\log\left(\frac{6nL}{\epsilon h}\right) \;+\; n \;<\; \frac{37nL}{\epsilon h}\log\left(\frac{6nL}{\epsilon h}\right)\;.$$

We will now compute $|V|$ and $|E|$. Let $c = \frac{37nL}{\epsilon h} \log\left(\frac{6nL}{\epsilon h}\right)$ for ease of discussion. Because the number of edges is at most $3n$, we have: $|V| < 3nc = \frac{111n^2L}{\epsilon h} \log\left(\frac{6nL}{\epsilon h}\right)$. Using the same argument we used in the proof of Lemma 4.5, we can say that the number of directed links in E contributed by each face of the terrain is less than $6c^2$. Because there are at most $2n$ faces, $|E| < 12nc^2 = O\left(\frac{n^3L^2}{\epsilon^2h^2} \log^2\left(\frac{nL}{\epsilon h}\right)\right)$. □

Theorem 4.2. *Let $X' = \frac{L}{h}$, where L is the length of the longest edge, and h is the smallest distance of a vertex from a non-incident edge in the same face. Given a vertex s, and a constant $\epsilon \in (0, 1]$, we can discretize the terrain with at most $\frac{150n^2X'}{\epsilon} \log\left(\frac{6nX'}{\epsilon}\right)$ Steiner points so that after a preprocessing phase that takes $O\left(\frac{n^2X'}{\epsilon} \log^2\left(\frac{nX'}{\epsilon}\right)\right)$ time for a given vertex s, we can determine a $(1+\epsilon)$-approximate SDP from s to any point v in:*

(i) $O(n)$ time if v is a vertex of the terrain or a Steiner point, and

(ii) $O\left(\frac{nX'}{\epsilon} \log\left(\frac{nX'}{\epsilon}\right)\right)$ time otherwise.

Proof. The proof is the same as in Theorem 4.1 except that we use Lemmas 4.10 and 4.11 instead of Lemmas 4.4 and 4.5, respectively. □

Corollary 4.2. *If the answer to a query is the length of an SDP, the query times for Cases (i) and (ii) of Theorem 4.2 become $O(1)$ and $O\left(\frac{nX'}{\epsilon} \log\left(\frac{nX'}{\epsilon}\right)\right)$ respectively.*

As in the case of our first algorithm, we can use Dijkstra's algorithm with a Fibonacci heap [47] instead of the Bushwhack algorithm to have an even simpler algorithm with a preprocessing time of $O\left(\frac{n^3X'^2}{\epsilon^2} \log^2\left(\frac{nX'}{\epsilon}\right)\right)$.

4.5 Conclusion

In this chapter we have presented two robust and easy to implement approximation algorithms for shortest descending paths (SDPs) in general terrains. The algorithms compute $(1+\epsilon)$-approximate SDPs in $O\left(\frac{n^2X}{\epsilon} \log\left(\frac{nX}{\epsilon}\right)\right)$ and $O\left(\frac{n^2X'}{\epsilon} \log^2\left(\frac{nX'}{\epsilon}\right)\right)$ time respectively, where $X = \frac{L}{h\cos\theta}$ and $X' = \frac{L}{h}$. The running times are not comparable: the first algorithm is faster in terms of n, ϵ and $\frac{L}{h}$, but its running time depends on θ which is not the case for the second algorithm. Another "hidden" point in this comparison is that the constant factor in the number of Steiner points used in the first algorithm is much smaller than the same factor for the second algorithm, see Lemmas 4.5 and 4.11. The "hybrid" approach we mentioned in the beginning of this chapter should consider all these issues when deciding which of the algorithms is better for a given terrain.

Our algorithms place Steiner points without using any structural information about SDPs, like the characterization of the bend angle we discussed in Chapter 3. The same is true for the Steiner point based algorithms for many other shortest path problems in terrains, e.g., the weighted region problem and the shortest anisotropic path problem—we have discussed them in

Sections 2.3.1 and 2.3.2. One direction for further research lies in using structural information about shortest paths to reduce the number of Steiner points. We think that the continuous Dijkstra approach or the sequence tree approach should be helpful, particularly for the case of single source shortest paths where we are interested in paths from a given source s to many other points in the terrain. In this case, it may be possible to guarantee a good approximation ratio by making the Steiner points dense in the faces near s and sparse in the faces far away from s. A similar approach was used by Cheng et al. [29] for the shortest anisotropic path problem, but their approach works only for a subdivision of the plane (i.e., not for a terrain) where it is relatively easy to *estimate* the distance of a point from s and determine the density of Steiner points accordingly. We think that for terrains, the continuous Dijkstra approach or the sequence tree approach can be used to "flatten" the terrain for the purpose of finding distance estimates. Since the characterization of bend angles are known for shortest paths in both the SDP problem and the weighted region problem, the idea may be helpful for both problems. Note that a similar characterization is not known for paths in the shortest anisotropic path problem.

The fastest algorithm for the weighted region problem is the one by Aleksandrov et al. [15], who managed to use fewer Steiner points than all previous algorithms for the problem. Their trick is to place Steiner points along the bisectors of the face angles rather than along the edges. It may appear that a faster SDP algorithm is possible using the same technique. The technique, however, cannot be used for our problem very easily. The main problem is that it is not clear how to guarantee the existence of a *feasible* path that approximates an SDP. More precisely, even after isohypse Steiner points are added as in our second algorithm, there may be no descending path through the Steiner points (along the bisectors) that lies very close to an SDP.

When query point v is neither a vertex of the terrain nor a Steiner point, the query phase can be made faster by using a point location data structure on the underlying weighted Voronoi diagram defined by the Steiner points on the boundary of each face. Note that the Voronoi diagram consists of hyperbolic arcs.

Moet et al. [73] showed that the visibility map, shortest path map and Voronoi diagram can be computed more efficiently in a terrain in which certain geometric parameters have bounded values. They call such terrains *realistic terrains*, although they mention that their assumptions on the bounds on those parameters are *not* based on the terrains encountered in practice. It is interesting to know if practical terrains follow their assumptions. If this is the case, then two of our algorithms, more precisely those using non-uniform Steiner points, become independent of geometry in such terrains because realistic terrains satisfy $\frac{L}{h} = O(1)$.

Chapter 5

Shortest Gently Descending Paths

5.1 Introduction

In this chapter we introduce a generalization of the shortest descending path (SDP) problem, called the *shortest gently descending path (SGDP)* problem, where a path descends, but *not too steeply*. The additional constraint to disallow a very steep descent makes the paths more realistic in practice. For example, when we ski down a mountain we avoid a too steep descent. In such cases, a very steep segment of a descending path should be replaced by "switchbacks" that go back and forth at a gentler slope, like the hairpin bends on a mountain road (see Figure 5.1). The SGDP problem combines the SDP problem with the more general problem of finding shortest anisotropic paths (discussed in Section 2.3.2), where there is a direction-dependent weight function for every triangular face of the terrain, and the goal is to minimize the weighted length of a path.

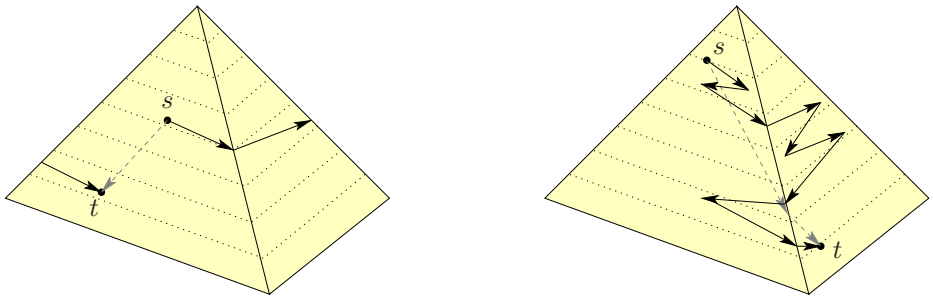


Figure 5.1: Descending gently towards a steep direction.

Although the SGDP problem is a special case of the shortest anisotropic path problem, we cannot solve the SGDP problem using an existing algorithm for shortest anisotropic paths. This is because results on shortest anisotropic paths [31, 64, 89, 91] assume that there is a feasible path between any two points in a common face, a property that is violated when ascending paths are forbidden. We have discussed the issue in detail for the SDP problem in Section 4.2, and the discussion applies to the SGDP problem. In this chapter we combine techniques used for SDPs and for shortest anisotropic paths and present two $(1+\epsilon)$-approximation algorithms

to solve the SGDP problem on general terrains. Both algorithms are based on the Steiner point approach, and they are simple, robust and easy to implement. The algorithms are very similar to the ones described in Chapter 4. The main difference is that in our algorithms for SGDPs, an edge in the graph of Steiner points represents a sequence of gently descending segments that can lie in different faces. This is unlike our SDP algorithms where an edge in the graph represents a *single* segment of a descending path. The running times of our SGDP algorithms depend on the number n of vertices of the terrain, the largest degree d of a vertex, and the desired approximation factor ϵ. Moreover, as in Chapter 4, the geometry of the terrain determines the number of Steiner points required to achieve the desired approximation factor, thus the running times depend on four geometric parameters: the length L of the longest edge, the smallest distance h of a vertex from a non-incident edge in the same face, the largest acute angle θ between a non-level edge and a vertical line, and the angle of steepness ψ (see Section 5.2 for details). The running times are as follows:

(i) Given a vertex s, the first algorithm places Steiner points uniformly along terrain edges during an $O\left(\frac{n^2X}{\epsilon}\log\left(\frac{nX}{\epsilon}\right)\right)$ -time preprocessing so that we can determine a $(1+\epsilon)$-approximate SGDP from s to any point v in $O(nd)$ time if v is either a vertex of the terrain or a Steiner point, and in $O\left(n\left(d+\frac{X}{\epsilon}\right)\right)$ time otherwise, where $X = \frac{L}{h\cos\theta\,\cos\psi}$.

(ii) The second algorithm places Steiner points in a geometric progression along edges, and modifies the above preprocessing time and the two query times to $O\left(\frac{n^2X'}{\epsilon}\log^2\left(\frac{nX'}{\epsilon}\right)\right)$, $O(nd)$, and $O\left(nd+\frac{nX'}{\epsilon}\log\left(\frac{nX'}{\epsilon}\right)\right)$, respectively, where $X' = \frac{L}{h\cos\psi}$.

The first algorithm is faster in terms of n, ϵ, $\frac{L}{h}$ and ψ. But the running time of the second algorithm is independent of θ, making it faster for terrains with almost level edges. Like the algorithms in Chapter 4, we can combine these two algorithms into a "hybrid" one that first checks the edge inclinations of the input terrain, and then runs whichever of these two algorithms is faster for the terrain.[1]

An SGDP can have many bends in the interior of a face—Figure 5.1(b) shows such an example. Moreover, there can be infinitely many SGDPs even between two points in a common face. In practice, however, not all of these SGDPs are equally good. For example, for robot motion planning, an SGDP with fewer bends is preferable because a robot needs extra time and energy to change its direction at a bend. It is therefore natural to look for an SGDP with a limited number of bends. Furthermore, when the energy needed to turn a robot at a bend is proportional to the turn-angle at that bend, one may be interested in finding an SGDP with a limited total turn-angle. We show in this chapter that both these problems are intractable. More precisely, we prove that it is NP-hard to decide if there exists a gently descending path of length at most $\mathcal{L}$ that has (i) at most k bends, or (ii) a total turn-angle at most ϕ.[2]

Our hardness results are on *bicriteria* shortest path problems, where the goal is to minimize two objective functions of a path (one of them is the path length). See Mitchell [69, Section 4] for a survey of bicriteria shortest path problems, and Daescu et al. [35] for more recent approximation algorithms. Arkin et al. [16] give hardness results on three bicriteria shortest path

[1] Our SGDP algorithms first appeared in Ahmed, Lubiw and Maheshwari [11]. These algorithms together with the ones in Chapter 4 appeared later on in Ahmed et al. [4].

[2] A preliminary version of our hardness result appeared in Ahmed and Lubiw [8].

problems: (i) in a polygon with holes, finding a path whose length and total turn-angle are below given limits; (ii) in a polygon with holes, finding a path whose lengths in two different L_i norms are below given limits; and (iii) in a polygon with red and blue faces, finding a path whose total length in red [and blue] faces is below a given limit.

Our hardness results are significant in two ways. First, the (single criteria) shortest path problem corresponding to our bicriteria problems is of unknown complexity. The SGDP problem and other well-studied special cases of the shortest anisotropic path problem, like the weighted region problem and the SDP problem, share the interesting feature that although approximation algorithms are the best results known, hardness proofs seem elusive. Even some bicriteria shortest path problems have this feature. For example, the complexity of finding a shortest k-link path in a polygon with holes is still open [69, Open Problem 11].

Secondly, our results imply that finding a shortest anisotropic path with a limited number of bends or a limited total turn-angle is hard. These results apply even for the anisotropic cost model of Rowe and Ross (discussed in Section 2.3.2), a well-investigated model with many practical applications. The only previously known hardness result on the shortest anisotropic path problem is the hardness with a limited number of bends. This result follows from the first result of Arkin et al. mentioned above, when the direction-dependent weight function of a face is either a unit function (returning one for all directions) or an infinity function (returning infinity for all directions). This previously known hardness result does *not* apply for the cost model of Rowe and Ross because in this model the direction-dependent weight function in a face can be a constant function iff the face is level, thus we cannot construct a terrain that "represents" holes surrounded by polygon faces in the problem of Arkin et al.

The chapter is organized as follows. We define relevant terms in Section 5.2, and establish a few properties of an SGDP in Section 5.3. We give our two algorithms in Section 5.4 and Section 5.5. In Section 5.6 we describe the details of our construction used for our hardness results, and prove the hardness of SGDPs with few bends. We give a few additional lemmas in Section 5.7 to show the hardness of and SGDPs with limited total turn-angle. We conclude in Section 5.8 with a few open problems.

5.2 Terminology

A path P from s to t on the terrain is *descending* if the z-coordinate of a point p never increases while we move p along the path from s to t. Given an angle $\psi \in [0, \frac{\pi}{2})$, a line segment pq in 3D is *steep* if it makes an angle less than ψ with a vertical line. A steep line segment pq lies strictly inside the cone defined by the lines at an angle ψ with a vertical line at p, as shown in Figure 5.2. A path P is *gently descending* if P is descending, and no segment of P is steep. A downward direction in a face is called a *critical direction* if the direction makes an angle equal to ψ with a vertical line. (Note that only a *downward* direction can be a critical direction, although both upward and downward directions can be steep.) A gently descending path is called a *critical path* if each of its segments is in a critical direction. A critical path may travel through more than one face; inside a face it will zigzag back and forth. We would like to replace steep descending segments by critical paths. This is sometimes possible, e.g., for a steep segment starting and ending at points interior to a face, but is not possible in general. The details are in Lemma 5.3, which uses the following terms. A vertex v in face f is *locally*

sharp in f if v is either the higher endpoint of two steep edges or the lower endpoint of two steep edges of f. A vertex v is *sharp* if it is locally sharp in all its incident faces. Note that a sharp vertex is either the higher endpoint of all the edges incident to it, or the lower endpoint of all such edges. A sharp vertex of the first type is like a pinnacle from which one cannot descend gently.

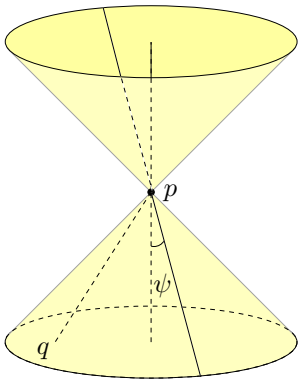


Figure 5.2: The cone defined by the lines at an angle ψ with a vertical line at p, and a steep line segment pq.

A *bend* on a path P in the 3D terrain (i.e., in the *original* terrain, without unfolding the faces containing P onto a common plane) is a point where P changes its direction. To be precise, P bends at a node b if the two segments ab and bc of P at b are not collinear in the (3D) terrain. We define the *turn-angle of P at b* as the angle between the vectors $\overrightarrow{ab}$ and $\overrightarrow{bc}$. (Note that the turn angle is zero if P goes straight at b.) The *total turn-angle of P* is the summation of the turn-angles of P at all intermediate nodes. Note that segments ab and bc in these definitions can lie either in a common face or in two adjacent faces, and in the latter case it is possible that ab and bc become collinear *after* the faces are unfolded onto a common plane. This suggests that bends and turn-angles can alternatively be defined in an unfolding of the faces. The latter definitions may be more acceptable for certain applications of the SGDP problem, even though they are not clear when b is a vertex. Our reduction works for any of these alternate definitions, and we will use the original ones.

The terms relevant to the 3-SAT problem [48] used in our hardness results are as follows. A *literal* in a Boolean formula is an occurrence of a variable b or its negation $\bar{b}$. A *clause* is the logical disjunction (OR) of one or more literals. For our reduction in Sections 5.6 and 5.7, we are given an instance f of 3-SAT, which is a Boolean formula over n variables b_0, b_1, b_2, $\ldots$, b_{n-1} that is expressed as a logical conjunction (AND) of m clauses C_0, C_1, C_2, $\ldots$, C_{m-1}. More precisely, $f = C_0 \wedge C_1 \wedge C_2 \wedge \ldots C_{m-1}$, where for all $i \in [0, m-1]$, $C_i = (l_{i,1} \vee l_{i,2} \vee l_{i,3})$ using $l_{i,j}$ to denote a literal.

As in previous chapters, "edge" means an edge of a (triangular) terrain face, "segment" means a line segment of a path, "vertex" means an endpoint of an edge, and "node" means an endpoint of a segment. "Node" and "link" mean the corresponding entities in a graph of Steiner points. We assume that all paths are directed. The figures throughout this chapter follow the same conventions shown in Figure 3.1. The figures in our hardness construction in Section 5.6 use the two additional "symbols": (i) when a path segment (which is denoted by an arrow with a solid, dark arrowhead) is blocked by a Blocker gadget (defined in Section 5.6.2),

we mark the segment with a dashed arrow; and (ii) in some of the figures, we draw grids using light-colored dotted lines for measuring purpose. The figures with face sequences show the faces after unfolding them onto a common plane.

5.3 Properties of SGDPs

Observe that as in the case of shortest descending paths (Lemma 3.1), the following holds for shortest gently descending paths.

Lemma 5.1. *Any subpath of an SGDP is an SGDP.*

Proof. The proof is trivial, and similar to the proof of Lemma 3.1. □

Recall the definition of a critical path in Section 5.2. We show that any critical path in the terrain is an SGDP.

Lemma 5.2. *Any critical path from a point a to a point b in the terrain is an SGDP of length $(h(a) - h(b)) \sec \psi$.*

Proof. Any critical path P is a gently descending path. Since each segment of P makes an angle ψ with a vertical line, the length of P is $\frac{h(a)-h(b)}{\cos \psi} = (h(a) - h(b)) \sec \psi$. Ignoring the terrain, *any* gently descending path from a to any point at height $h(b)$ has length at least $(h(a) - h(b)) \sec \psi$. So, P is an SGDP. □

We will introduce a notation to simplify our expressions involving the length of an SGDP. For any two points p and q in a common face f, let $\|pq\| = |(h(p) - h(q))| \sec \psi$ when line pq is steep, and $\|pq\| = |pq|$ otherwise. Thus $\|pq\|$ is the length of an SGDP from p to q if one exists.

Observation 5.1. *For any three points p, q and r in a face f:*

(i) $\|pq\| = \|qp\|$,

(ii) $\|pr\| \leq \|pq\| + \|qr\|$, *and*

(iii) $|pq| \leq \|pq\| \leq |pq| \sec \psi$.

Proof. Cases (i) and (ii) are trivial. When pq is not a steep segment, Case (iii) follows immediately since $\|pq\| = |pq|$ in this case, and $\sec \psi \geq 1$. When pq is steep, let ψ' be the angle line pq makes with a vertical line. Clearly $0 \leq \psi' < \psi$ and therefore $\sec \psi' < \sec \psi$. So, $|pq| = |(h(p) - h(q))| \sec \psi' < |(h(p) - h(q))| \sec \psi = \|pq\|$. Moreover, since $|pq| \geq |h(p) - h(q)|$, we have $\|pq\| = |(h(p) - h(q))| \sec \psi \leq |pq| \sec \psi$. □

Like other Steiner point approaches, our graph of Steiner points has a directed link (of appropriate cost) between two Steiner points a and b in a common face f such that the link represents an SGDP from a to b. However, unlike other well-studied shortest paths in terrains where the shortest path represented by the link lies completely in f, the SGDP in our case may go through many other faces. For example, the critical path in Figure 5.1(a) is an SGDP by Lemma 5.2, and it goes through four faces even though both s and t lie on a common face. It is not straightforward to determine if such an SGDP exists at all—we need this information during the construction of the graph. Moreover, in case an SGDP exists, we want to know the number of faces used by the path because our algorithm has to return the path in the terrain that corresponds to the shortest path in the graph. Lemma 5.3 below handles these issues:

Lemma 5.3. *Let a and b be two points in a face f with $h(a) \geq h(b)$.*

(i) If at least one of a and b is a sharp vertex, no gently descending path exists from a to b.

(ii) If neither a nor b is a locally sharp vertex in f, there exists an SGDP from a to b lying completely in f. Moreover, the SGDP is a critical path if ab is steep.

(iii) Otherwise, there exists an SGDP from a to b that uses at most $d+1$ faces, and is a critical path.

Proof. (i) If a [or b] is a sharp vertex, the segment ap [respectively pb] is steep for any point p in any face incident to a [respectively b]. So, no gently descending path exists from a to b.

(ii) If ab is not a steep segment, this is the SGDP. Otherwise, there are many critical paths in f from a to b, as shown Figure 5.3(a). To be precise, we trace one such path as follows. Let $apbq$ be the parallelogram defined by the critical directions at a and b in the plane containing f (Figure 5.3). We follow a critical direction in f from a until we intersect either pb or qb or an edge of f. Let a_1 be the intersection point. Clearly, a_1 is not a locally sharp vertex in f, and therefore, if a_1 is on neither pb nor qb, we can again follow a critical direction from a_1 until we intersect either pb or qb or an edge of f. Let a_2 be this intersection point. We repeat this step until we reach a point a_k on either pb or qb. Thus we get the critical path $(a, a_1, a_2, \ldots, a_k, b)$, which is an SGDP by Lemma 5.2. This completes the proof provided that this critical path always reaches either pb or qb.

When at most one edge of f intersects the interior of the parallelogram, our critical path reaches either p or q by following an edge of the parallelogram at a (Figure 5.3(b)). When two edges e_1 and e_2 of f intersect the interior of the parallelogram (Figure 5.3(c)), the point $e_1 \cap e_2$ is neither a nor b nor an interior point of f. This is because f is a triangle, and at most one edge of f at a [and b] intersects the interior of the parallelogram (since neither a nor b is locally sharp in f). So, there is a "gap" between e_1 and e_2 in the parallelogram, which gives us a positive constant c such that $|a_i a_{i+1}| > c$ for all $i \in [1, k-2]$. We therefore reach a_k in a finite number of steps (in at most $\frac{\|ab\|}{c} + 2$ steps to be precise).

(iii) In this case, at least one of a and b is a locally sharp vertex in f, and therefore, ab is a steep line segment. If a is not a locally sharp vertex in f, let $a' = a$. Otherwise, let a' be an interior point of line segment ab such that no vertex of the terrain lies strictly in

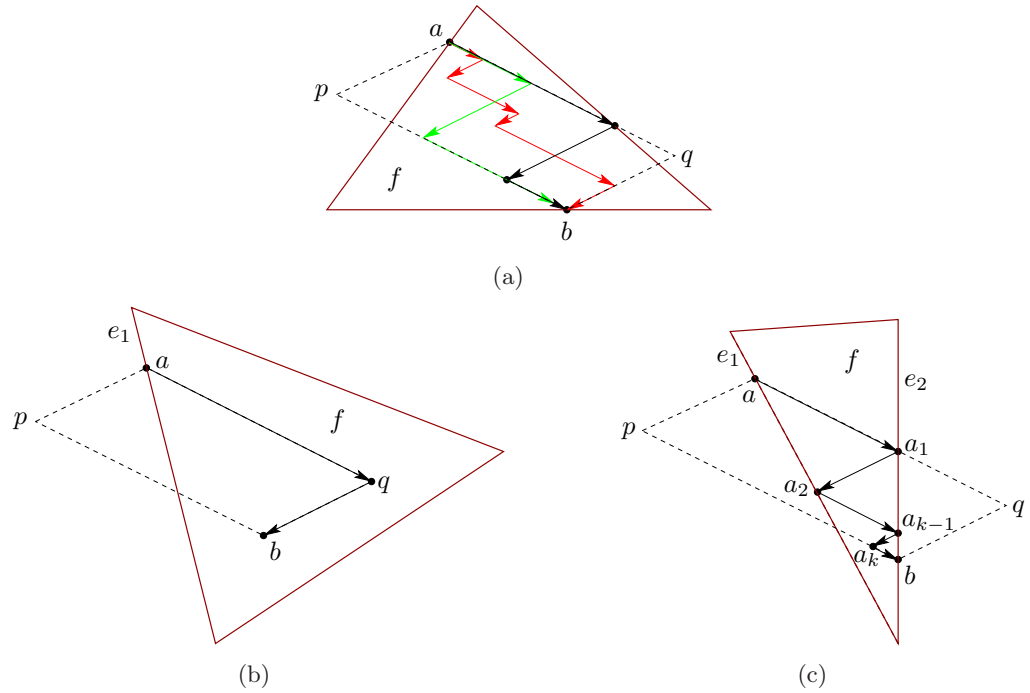


Figure 5.3: (a) Three of the infinitely many critical paths from a to b when ab is a steep segment and neither a nor b is a locally sharp vertex in f. A critical path when parallelogram $apbq$ contains (b) one and (c) two edges of f.

between the planes $z = h(a)$ and $z = h(a')$. We similarly define a point b', and make sure that $h(a') > h(b')$. Clearly $h(a) > h(a') > h(b') > h(b)$, and $a'b'$ is a steep line segment. By Case (ii) of the lemma there exists a critical path from a' to b' in f. We claim that there exists a critical path from a to a' through at most $\lfloor \frac{d}{2} \rfloor + 1$ faces when $a \neq a'$, and that there exists a critical path from b' to b through at most $\lfloor \frac{d}{2} \rfloor + 1$ faces when $b \neq b'$. Because face f is used by all these three subpaths, the whole path uses at most $d + 1$ faces. The proof then follows from Lemma 5.2. We will now prove the first claim. The proof for the second claim is similar, and hence omitted.

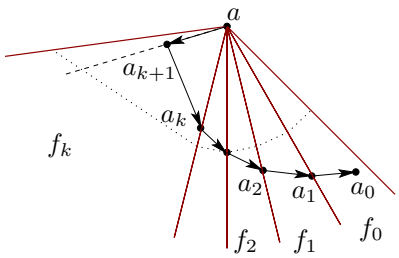


Figure 5.4: A critical path from a vertex a that is locally sharp in $f = f_0$, but not in f_k.

Since a is not a sharp vertex, it must have some incident face in which it is not locally sharp. Let $(f = f_0, f_1, f_2, \ldots, f_k)$ be (one of) the shortest sequence of faces around vertex a such that a is not a locally sharp vertex in f_k (Figure 5.4). Clearly, $k \leq \lfloor \frac{d}{2} \rfloor + 1$. We will

build as follows a critical path backward from a'. Let $a_0 = a'$. For each $i \in [0, k-1]$ in this order, since a is a locally sharp vertex in f_i, there is a point $a_{i+1} \in f_i \cap f_{i+1}$ such that $a_{i+1}a_i$ is a critical direction in f_i. The final point a_k lies on the edge between f_k and f_{k-1} which is a steep edge. Because a is not a locally sharp vertex in f_k, and no other vertex of f_k lies above the plane $z = h(a_k)$, there exists an interior point $a_{k+1} \in f_k$ such that both aa_{k+1} and $a_{k+1}a_k$ are critical directions. Clearly the path $(a, a_{k+1}, a_k, \ldots, a_0 = a')$ is a critical path through at most $\lfloor \frac{d}{2} \rfloor + 1$ faces.

□

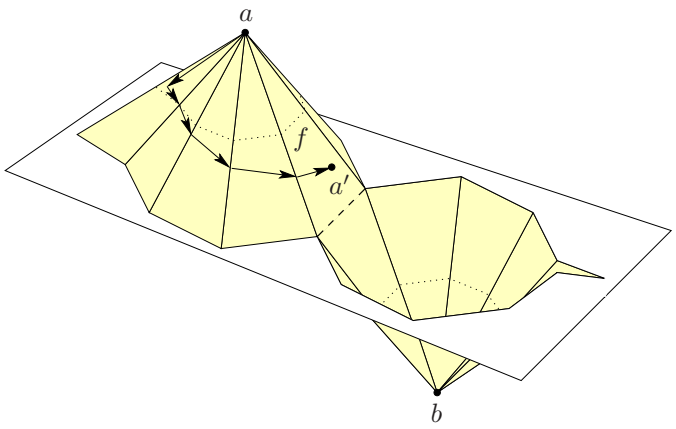


Figure 5.5: An SGDP between two locally sharp vertices a and b in f that goes through $\Theta(d)$ other faces.

It is easy to construct a terrain in which the number of faces in Case (iii) of the above lemma is exactly d, see Figure 5.5 for an example (note that face f is a quadrilateral in the figure, and triangulating f by adding edge ab keeps both a and b in a common face).

We would like the property of an SGDP that the path would visit a face at most once, because our method of approximating a path introduces some error each time the path crosses an edge. But unlike SDPs, this property does *not* hold for an SGDP. For example, the two SGDPs in Figure 5.1 visit a face twice. In fact, there is no bound on the number of times a face may be visited—the SGDP in Figure 5.1(a) can be made to spiral around the pyramid more and more times by making angle ψ closer and closer to $\frac{\pi}{2}$. We call a gently descending path *ideal* if it crosses the interior of each face at most once. The good news is that we can "convert" any non-ideal SGDP into an ideal one—we will prove this claim in the following lemma:

Lemma 5.4. *If there is a gently descending path from s to t, there exists an ideal SGDP from s to t.*

Proof. Let P_0 be a gently descending path from s to t. We first show that P_0 can be converted to an ideal path without increasing its length. Then we prove that the set of all ideal s-t paths is closed, which implies that the minimum length can be attained (not just approached). The lemma then follows immediately.

For the first claim, let f be a face whose interior intersects P_0, and let a_f [and b_f] be the infimum [respectively supremum] of the points in P_0 that lie in the *interior* of f (considering the natural ordering of the points in P_0, i.e., from s to t). Because both P_0 and f are closed sets, both a_f and b_f are points in P_0 on the boundary of f. It follows that neither a_f nor b_f is a locally sharp vertex in f by definition. So, Lemma 5.3(ii) implies that there is an SGDP from a_f to b_f that lies completely in f. Replacing the part of P_0 from a_f to b_f with an SGDP in f for all f yields an ideal s-t path P_1 which is not longer than P_0.

For the second claim we need to show that the infimum of the set of lengths of ideal gently descending paths from s to t is attained by an ideal path. Each ideal path corresponds to a sequence of faces with no repetition of faces. There are finitely many such face sequences. Thus it suffices to prove the claim for a single face sequence. This follows by induction on the length of the face sequence, based on the fact that from one face to the next face the path crosses an edge, which is a closed set. □

5.4 Approximation using uniform Steiner points

5.4.1 Algorithm

Preprocessing phase

In our first approximation algorithm (which is similar to the algorithm in Section 4.3), we preprocess the terrain by adding uniformly spaced Steiner points as follows. We take a set of level planes such that the distance between any two consecutive planes is at most $\delta = \frac{\epsilon h}{4n} \cos\theta \cos\psi$. We make sure that there is a level plane through every vertex. We then put Steiner points at all the intersection points of these planes with the non-level edges of the terrain. On the level edges, we add enough Steiner points so that consecutive points are at most $\delta \sec\theta$ units apart. We then construct a weighted directed graph $G(V, E)$ in which each node in V represents either a Steiner point or a vertex, and there is a directed link $pq \in E$ if and only if all of the following are true:

(i) p and q lie in a common face,

(ii) $h(p) \geq h(q)$, and

(iii) neither p nor q is a sharp vertex.

By Lemma 5.3, there is an SGDP from p to q. The weight of link pq is the length of such an SGDP, i.e., $\|pq\|$. In the final step of our preprocessing we compute a shortest path tree T rooted at s using the Bushwhack algorithm (discussed in page 15). The Bushwhack algorithm works for any distance metric that satisfies the following property: if e and e' are two edges from the same face, and a and b are two points on edge e, then edge e' can be divided into two sub-intervals A and B, where points in A have a shorter path from a than from b and points in B have a shorter path from b than from a. This property holds for our distance metric (even though an SGDP connecting two points in face f may leave f). The proof follows from Sun and Reif [91], Lemmas 3 and 4, where they showed that this property holds for anisotropic paths.

Query phase

To answer a query, we simply return the path from s to query point v in T if $v \in V$ and v is not a sharp vertex. If v is a sharp vertex, we return nothing since there is no SGDP from s to v. Otherwise, $v \notin V$. In this case, we find the node u among those in V lying in the face(s) containing v such that $h(u) \geq h(v)$, and the sum of $\|uv\|$ and the length of the path from s to u in T is minimum. Finally we return the corresponding path from s to v as an approximate SGDP.

There is an important issue regarding the path returned by our algorithm. It is a path in the graph augmented by vertex v. To obtain an actual path on the terrain we must replace each link ab in the path by an SGDP of the same length, which is possible by the definition of the links of the graph. Such an SGDP is not unique if ab is steep (Figure 5.3(a)), but it is easy to compute one such path. In the case where neither a nor b is locally sharp in their common face f, we can even compute an SGDP from a to b in f with the minimum possible number of bends (i.e., with no more bends than any other SGDP from a to b in f). Note, however, that the problem of minimizing the total number of bends in an SGDP is NP-hard, as we will see in Section 5.6.

5.4.2 Correctness and analysis

The proof of correctness follows an approach similar to the one in Section 4.3.2: we show that an ideal SGDP P from s to any point v in the terrain is approximated by a path P' from s to v in the augmented graph $G_v(V_v, E_v)$ constructed as follows. We first add v to graph G, and then add to this graph a link uv with weight $\|uv\|$ for every $u \in V$ such that u and v lie in a common face, and $h(u) \geq h(v)$.

Let $\sigma_P = (s = p_0, p_1, p_2, \ldots, p_k, v = p_{k+1})$ be an ordered subsequence of the nodes in P such that (a) for each $i \in [0, k]$, the segments in-between p_i and p_{i+1} lie in a common face f_i, and (b) for each $i \in [0, k-1]$, the segment exiting from p_{i+1} does *not* lie in f_i. Note that p_i and p_{i+1} are two different boundary points of face f_i for all $i \in [0, k-1]$, and p_k and p_{k+1} are two different points of face f_k (p_{k+1} can be an interior point of f_k). For all $i \in [0, k]$, the part of P between p_i and p_{i+1} remains in f_i. Let e_i be an edge of the terrain through p_i for all $i \in [1, k]$ (e_i can be any edge through p_i if p_i is a vertex).

We construct a node sequence $P' = (s = p'_0, p'_1, p'_2, \ldots, p'_k, v = p'_{k+1})$ as follows: for each $i \in [1, k]$, let $p'_i = p_i$ if p_i is a vertex of the terrain; otherwise, let p'_i be the nearest point from p_i in $V \cap e_i$ such that $h(p'_i) \geq h(p_i)$. We will first prove in Lemma 5.7 that this node sequence defines a path in G_v.

Lemma 5.5. *For all $i \in [0, k]$, $h(p'_i) \geq h(p'_{i+1})$.*

Proof. We have $h(p'_i) \geq h(p_i)$ and $h(p_i) \geq h(p_{i+1})$ as in the proof of Lemma 4.1, even though both p_i and p'_i are defined differently here. This lemma then follows from the rest of the proof of Lemma 4.1. □

Lemma 5.6. *For all $i \in [0, k+1]$, p'_i is not a sharp vertex.*

Proof. None of $p'_0 = s$ and $p'_{k+1} = v$ are sharp vertices because both the segments sp_1 and $p_k v$ are gently descending. For each $i \in [1, k]$, if p'_i is a sharp vertex, then p'_i is either the unique topmost vertex or the unique bottommost vertex in all incident faces. Therefore, either $h(p'_{i-1}) < h(p'_i) > h(p'_{i+1})$, or $h(p'_{i-1}) > h(p'_i) < h(p'_{i+1})$. Both of these are impossible by Lemma 5.5. So p'_i is a not sharp vertex. □

Lemma 5.7. *Node sequence P' defines a path in G_v.*

Proof. It is sufficient to show that $p'_i p'_{i+1} \in E_v$ for all $i \in [0, k]$. For $i \in [0, k-1]$, since p'_i and p'_{i+1} are boundary points of face f_i by definition, $h(p'_i) \geq h(p'_{i+1})$ by Lemma 5.5, and neither p'_i nor p'_{i+1} is a sharp vertex by Lemma 5.6, it follows from the construction that $p'_i p'_{i+1} \in E \subseteq E_v$. The case $i = k$ is similar except that p'_k and $v = p'_{k+1}$ are points (i.e., not boundary points) in face f_k, and that $p'_k v \in E_v$. □

Lemma 5.8. *For all $i \in [1, k]$, $\|p_i p'_i\| \leq \frac{\epsilon h}{4n}$.*

Proof. Using the current definition of δ in the proof of Lemma 4.3, we have: $|p_i p'_i| \leq \delta \sec\theta$ for all i. Observation 5.1(iii) then implies:

$$\|p_i p'_i\| \;\leq\; |p_i p'_i| \sec\psi \;\leq\; \delta \sec\theta \sec\psi \;=\; \frac{\epsilon h}{4n} \;,$$

which completes the proof. □

Lemma 5.9. *The algorithm returns a $(1+\epsilon)$-SGDP.*

Proof. Let P and P' be respectively an ideal SGDP and a node sequence in G_v as described above. Our algorithm finds a shortest path P'' in G_v, which provides a gently descending path of the same length. Since P' is a path in G_v (Lemma 5.7), the length of P'' is at most the length of P', and therefore it is sufficient to prove that the length of P' is at most $(1+\epsilon)$ times the length of P.

When $k = 0$, both P and P' have length $\|sv\|$, which proves the lemma trivially. When $k > 0$, the length of P' is equal to:

$$\begin{aligned}
\sum_{i=0}^{k} \|p'_i p'_{i+1}\| &\leq \sum_{i=0}^{k} \left(\|p'_i p_i\| + \|p_i p_{i+1}\| + \|p_{i+1} p'_{i+1}\|\right) \quad \text{(Observation 5.1(ii))} \\
&= \sum_{i=0}^{k} \|p_i p_{i+1}\| + 2\sum_{i=1}^{k} \|p_i p'_i\| \quad \text{(Observation 5.1(i))} \\
&\leq \sum_{i=0}^{k} \|p_i p_{i+1}\| + \frac{\epsilon h k}{2n} \qquad \text{(Lemma 5.8)} \\
&< \sum_{i=0}^{k} \|p_i p_{i+1}\| + \epsilon h \;,
\end{aligned}$$

since $k < 2n$. Now, because $k > 0$, we have:

$$h \;\leq\; |p_0 p_1| \;\leq\; \sum_{i=0}^{k} |p_i p_{i+1}| \;\leq\; \sum_{i=0}^{k} \|p_i p_{i+1}\|$$

by Observation 5.1(iii). Therefore, $\sum_{i=0}^{k} \|p'_i p'_{i+1}\| < (1+\epsilon) \sum_{i=0}^{k} \|p_i p_{i+1}\|$. So, the length of P' is at most $(1+\epsilon)$ times the length of P. □

Lemma 5.10. *Let $X = \frac{L}{h} \sec\theta \sec\psi$. Graph G has $O\left(\frac{n^2 X}{\epsilon}\right)$ nodes in total, and $O\left(\frac{nX}{\epsilon}\right)$ nodes along any edge of the terrain.*

Proof. The proof is the same as that of Lemma 4.5, except that we use δ and X defined here. □

Theorem 5.1. *Let $X = \frac{L}{h} \sec\theta \sec\psi$, where L is the length of the longest edge, h is the smallest distance of a vertex from a non-incident edge in the same face, θ is the largest acute angle between a non-level edge and a vertical line, and ψ is the angle of steepness. Given a vertex s, we can preprocess the terrain in $O\left(\frac{n^2 X}{\epsilon} \log\left(\frac{nX}{\epsilon}\right)\right)$ time after which we can determine a $(1+\epsilon)$-approximate SGDP from s to any query point v in:*

(i) $O(nd)$ time if v is a vertex or a Steiner point, and

(ii) $O\left(n\left(d + \frac{X}{\epsilon}\right)\right)$ time otherwise.

Proof. The approximation factor follows from Lemma 5.9.

The preprocessing time of our algorithm is the same as the running time of the Bushwhack algorithm, which is $O(|V| \log |V|) = O\left(\frac{n^2 X}{\epsilon} \log\left(\frac{nX}{\epsilon}\right)\right)$ by Lemma 5.10.

During the query phase, if v is a vertex or a Steiner point, the approximate path is in the tree T. Because the tree has height $O(n)$, it takes $O(n)$ time to trace the path in the tree. Tracing the corresponding path in the terrain takes $O(nd)$ time by Lemma 5.3. The total query time is thus $O(nd)$ in this case. If v is neither a vertex nor a Steiner point, v is an interior point of a face or an edge of the terrain. The last intermediate node u on the path to v is a vertex or a Steiner point that lies on the boundary of a face containing v. If v is interior to a face [an edge], there are 3 [respectively 4] edges of the terrain on which u can lie. Thus there are $O\left(\frac{nX}{\epsilon}\right)$ choices for u by Lemma 5.10, and we try all of them to find the shortest approximate distance from s to v. Finally tracing the corresponding path in the terrain takes $O(nd)$ time by Lemma 5.3. The total query time in this case is $O\left(\frac{nX}{\epsilon}\right) + O(nd) = O\left(n\left(d + \frac{X}{\epsilon}\right)\right)$. □

Corollary 5.1. *If the answer to a query is the length of an SGDP (rather than the SGDP itself), the query times for Cases (i) and (ii) of Theorem 5.1 become $O(1)$ and $O\left(\frac{nX}{\epsilon}\right)$, respectively.*

5.5 Approximation using non-uniform Steiner points

Our second approximation algorithm (which is similar to the algorithm in Section 4.4) differs from the first one only in the way Steiner points are placed. We follow the same scheme as in Section 4.4, except that the parameters δ_1 and δ_2 take into account the steepness. More

precisely, we place Steiner points in two phases. First, on every edge $e = v_1v_2$ we place Steiner points at points $p \in e$ such that $|pq| = \delta_1(1+\delta_2)^i$ for $q \in \{v_1, v_2\}$ and $i \in \{0, 1, 2, \ldots\}$, where

$$\delta_1 = \frac{\epsilon h}{6n} \cos\psi$$

and

$$\delta_2 = \frac{\epsilon h}{6L} \cos\psi \ .$$

In the second phase we slice the terrain with a level plane through every Phase 1 Steiner point and every vertex, and add Steiner points at the points where these planes intersect the terrain. We then construct a weighted directed graph $G(V, E)$, and then a shortest path tree T rooted at s in the same manner as in Section 5.4.1. The queries are handled in the same manner as in Section 5.4.1.

5.5.1 Correctness and analysis

For the proof of correctness, we follow the same approach used in Section 5.4.2: we show that an ideal SGDP P from s to any point v in the terrain is approximated by a path P' from s to v in the augmented graph $G_v(V_v, E_v)$ constructed as follows. We first add v to graph G, and then add to this graph a link uv with weight $\|uv\|$ for every $u \in V$ such that u and v lie in a common face, and $h(u) \geq h(v)$. We then construct a node sequence $P' = (s = p'_0, p'_1, p'_2, \ldots, p'_k, v = p'_{k+1})$ as in Section 5.4.2.

Lemma 5.11. *For all $i \in [0, k]$, $h(p'_i) \geq h(p'_{i+1})$.*

Proof. The proof is exactly the same as in Lemma 5.5. □

Lemma 5.12. *For all $i \in [0, k+1]$, p'_i is not a sharp vertex.*

Proof. The proof is exactly the same as in Lemma 5.6. □

Lemma 5.13. *Node sequence P' defines a path in G_v.*

Proof. The proof is exactly the same as in Lemma 5.7. □

For the rest of the lemmas, we rely on the same definition of vicinity of a vertex as in Section 4.4 except that we use δ_1 defined here.

Lemma 5.14. *For all $i \in [1, k]$ such that p_i is not inside a vertex vicinity, $\|p_ip'_i\| < \frac{\epsilon}{6} |p_{i-1}p_i|$.*

Proof. Using the same argument as in Lemma 4.8 but with δ_2 defined here, we have: $|p'_ip''_i| < \delta_2 |p_{i-1}p_i| \cdot \frac{L}{h}$. The lemma then follows from Observation 5.1(iii) and the definition of δ_2. □

Lemma 5.15. *For all $i \in [1, k]$ such that p_i is on or inside a vertex vicinity, $\|p_ip'_i\| \leq \frac{\epsilon h}{6n}$.*

Proof. Using the same argument as in Lemma 4.9 but with δ_1 defined here, we have: $|p_ip'_i| \leq \delta_1$. The lemma then follows from Observation 5.1(iii) and the definition of δ_1. □

Lemma 5.16. *The algorithm returns a $(1+\epsilon)$-SGDP.*

Proof. As in the proof of Lemma 5.9, it is sufficient to prove that the length of P' is at most $(1+\epsilon)$ times the length of P, where P and P' are respectively an ideal SGDP and a node sequence in G_v as described above.

When $k = 0$, both P and P' have length $\|sv\|$, which proves the lemma trivially. When $k > 0$, the length of P' is equal to:

$$\begin{aligned}
\sum_{i=0}^{k} \|p'_i p'_{i+1}\| &\leq \sum_{i=0}^{k} \|p_i p_{i+1}\| + 2\sum_{i=1}^{k} \|p_i p'_i\| && \text{(as in the proof of Lemma 5.9)} \\
&< \sum_{i=0}^{k} \|p_i p_{i+1}\| + 2\sum_{i=1}^{k} \left(\frac{\epsilon}{6}\|p_{i-1}p_i\| + \frac{\epsilon h}{6n}\right) && \text{(by Lemmas 5.14 and 5.15)} \\
&= \sum_{i=0}^{k} \|p_i p_{i+1}\| + \frac{\epsilon}{3}\sum_{i=1}^{k} \|p_{i-1}p_i\| + \frac{\epsilon h k}{3n} \\
&\leq \left(1+\frac{\epsilon}{3}\right)\sum_{i=0}^{k} \|p_i p_{i+1}\| + \frac{\epsilon h k}{3n} \\
&< \left(1+\frac{\epsilon}{3}\right)\sum_{i=0}^{k} \|p_i p_{i+1}\| + \frac{2\epsilon h}{3}\ ,
\end{aligned}$$

since $k < 2n$. Now, because $k > 0$, we have:

$$h \leq |p_0 p_1| \leq \sum_{i=0}^{k} |p_i p_{i+1}| \leq \sum_{i=0}^{k} \|p_i p_{i+1}\|$$

by Observation 5.1(iii). So, $\sum_{i=0}^{k} \|p'_i p'_{i+1}\| < (1+\epsilon)\sum_{i=0}^{k} \|p_i p_{i+1}\|$, and therefore, the length of P' is at most $(1+\epsilon)$ times the length of P. □

Lemma 5.17. *Let $X' = \frac{L}{h}\sec\psi$. Graph G has $O\left(\frac{n^2 X'}{\epsilon}\log\left(\frac{nX'}{\epsilon}\right)\right)$ nodes in total, and $O\left(\frac{nX'}{\epsilon}\log\left(\frac{nX'}{\epsilon}\right)\right)$ nodes along any edge of the terrain.*

Proof. The proof is the same as that of Lemma 4.11, except that we use δ_1, δ_2 and X' defined here. □

Theorem 5.2. *Let $X' = \frac{L}{h}\sec\psi$, where L is the length of the longest edge, h is the smallest distance of a vertex from a non-incident edge in the same face, and ψ is the angle of steepness. Given a vertex s, we can preprocess the terrain in $O\left(\frac{n^2 X'}{\epsilon}\log^2\left(\frac{nX'}{\epsilon}\right)\right)$ time after which we can determine a $(1+\epsilon)$-approximate SGDP from s to any point v in:*

(i) $O(nd)$ time if v is a vertex or a Steiner point, and

(ii) $O\left(nd + \frac{nX'}{\epsilon}\log\left(\frac{nX'}{\epsilon}\right)\right)$ time otherwise.

Proof. The proof is the same as in Theorem 5.1 except that we use Lemmas 5.16 and 5.17 instead of Lemmas 5.9 and 5.10, respectively. □

Corollary 5.2. *If the answer to a query is the length of an SGDP, the query times for Cases (i) and (ii) of Theorem 5.2 become* $O(1)$ *and* $O\left(\frac{nX'}{\epsilon}\log(\frac{nX'}{\epsilon})\right)$*, respectively.*

5.6 Hardness of SGDPs with few bends

We now turn to our hardness results. In this section we show using a reduction from 3-SAT that it is NP-hard to decide if there exists a gently descending path of length at most $\mathcal{L}$ that has at most k bends. The "big picture" of our construction follows immediately from the idea of Canny and Reif [24], who proved that finding a shortest path among obstacles in 3D is NP-hard. However, our gadgets are different from theirs because the paths in the SGDP problem lie on a 2D surface.

From the given 3-SAT instance f, our reduction constructs an instance of the SGDP problem consisting of a terrain that contains (i) a source vertex s, (ii) a destination vertex t, and (iii) faces bounded by axis-parallel edges, possibly with an axis-parallel rectangular hole. The angle of steepness ψ is part of the input to the SGDP instance. Note that it is trivial to triangulate each face of this terrain in constant time, to be consistent with the definition of the SGDP problem. But we avoid doing so because the "extra" non-axis-parallel edges only complicate the description.

We will start with an overview of our construction, where we will mention the *functions* of the main gadgets used in the construction. Then we will go into the details of our construction, describing the *structures* of all the gadgets in a bottom-up manner, i.e., beginning with the simplest gadgets and then gradually building the more "complicated" ones.

5.6.1 Overview

Given the 3-SAT instance f, we first construct a polynomial sized terrain that has exactly 2^n SGDPs with a limited number of bends from s to t. These 2^n paths are parallel to each other in most of the faces in the terrain, where they form a *Path Bundle*. Each path in the Path Bundle represents one of the 2^n possible truth assignments for the n variables. We then add m *Clause Filters* in series along the Path Bundle (as shown in Figure 5.6), one for each clause in f. As the name suggests, a Clause Filter "blocks" a path in the Path Bundle if the truth assignment corresponding to the path does not satisfy the clause corresponding to the filter. Inside each Clause Filter the Path Bundle splits into three and passes through three *Literal Filters*, one for each literal in the corresponding clause. A Literal Filter has an analogous function: it "blocks" a path if the truth assignment corresponding to the path does not satisfy the literal corresponding to the filter. A path in the Path Bundle that "survives" all the filters gives a solution to f.

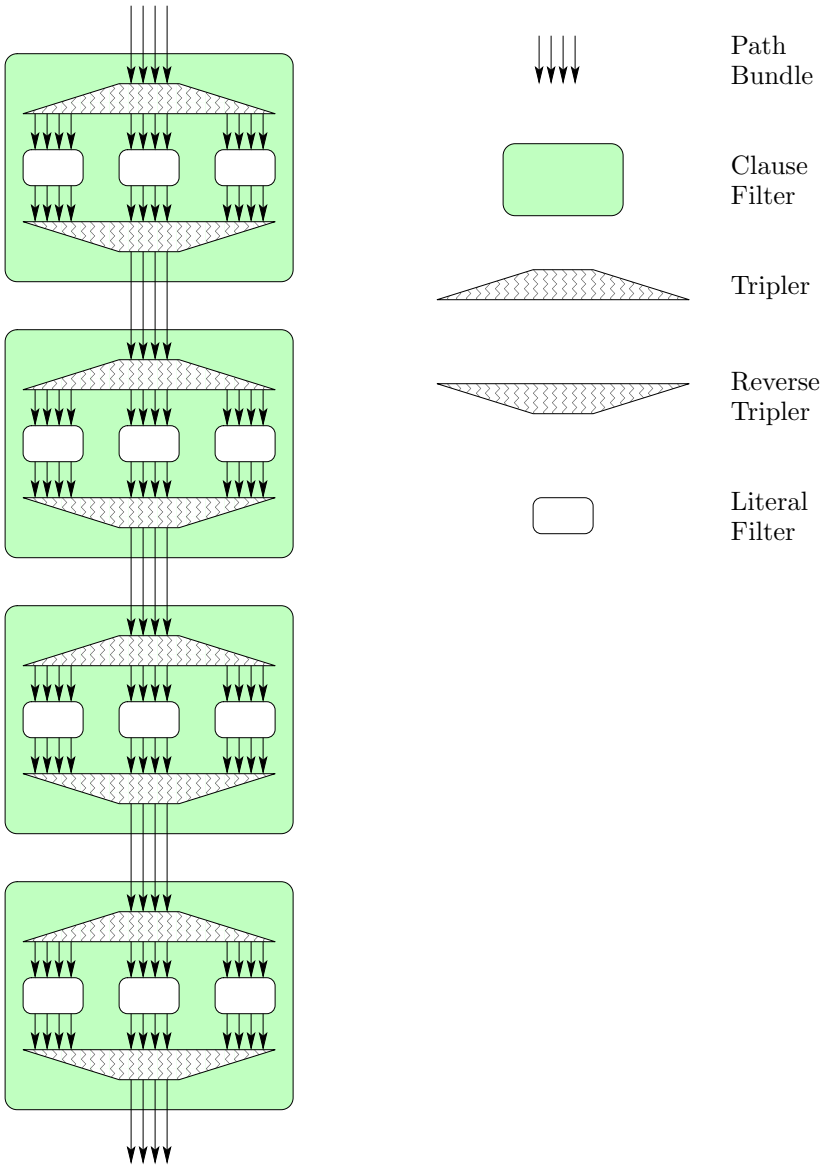


Figure 5.6: Clause Filters along the Path Bundle.

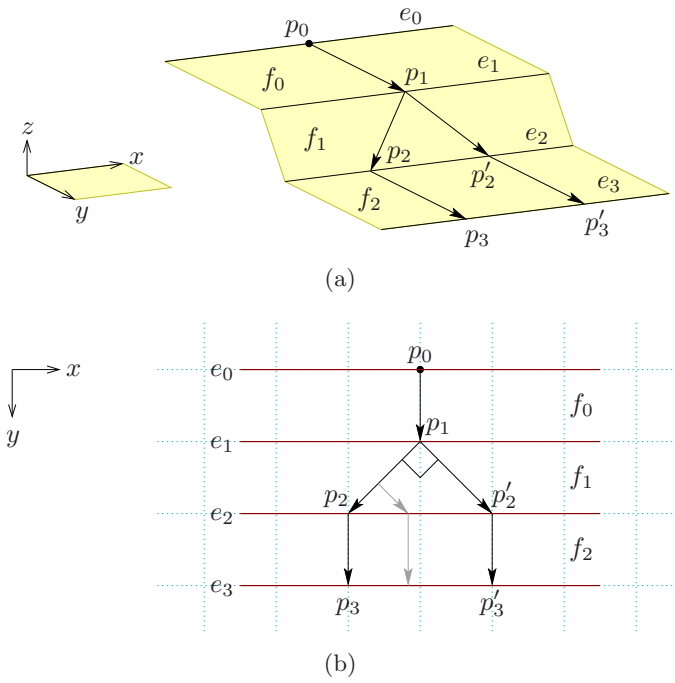


Figure 5.7: (a) A Splitter, and (b) its unfolded view.

5.6.2 Elementary gadgets

All the gadgets used in our construction are made up of two elementary gadgets. The first one is a *Splitter* which "splits" an SGDP into two SGDPs with exactly two bends each.

A Splitter consists of three faces bounded by edges parallel to the x-axis, as shown in Figure 5.7(a). Faces f_0 and f_2 are horizontal, and the slope of face f_1 is such that the angle between the two critical directions in this face is $\frac{\pi}{2}$. This ensures that if we unfold the faces onto the xy-plane (as shown in Figure 5.7(b)), each of the critical directions in f_1 will make an angle $\frac{\pi}{4}$ with the direction of the y-axis. This is possible when, for example, face f_1 is vertical and the angle of steepness ψ is $\frac{\pi}{4}$. (Note that our definition of a terrain in Section 2.1 allows vertical faces.) Our construction uses these particular values for the slope of f_1 and ψ because they make the construction simple. Later on in Section 5.6.7 we will mention how to adapt our construction when vertical faces are not allowed.

To understand how a Splitter works, take any point $p_0 \in e_0$ as in Figure 5.7(a), and then construct two different paths (p_0, p_1, p_2, p_3) and (p_0, p_1, p'_2, p'_3) as follows. Let $p_1 \in e_1$ be the point such that p_0p_1 is parallel to the y-axis. Let p_2 and p'_2 be two different points in e_2 such that p_1p_2 and $p_1p'_2$ are two different critical directions in f_1 from p_1. Finally, let p_3 and p'_3 be the points in e_3 such that both p_2p_3 and $p'_2p'_3$ are parallel to the y-axis. Now observe that the SGDP from p_0 to any point in e_3 must leave e_0 along the direction parallel to the y-axis, and must reach e_3 in the same direction. It then follows that both of the paths (p_0, p_1, p_2, p_3) and (p_0, p_1, p'_2, p'_3) are SGDPs, and they have the same length. Moreover, each of the paths has two bends. It also follows that these paths are shorter than the SGDP from p_0 to any point in e_3 that lies outside $p_3p'_3$. Also, the SGDP from p_0 to any point in the interior of $p_3p'_3$ has at

least three bends—the gray segments in Figure 5.7(b) form such an SGDP. Thus from p_0 to edge e_3 there are exactly two SGDPs with at most two bends each, and this is true for *any* the position of p_0 along e_0. Therefore, each SGDP from a point in edge e_0 to edge e_3 is split into two SGDPs with two bends each, which justifies the gadget's name. Note that using the same gadget, two SGDPs from edge e_0 to edge e_3 can be merged together to reach a single point in e_3, where each of the SGDPs has two bends.

The gap between the two SGDPs in f_2 (i.e., the length of segment p_3p_3') plays an important role in our construction, and we will need Splitters with many different gaps. Because the gap is twice the distance between e_1 and e_2, it is easy to construct a Splitter with any desired gap.

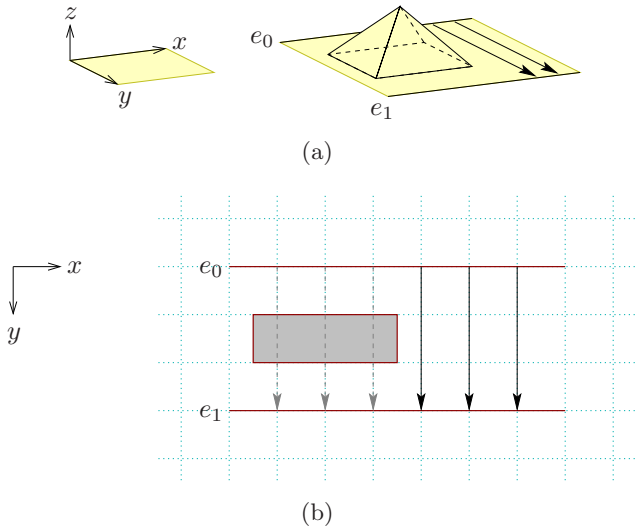


Figure 5.8: (a) A Blocker, and (b) its 2D representation.

The other elementary gadget is a *Blocker*, which is a pyramid with an axis-parallel rectangular base in a horizontal face. A base of the pyramid always lies inside a face bounded by edges parallel to the x-axis, as shown in Figure 5.8. If we want to trace an SGDP from e_0 to e_1, we must follow a segment in the direction of the y-axis that does not go through the Blocker. If the segment "tries" to go through the Blocker, we must travel around or over the pyramid, which costs more in terms of length and number of bends. (Note that we can also define a Blocker as an axis-parallel rectangular hole in the terrain.)

For the ease of discussion in the rest of this section, we unfold the terrain onto the xy-plane. We will use a shaded rectangles as in Figure 5.8(b) to represent a Blocker.

5.6.3 The Path Bundle and its labeling

Given the 3-SAT instance f, we first construct as follows a polynomial sized terrain that has exactly 2^n SGDPs with a limited number of bends. Let $s = (0, 0)$, and t be a point on the positive side of the y-axis. Add n Splitters near s and n Splitters near t as shown in Figure 5.9, so that we have from s to t exactly 2^n SGDPs with $4n$ bends each. In the middle face of the terrain in Figure 5.9, all these 2^n paths are distinct, parallel to the y-axis and equidistant from the nearest paths in the sequence. This sequence of paths forms the Path Bundle.

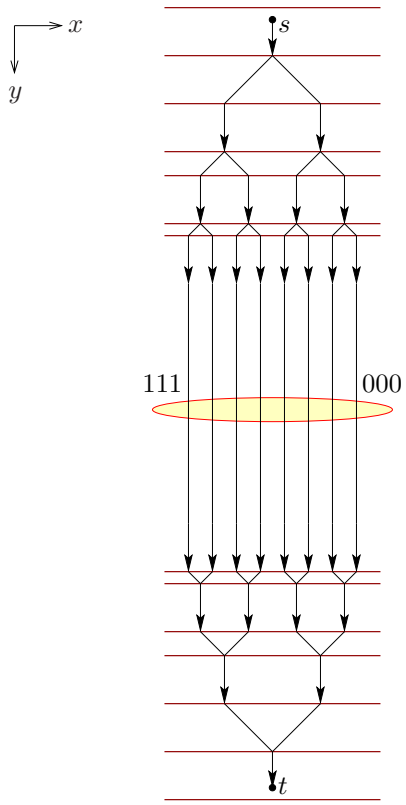


Figure 5.9: The Path Bundle in the initial terrain ($n = 3$).

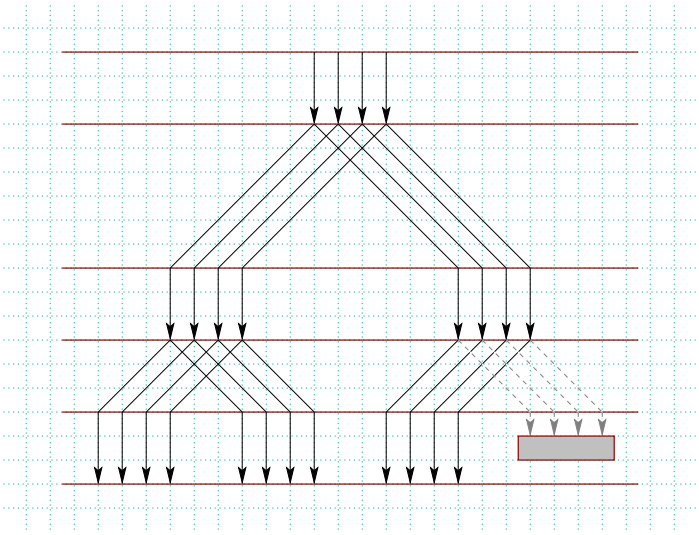
The paths in the Path Bundle are labeled with the binary representations of the integers from 0 to $2^n - 1$ inclusive, from right to left as in Figure 5.9. More precisely, the ith path from the right in the figure is labeled with the binary representation of i, where the 0th path is the rightmost one. As we have mentioned before, each of these paths represents one possible truth assignment for the n variables in f. More precisely, using the convention that the least significant bit is the 0th bit, the ith bit in the label represents the "value" of b_i. For example, when $n = 4$, the Path Bundle contains 16 paths, and the third path from the right has the label 0010, which represents the truth assignment $b_3 = 0$, $b_2 = 0$, $b_1 = 1$ and $b_0 = 0$.

In the subsequent figures, a set of closely-packed "parallel" paths as in Figure 5.9 will denote the Path Bundle; the number of such paths in a figure is not important unless n is mentioned in the caption.

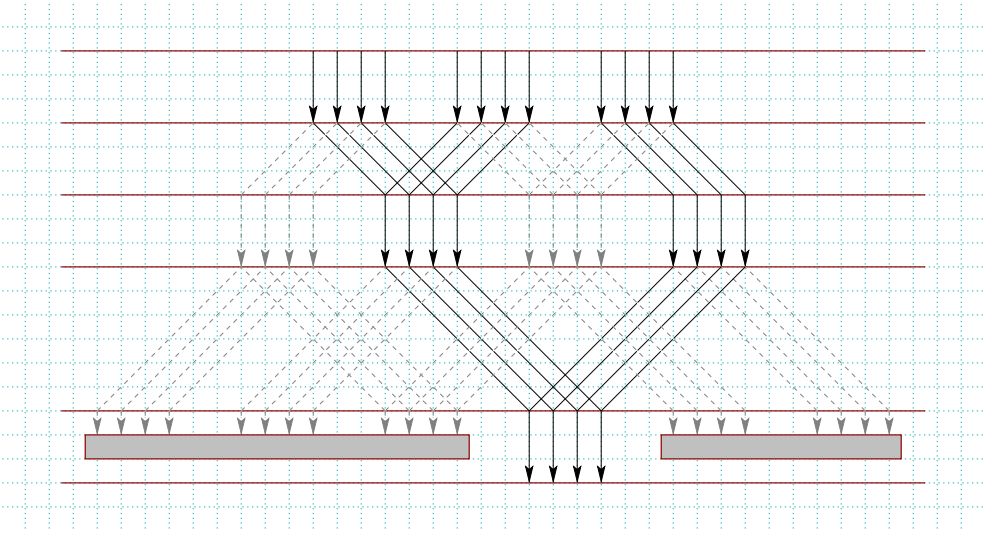
5.6.4 Intermediate gadgets

We will soon add more gadgets in the middle face of the terrain in Figure 5.9. We will now describe all those gadgets, starting in this section with three intermediate gadgets that consist of a constant number of elementary gadgets.

A *Tripler* splits the Path Bundle into three non-overlapping copies of the bundle. By "non-overlapping" we mean the three copies are disjoint in their x-extent. Each of the three copies



(a)



(b)

Figure 5.10: (a) A Tripler and (b) a Reverse Tripler.

of the Path Bundle contains the labeled paths in their original ordering. A Tripler consists of two Splitters and a Blocker as shown in Figure 5.10(a).

A *Reverse Tripler* merges three copies of the Path Bundle made by a Tripler into one. It consists of two Splitters and two Blockers as shown in Figure 5.10(b). Note that a Reverse Tripler brings together those paths in the copies of the bundle that have a common label. Triplers and Reverse Triplers come in pairs, as shown in Figure 5.6.

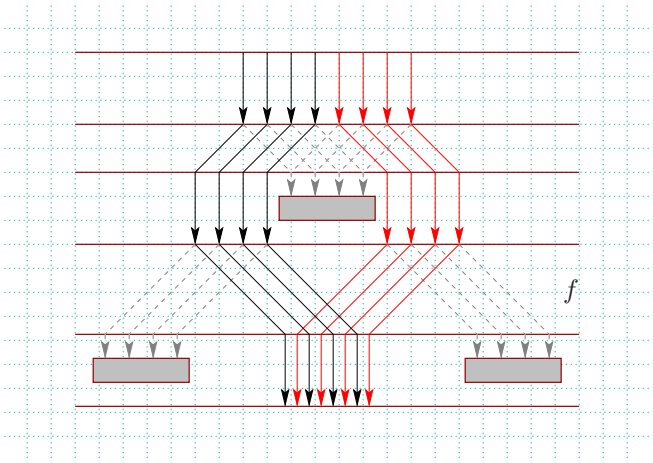


Figure 5.11: A Shuffler.

A *Shuffler* makes a perfect shuffle of the paths in the Path Bundle, as shown in Figure 5.11. To be precise, each of the leftmost 2^{n-1} paths in the bundle goes in-between a pair of consecutive paths in the rightmost 2^{n-1} paths in such a way that:

(i) no two paths from the left half become consecutive,

(ii) the rightmost path remains the rightmost one, and

(iii) the relative order of the paths in each half of the bundle is maintained.

A Shuffler consists of two Splitters and three Blockers.

Note that none of our gadgets except a Shuffler changes the relative order of the paths in the Path Bundle. This is true even for the gadgets containing Shufflers—we will describe them soon.

Another important issue is that a Shuffler makes the x-extent of the Path Bundle smaller, by halving the gap between every pair of consecutive paths. As a result, after we add each Shuffler, we will have to "adjust" the sizes of all the gadgets that come after this Shuffler on the Path Bundle.

Observation 5.2. *After going through a Shuffler, the ith path in the Path Bundle becomes the jth path, where j is the integer obtained by rotating the n-bit binary representation of i to the left by one bit.*

It follows from the above observation that after the Path Bundle goes through a Shuffler, every path whose label has a one [zero] in the second most significant bit appears at the left [respectively right] half of the modified bundle. It then follows by induction that after the *original* Path Bundle goes through i Shufflers for some $i < n$, a path whose label has a one [zero] in the $(n-1-i)$th most significant bit appears at the left [respectively right] half of the bundle at the "output".

5.6.5 Main gadgets

There are two types of main gadgets: Literal Filters and Clause Filters. We have already mentioned their functions in Section 5.6.1.

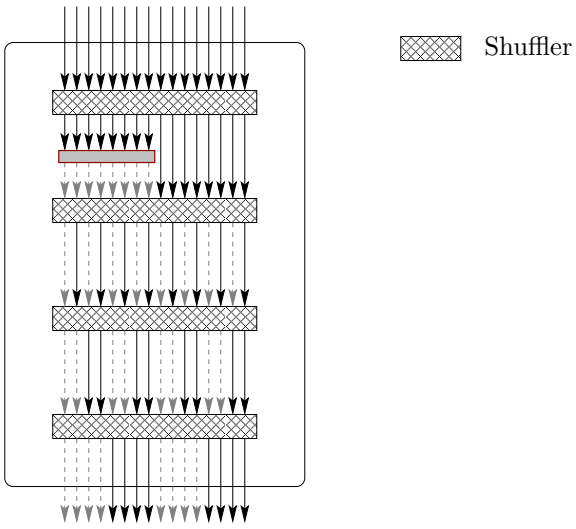


Figure 5.12: The structure of the Literal Filter for $\bar{b_2}$ $(n = 4)$.

A Literal Filter represents a literal in a clause. It consists of a sequence of n Shufflers and a Blocker as shown in Figure 5.12. The Path Bundle entering a Literal Filter passes through all the n Shufflers in sequence, and the Blocker obstructs half of the paths in the Path Bundle before they enter one of the Shufflers. To be more precise, if the literal represented by the Literal Filter is $\bar{b_i}$, the Blocker obstructs the leftmost 2^{n-1} paths in the Path Bundle right before they enter the $(n-i)$th Shuffler (note that "1th Shuffler" is the first one). On the other hand, if the literal is b_i, the Blocker obstructs the rightmost 2^{n-1} paths right before they enter the same Shuffler. For example, the Literal Filter in Figure 5.12 corresponds to the literal $\bar{b_2}$ $(n = 4)$.

A Clause Filter represents one of the m clauses. It consists of a Tripler, a Reverse Tripler, and three Literal Filters, one for each literal in the corresponding clause. See Figure 5.6. Each of the three copies of the Path Bundle inside a Clause Filter goes through one of the Literal Filters.

Because the Path Bundle passing through a Literal Filter goes through n Shufflers in sequence, it follows from Observation 5.2 that the Literal Filter keeps the the relative order of the paths in the Path Bundle unchanged. Clearly, a Clause Filter also maintains the order.

5.6.6 Correctness

To prove that deciding the existence of a gently descending path of length at most $\mathcal{L}$ that has at most k bends is NP-hard, we will have to complete our reduction by determining the values of $\mathcal{L}$ and k. Let $\mathcal{L}$ be the length of any path in the Path Bundle when the Blockers from all Literal Filters are *temporarily* removed so that no paths in the Path Bundle is blocked.

Lemma 5.18. *Any path P in the Path Bundle has length $\mathcal{L}$, and has $k = 4mn + 8m + 4n$ bends, provided that P is not blocked by a Blocker in a Literal Filter. Moreover, no s-t path in the terrain is better than P in terms of both length and number of bends.*

Proof. For the number of bends, we will first count the number of Splitters used by P. A Literal Filter contains n Shufflers, each of which contains of two Splitters. So, P goes through $2n$ Splitters in a Literal Filter. In every Clause Filter, P goes through 2 Splitters in the Tripler, $2n$ Splitters in a Literal Filter and 2 Splitters in the Reverse Tripler. The total number of Splitters used in all Clause Filters is $m(2n+4) = 2mn+4m$. There are n Splitters before the Path Bundle, and after n after it. So the total number of Splitters used by P is $2mn+4m+2n$. The number of bends then follows since in each Splitter P has two bends.

The length of P follows from the definition of $\mathcal{L}$, and from the fact that all paths in the Path Bundle have the same length by construction. The last part follows from the construction of the elementary gadgets. □

Lemma 5.19. *A path P in the Path Bundle passes through the Literal Filter for a literal l in f iff either:*

(i) the ith bit of the label of P is one, and $l = b_i$, or

(ii) the ith bit of the label of P is zero, and $l = \bar{b_i}$.

Proof. In this Literal Filter, the Blocker appears right before the $(n-i)$th Shuffler. When the ith bit of the label of P is one, it follows from Observation 5.2 that in the Path Bundle at this place, P is one of the 2^{n-1} leftmost paths. By construction, the Blocker does not obstruct this position of the Path Bundle iff $l = b_i$. The proof is similar when the ith bit of the label of P is zero. □

Lemma 5.20. *A path P in the Path Bundle passes through all the Clause Filters iff the truth assignment corresponding to the label of P satisfies f.*

Proof. $\Rightarrow$: If P passes through all the Clause Filters, it passes through at least one Literal Filter in every Clause Filter. It then follows from Lemma 5.19 that the corresponding literals in f are satisfied by the truth assignment corresponding to the label of P. Since at least one literal in every clause in f is satisfied, f is satisfied by the truth assignment.

$\Leftarrow$: It is sufficient to show that if P does not pass through the Clause Filter for C_i for some $i \in [0, m-1]$, the truth assignment corresponding to the label of P does not satisfy f. In this case, P does not pass through any of the three Literal Filters in the Clause Filter for C_i by construction. Lemma 5.19 then implies that the corresponding literals in f are not satisfied by the truth assignment. So, the truth assignment does not satisfy C_i, and therefore does not satisfy f. □

Lemma 5.21. *Our construction can be described using $O(mn)$ bits, and the construction takes $O(mn)$ time to build.*

Proof. We will first measure, in the unfolded terrain, the lengths (along the coordinate axes) of each gadget in terms of the gap between two consecutive paths in the Path Bundle *inside that gadget*. Since a Shuffler halves the gap between two consecutive paths in the bundle, we will measure in terms of the gap in the "narrower" bundle for a Shuffler.

First observe that the smallest distance between any two "features" of a gadget is half the gap between two consecutive paths in the Path Bundle leaving the gadget. This smallest measure appears either as the distance of a vertex of the Blocker from the nearest path in the Path Bundle (in Figures 5.10(a), 5.10(b), 5.11 and 5.12), or as the distance between the parallel edges of the "nearest" Splitters at the two ends of the Path Bundle in Figure 5.9.

It can be shown that for every gadget, the distance along each of the coordinate axes between every pair of features (in terms of the gap between two consecutive paths in the Path Bundle in that gadget) is a fraction of the form $\frac{r}{2}$, where r is an integer. Clearly r is an $O(n)$ bit integer for all the gadgets. In fact, r is equal to one for the "smallest measures" mentioned above, and is an even number for all other cases except that it is an odd number for the distance between the parallel edges of f in the Shuffler in Figure 5.11.

Now we will switch to a "global" measure. Since each Shuffler halves the gap between two consecutive paths in the bundle, the gap is the smallest (in the whole construction) at the end of the Path Bundle in Figure 5.9. Let the measure of this smallest gap be two *units* (because it follows from the above paragraph that choosing "two units" instead of "one unit" makes the gap between any two "features" an integer). The largest gap between two consecutive paths in the bundle is $2^{O(mn)}$ units at the beginning of the Path Bundle in Figure 5.9. So for every gadget, the distance (along each of the coordinate axes) between every pair of features is $2^{O(mn)} \times 2^{O(n)} = 2^{O(mn)}$ units.

Finally, the number of elementary gadgets used in our construction is $O(mn)$ because there are m Clause filters, and each of these filters has $O(n)$ elementary gadgets. Since $O(mn) \times 2^{O(mn)}$ is $2^{O(mn)}$, the unfolded terrain can be described using $2^{O(mn)}$ bits. The bound holds for the description length of the original terrain (i.e., the one before folding) because each non-level face of the original terrain is vertical.

The lemma then follows since $\psi = \frac{\pi}{4}$ can be specified using a constant number of bits when expressed as a multiple of π. □

Theorem 5.3. *Deciding if there exists a gently descending path of length at most $\mathcal{L}$ that has at most k bends is NP-hard.*

Proof. Lemma 5.21 implies that ours is a polynomial-time reduction. The theorem then follows from Lemmas 5.18 and 5.20, and the NP-hardness of 3-SAT [48]. □

5.6.7 Constructing a terrain without vertical faces

We assumed so far that $\psi = \frac{\pi}{4}$ and the middle face of a Splitter (i.e., face f_1 in Figure 5.7) is vertical. The classical definition of a terrain does not allow vertical faces, and in that case, we

adapt our construction of a Splitter as follows. We relax the restriction that the angle between the two critical directions in f_1 is $\frac{\pi}{2}$. To keep the description length of a Splitter limited, we ensure that:

(i) the ratio of the distance of e_2 from p_1 and the distance $\frac{|p_2p_2'|}{2}$ is a rational number, and

(ii) the coordinates of e_1 and e_2 are rational numbers both before and after unfolding f_1 onto the xy-plane.

We then scale the coordinate system appropriately to make all the coordinates integer.

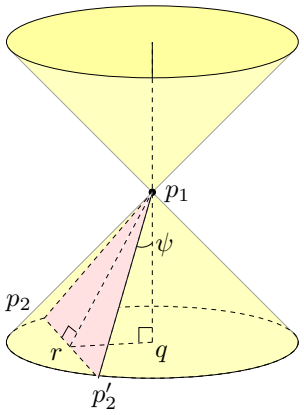


Figure 5.13: Determining the values of ψ, and the slope of f_1 when f_1 is not vertical.

We choose the value of ψ, and the slope of f_1 as follows. Let $\psi = \arccos(\frac{4}{13})$. Consider the 3D cone in Figure 5.13 that defines the steepness at point p_1 of the 3D Splitter in Figure 5.7(a). Let r be the midpoint of p_2p_2', and q be the point on the level plane containing p_2 (and p_2') such that p_1q is a vertical line. We choose the slope of f_1 in such a way that $|qr| = 3\kappa$ and $|p_1q| = 4\kappa$, thus $|p_1r| = \sqrt{3^2 + 4^2} = 5\kappa$, for some real number $\kappa > 0$. It follows from the value of $\psi = \angle qp_1p_2'$ that $|p_1p_2| = |p_1p_2'| = 13\kappa$, and therefore, $|rp_2| = |rp_2'| = \sqrt{13^2 - 5^2} = 12\kappa$.

The critical directions in f_1 now make an angle greater than $\frac{\pi}{2}$. Because the ratio $\frac{|p_1r|}{|rp_2|} = \frac{5}{12}$, all the y-coordinates in the unfolded terrain become integral multiples of $(\frac{1}{12})^{k'}$, where k' is the total number of Splitters in the terrain. If we now "undo" unfolding, all the y-coordinates and all the z-coordinates become integral multiples of $(\frac{1}{12} \cdot \frac{1}{5})^{k'}$. Since k' is $O(mn)$ (to be precise, $2mn + 4m + 2n$, as shown in the proof of Lemma 5.18), multiplying all the coordinates with $(5 \cdot 12)^{k'}$ makes them $2^{O(mn)}$ bit integers.

By specifying the steepness using $\cos\psi$ (or $\sin\psi$ or $\tan\psi$) instead of ψ in the input to the SGDP algorithm, Theorem 5.3 holds even when vertical faces are not allowed in a terrain.

5.7 Hardness of SGDPs with limited total turn-angle

To show that deciding existence of a gently descending path of length at most $\mathcal{L}$ that has a total turn-angle at most ϕ is NP-hard, we use the same construction we have used in Section 5.6.

Because the turn-angle is $\frac{\pi}{2}$ at every bend in each path of the Path Bundle in Section 5.6, the following lemma holds in a straightforward manner:

Lemma 5.22. *Any path P in the Path Bundle has length $\mathcal{L}$, and has a total turn-angle $\phi = (2mn + 4m + 2n)\pi$, provided that P is not blocked by a Blocker in a Literal Filter. Moreover, no s-t path in the terrain is better than P in terms of both length and total turn-angle.*

Proof. The lemma follows from Lemma 5.18 since $k \cdot \frac{\pi}{2} = (2mn + 4m + 2n)\pi$. □

Our second hardness result then follows:

Theorem 5.4. *Deciding if there exists a gently descending path of length at most $\mathcal{L}$ that has a total turn-angle at most ϕ is NP-hard.*

Proof. The proof is the same as that of Theorem 5.3, except that we use Lemma 5.22 instead of Lemma 5.3. □

The argument in Section 5.6.7 implies that the above theorem holds even when vertical faces are not allowed in a terrain.

5.8 Conclusion

In this chapter we have introduced the shortest gently descending path (SGDP) problem, and have deduced some properties of SGDPs that led us to two easy-to-implement approximation algorithms for SGDPs. Both of ours algorithms are based on the Steiner point approach. Between a pair of points there can be many SGDPs with different number of bends. We have shows using a reduction from 3-SAT that it is NP-hard to decide the existence of a gently descending path that has a length at most $\mathcal{L}$ and has (i) at most k bends, or (i) a total turn-angle at most ϕ. This result implies the hardness of the following two optimization problems:

(a) finding an SGDP that has either at most k bends, or a total turn-angle at most ϕ, and

(b) finding a gently descending path of length is at most $\mathcal{L}$ that has (i) fewest bends or (ii) smallest total turn-angle.

The second problem may have significance in, say, robot motion planning, where $\mathcal{L}$ is the maximum distance the robot can cover without refueling/recharging, and the goal is to minimize the number or the amount of turns made by the robot. Our hardness results apply to the shortest anisotropic path problem, and provide the first such result for a well-studied cost model for this problem.

The complexity of finding an SGDP is unknown even when we are given the sequence of faces used by the path. In fact, the numerical issues in computing the exact SDP through a given sequence of faces from a given source point to a given target point (discussed in Section 3.6) applies also to SGDPs because the additional constraint disallowing a very steep descent does not make the problem any easier.

We discussed a few properties of an SGDP in Section 5.3. We still do not know how the bend angles of an SGDP are related to one another. A characterization of those angles, like the one for SDPs in Chapter 3, may be helpful in identifying terrains for which the SGDP problem is easily solvable. Such a characterization may also be helpful in finding a faster approximation scheme. Of course, research in this latter direction will deserve attention only if a similar approach for SDP proves to be successful.

We presented two $(1 + \epsilon)$-approximation algorithms for SGDPs that are very similar to our approximation algorithms in Chapter 4. The discussion in Section 4.5 regarding possible extensions of our SDP algorithms applies to our SGDP algorithms in this chapter.

Chapter 6

Shortest Paths avoiding Forbidden Subpaths

6.1 Introduction

In this chapter of the thesis we switch our focus from shortest paths on terrains to shortest paths in graphs. Finding shortest paths in graphs is one of the most fundamental combinatorial optimization problems. It is a more fundamental problem than the geometric shortest path problem, and is used as a "black-box" in many algorithms for geometric shortest paths. For example, all the terrain shortest path algorithms mentioned in this thesis that are based on the Steiner point approach rely on graph shortest path algorithms.

It is natural both in geometric settings and in graphs to have certain subpaths that are forbidden. For example, if we disallow sharp bends in a path on a terrain, all two-segment paths with sharp bends in the terrain become forbidden. In this chapter deal with a similar problem in graphs. More precisely, we study in this chapter the following variant of the graph shortest path problem: given a weighted graph $G(V, E)$, and vertices s and t, and given a set X of *forbidden paths* in G, find a shortest s-t path P such that no path in X is a subpath of P. We call paths in X *exceptions*, and we call the desired path a *shortest exception avoiding path*. We allow an exception avoiding path to be non-simple, i.e., to repeat vertices and edges. In fact the problem becomes hard if the solution is restricted to simple paths [94]. This problem has been called the *shortest path problem with forbidden paths* by Villeneuve and Desaulniers [99]. Unlike them, we assume no a priori knowledge of X. More precisely, we can identify a forbidden path only after failing in our attempt to follow that path. This variant of the problem has not been studied before. It models the computation of shortest paths in optical networks, described in more detail in the "Motivation" section below. Note that when we fail to follow a path because of a newly discovered exception, we are still interested in a shortest path from s to t as opposed to a detour from the failure point. This is what is required in optical networks, because intermediate nodes do not store packets, and hence s must resend any lost packet.

This chapter presents an algorithm to compute shortest exception avoiding paths in the model where exceptions are not known a priori. Our algorithm is generic and depends on a (conventional) shortest path algorithm, such as Dijkstra's algorithm in the case of non-negative

edge-weights. Dijkstra's algorithm is a special case of a shortest path algorithm based on the relaxation process (see Section 6.1.3), and we can use any shortest path algorithm based on this process. The running time of our algorithm depends on the run-time of the specific relaxation-based shortest path algorithm we use. Specifically, our algorithm takes $O(k(n+m+T(n,m)))$ time, where $n = |V|$, $m = |E|$, k is the number of exceptions in X, and $T(n,m)$ is the run-time of the relaxation-based shortest path algorithm. In particular, $T(n,m)$ is $n+m$ for directed acyclic graphs, $n \log n + m$ for graphs with non-negative edge-weights, and nm for graphs with negative edge-weights but no negative weight cycle.[1]

Our algorithm uses a vertex replication technique similar to the one used to handle non-simple paths in other shortest path problems [39, 99]. The idea is to handle a forbidden path by replicating its vertices and judiciously deleting edges so that one copy of the forbidden path is missing its last edge and the other copy is missing its first edge. The result is to exclude the forbidden path but allow all of its subpaths. The main challenge is that vertex replication can result in an exponential number of copies of any forbidden path that overlaps the current one. Villeneuve and Desaulniers [99] address this challenge by identifying and compressing the overlaps of forbidden paths, an approach that is impossible for us since we do not have access to X. Our new idea is to couple vertex replication with the "growth" of a shortest path tree. By preserving certain structure in the shortest path tree we prove that the extra copies of forbidden paths that are produced during vertex replication are immaterial. Our algorithm is easy to implement, yet the proof of correctness and the run-time analysis are non-trivial.

6.1.1 Motivation

Our research on shortest exception avoiding paths was motivated by a problem in optical network routing from Nortel Networks. In an optical network when a ray of light of a particular wavelength tries to follow a path P consisting of a sequence of optical fibers, it may fail to reach the endpoint of P because of various transmission impairments such as attenuation, crosstalk, dispersion and non-linearities [51, 66]. This failure may happen even though the ray is able to follow *any* subpath P' of P. This "non-transitive behavior" occurs because those impairments depend on numerous physical parameters of the traversed path (e.g., length of the path, type of fiber, wavelength and type of laser used, location and gain of amplifiers, number of switching points, loss per switching point, etc.), and the effect of those parameters may be drastically different in P than in P' [17]. Forbidden subpaths provide a straightforward model of this situation.

We now turn to the issue of identifying forbidden paths. Because of the large number of physical parameters involved, and also because many of the parameters vary over the lifetime of the component [18], it is not easy to model the feasibility of a path. Researchers at Nortel suggested a model whereby an algorithm identifies a potential path, and then this path is tried out on the actual network. In case of failure, further tests can be done to pinpoint a minimal forbidden subpath. Because such tests are expensive, a routing algorithm should try out as few paths as possible. In particular it is practically impossible to identify all forbidden paths ahead of time—we have an exponential number of possible paths to examine in the network.

[1] The algorithm for graphs with non-negative edge-weights appeared in our STACS paper (Ahmed and Lubiw [10]).

This justifies our assumption of having no a priori knowledge of the forbidden paths, and of identifying forbidden paths only by testing feasibility of a path.

The shortest exception avoiding path problem may also have application in vehicle routing. Forbidden subpaths involving pairs of edges occur frequently ("No left turn") and can occur dynamically due to rush hour constraints, lane closures, construction, etc. Longer forbidden subpaths are less common, but can arise, for example if heavy traffic makes it impossible to turn left soon after entering a multi-lane roadway from the right. A vehicle in a road network on the slope of a mountain can encounter even longer forbidden paths because of various "mechanical" constraints. For example, going up [or down] for a certain time period may be too stressful for the engine [respectively, the brakes], and the time period may depend on dynamic factors like speed of traffic flow. If we are routing a single vehicle it is more natural to find a detour from the point of failure when a forbidden path is discovered. This is different from our model of rerouting from s upon discovery of a forbidden path. However, in the situation when vehicles are dispatched repeatedly from a service provider, our model does apply.

6.1.2 Preliminaries

We are given a directed graph $G(V, E)$ with $n = |V|$ vertices, $m = |E|$ edges, and a weight $w(u, v)$ for every edge $(u, v) \in E$. We are also given a source vertex $s \in V$ such that every other vertex in G is reachable from s, a destination vertex $t \in V$, and a set X of paths in G. The graph G together with X models a communication network, with edge-weights denoting the physical lengths of the links, where a packet cannot follow any path in X because of the physical constraints mentioned in Section 6.1.1. We assume that the algorithm can access the set X of forbidden paths only by performing queries to an oracle. Each query is a path P, and the oracle's response is either the confirmation that P is exception avoiding, or else an exception $x \in X$ that is a subpath of P and whose last vertex is earliest in P. Ties can be broken arbitrarily. In our discussion we say "we try a path" instead of saying "we query the oracle" because the former is more intuitive. In Section 6.6 we modify our algorithm for the case of an oracle that returns *any* exception on a path (not just the one that ends earliest). This requires more calls to the oracle but gives a faster run-time.

We want to find a shortest path from s to t that does not contain any path in X as a subpath—we make the goal more precise as follows. A *path* is a sequence of vertices each joined by an edge to the next vertex in the sequence. Note that we allow a path to visit vertices and edges more than once. If a path does not visit any vertex more than once, we explicitly call it a *simple path*. A simple directed path from vertex u to vertex v in G is called a *forbidden path* or an *exception* if a packet cannot follow the path from u to v because of the physical constraints. Given a set A of forbidden paths, a path $(v_1, v_2, v_3, \ldots, v_l)$ is said to *avoid* A if $(v_i, v_{i+1}, \ldots v_j) \notin A$ for all i, j such that $1 \leq i < j \leq l$. A path P from s to t is called a *shortest A-avoiding path* if the length of P is the shortest among all A-avoiding paths from s to t. We will use the term "exception avoiding" instead of "X-avoiding" when A is equal to X, the set of all forbidden paths in G.

We will use d to denote the largest degree of a vertex in G, k to denote the number of exceptions in X, and L to denote the total size of all exceptions in X.

Although only the graphs with *positive* edge-weights are relevant to the applications of the problem mentioned earlier, we will from now on use G to denote a more general graph—a

graph with no negative weight cycle. This is because, the "generic" algorithm we devise in this chapter applies to this bigger class of graphs. We will later adapt this generic algorithm for special classes of graphs, including those with positive edge-weights.

6.1.3 Relaxation

The process of *relaxation* [34, p. 585] forms the basis of many classical single-source shortest path algorithms, in particular (i) the Bellman-Ford algorithm for graphs with no negative weight cycle, (ii) Dijkstra's algorithm for graphs with positive edge-weights, and (iii) the one for directed acyclic graphs. The process starts with a temporary shortest path tree consisting of a single vertex s, and then gradually makes the tree bigger and changes the tree if necessary, until the tree converges to a shortest path tree rooted at s.

In more detail, the relaxation process maintains for every vertex $v \in V$ an *estimated distance from* s, denoted by $d(v)$, which is never less than the shortest s-v distance. The process also maintains for every vertex v a *predecessor* $p(v)$, which points to the parent of v in the current tree. The relaxation process is first *initialized* by setting $d(s) = 0$ and $d(v) = \infty$ for all $v \in V - \{s\}$, and pointing $p(v)$ to nil for all $v \in V$. The main step of the process is *relaxing an edge* (u, v), which consists of testing whether the current estimated path from s to v can be improved by going through u, and updating $d(v)$ and $p(v)$ accordingly. More precisely, if $d(v) > d(u) + w(u, v)$, then $d(v)$ is set to $d(u) + w(u, v)$ and $p(v)$ is set to point to u; otherwise $d(v)$ and $p(v)$ are kept unchanged. The process repeatedly relaxes the edges of the input graph until $d(v)$ is at most $d(u) + w(u, v)$ for every edge (u, v). (Note that this is possible if and only if the graph has no negative weight cycle.) At this point, $d(v)$ is the actual shortest distance from s to v for all v, and the sequence of predecessors originating at v runs backwards along a shortest s-v path. Clearly, the array of all the predecessors in the graph represents a shortest path tree rooted at s. The order in which the edges can be relaxed efficiently depends on the input graph, and the main differences between the classical algorithms for the three mentioned graph classes depend on this order.

Now consider a *partial shortest path tree* T' rooted at s, i.e., a tree T' rooted at s which is a subtree of a shortest path tree rooted at s. When we are given T', the relaxation process can "grow" T' into a shortest path tree in G if we use a slightly modified initialization step as follows. For each vertex $v \in T'$, we initialize $d(v)$ to the length of the path from s to v in T' (which is the shortest s-v distance), and we point $p(v)$ to the predecessor of v in T'. For each vertex $v \notin T'$, we set $d(v) = \infty$ and point $p(v)$ to nil.

Lemma 6.1. *After the above initialization step, if we relax the edges of G until we have $d(v) \leq d(u) + w(u, v)$ for every edge $(u, v) \in E$ that is reachable from s, we obtain a shortest path tree T rooted at s in G such that T' is a subgraph of T.*

Proof. Since the estimated distances of the vertices in T' does not change after our initialization step, the computed tree T has T' as a subgraph. Now consider a shortest path $(s = v_0, v_1, v_2, \ldots, v_l)$ in G_i. One of the properties of relaxation [34, p. 587] is that $d(v)$ is never less than the shortest distance from s to v for all $v \in V$. Since $d(s = v_0) = 0$, and $d(v_i) \leq d(v_{i-1}) + w(v_{i-1}, v_i)$ for all $i \in [1, k]$, it is easy to show using induction on i that $d(v_i)$ is the actual shortest distance for all i. □

6.1.4 Related work

A shortest s-t path in a directed acyclic graph can be computed in $O(n+m)$ time by relaxing the edges in a topological order of the first vertices (i.e., the "from" vertices) of the edges. A shortest s-t path can be computed in $O(n \log n + m)$ time in a graph with non-negative edge-weights using Dijkstra's algorithm with a Fibonacci heap, and in $O(nm)$ time in a graph with no negative weight cycle using the Bellman-Ford algorithm. All these three algorithms have linear space requirements, and are based on the relaxation process [34, Chapter 24].

We briefly mention other approaches to shortest paths (although we will not use them). When the edge-weights in a graph are non-negative integers, the shortest s-t path problem can be solved in deterministic $O(m \log\log n)$ time and linear space if the graph is directed [96], and in optimal $O(m)$ time if the graph is undirected [95]. In many of these cases, there are randomized algorithms with better expected run-times, as well as approximation schemes. See Zwick [103] for a survey of shortest path algorithms, and Cabello [23], Goldberg and Harrelson [49], Goldberg et al. [50] and Holzer et al. [59] for some of the more recent work.

Two recent papers on shortest paths in graphs address the issue of avoiding a set of forbidden paths, assuming that all the forbidden paths are known a priori. The first paper gives a hardness result. Szeider [94] shows, using a reduction from 3-SAT, that the problem of finding a shortest *simple* exception avoiding path is NP-complete even when each forbidden path has two edges. If the forbidden paths are *not* known a priori, the hardness result still applies to the case of simple paths because the lack of prior knowledge of the forbidden paths only makes the problem harder.

The second paper, by Villeneuve and Desaulniers [99], gives an algorithm for a shortest (possibly non-simple) exception avoiding path for the case when all the forbidden paths are known a priori. They preprocess the graph in $O((n+L)\log(n+L)+m+dL)$ time and $O(n+m+dL)$ space so that a shortest path from s to a query vertex can be found in $O(n+L)$ time. They first build a deterministic finite automaton (DFA) from the set of forbidden paths using the idea of Aho and Corasick [12], which can detect in linear time whether a given path contains any of the forbidden paths. They then "insert" the DFA into G by replicating certain vertices of G in the manner introduced by Martins [39], and then build a shortest path tree in this modified graph. Their algorithm cannot handle the case where the set of all forbidden paths is not explicitly given. Our algorithm is strictly more general.

We now mention two problems that seem related to ours, but do not in fact provide solutions to ours. The first one is maintaining shortest paths in a dynamic graph, i.e., where nodes or edges may fail [41, 43, 57], or edge-weights may change (e.g., [25, 41, 42]). Forbidden paths cannot be modeled by deleting edges or by modifying edge costs because *all* edges in a particular forbidden path may be essential—see Figure 6.1 for an example. The second seemingly related problem is finding the k shortest paths in a graph. This was the subject of Martins [39] who introduced the vertex replication technique that we use in our algorithm. There is considerable work on this problem, see Eppstein [45] for a brief survey. But the k shortest path problem is again different from our situation because a forbidden subpath may be a bottleneck that is present in all of the k shortest paths even for $k \in \Omega(2^{n/2})$, see Villeneuve and Desaulniers [99].

In the context of optical networks researchers have studied many theoretical problems. See Ramaswami and Sivarajan [79] for details on optical networks, and Lee and Shayman [66]

and McGregor and Shepherd [68] for a brief survey of the theoretical problems that have been investigated. In the previous work, the effect of physical constraints on paths in optical networks is either not considered at all (e.g., Khuller et al. [62]), or simply modeled by a known constant upper bound on the length of such a path (e.g., Gouveia et al [51], Lee and Shayman [66] and McGregor and Shepherd [68]). To the best of our knowledge, none of the previous work on shortest paths in optical networks considers the fact that it is practically infeasible to know a priori all the forbidden paths in the network, i.e., all the constraints in X. We handle the issue of physical constraints from a different and much more practical perspective.

6.2 A generic algorithm for a shortest s-t path

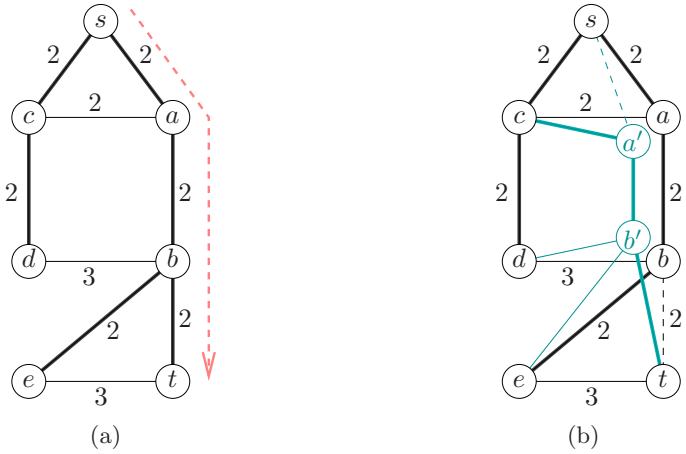


Figure 6.1: (a) Shortest paths and (b) shortest x-avoiding paths in a graph, where $x = (s, a, b, t)$.

In our algorithm we begin with a shortest path tree rooted at s, ignoring the exceptions. We then "try out" the path from s to t in the tree. If the path is free of exceptions, we are done. Otherwise, to take the newly discovered exception into account, we modify the graph using path replication as described in Section 6.1, and we modify the shortest path tree to match. In general, we maintain a modified graph and a shortest path tree in the graph that gives a shortest path in the original graph from s to every other vertex avoiding all the currently-known exceptions. We will first illustrate the idea with an example. Consider the graph G in Figure 6.1(a), where the integers denote edge-weights, and the dashed arrow marks the forbidden path $x = (s, a, b, t)$. Note that for simplicity we have used undirected edges in the figure to denote bidirectional edges. It is not hard to see that $P = (s, c, a, b, t)$ is the shortest x-avoiding path from s to t. To find P, we first construct a shortest path tree rooted at s (marked using the heavy edges in Figure 6.1(a)), and then try the path (s, a, b, t) in the tree. The path fails because it contains x, so we use a *vertex replication technique* similar to the one by Martins [39] to make duplicates of vertices a and b and delete edges (s, a') and (b, t), as shown in Figure 6.1(b). We then construct a shortest path tree rooted at s (marked using the heavy edges in Figure 6.1(b)) in the modified graph, and try the path (s, c, a', b', t) which "represents" the path P in G. We are done if x is the only forbidden path in G. Note that this approach can double the number of *undiscovered* forbidden paths. Suppose $y = (c, a, b)$

is another forbidden path in G. We have two copies of y in the modified graph: (c, a, b) and (c, a', b'), and we have to avoid both of them. Our solution to this doubling problem is to "grow" the shortest path tree in such a way that at most one of these two copies is encountered in future. Our algorithm is as follows:

```
1  construct the shortest path tree T_0 rooted at s in G_0 = G;
2  let i = 1;
3  send a packet from s to t through the path in T_0;
4  while the packet fails to reach t do
5      let x_i be the exception that caused the failure;
6      construct G_i from G_{i-1} by replicating the intermediate vertices of x_i and then
       deleting selected edges;
7      construct the shortest path tree T_i rooted at s in G_i using T_{i-1};
8      send a packet from s to t through the path in T_i;
9      let i = i + 1;
10 end
```

In the above algorithm, the only lines that need further discussion are Lines 6 and 7; details are in Sections 6.2.1 and 6.2.2 respectively. In the rest of the chapter, whenever we focus on a particular iteration $i > 0$, we use the following notation:

- the path from s to t in T_{i-1}, i.e., the path along which we try to send the packet to t in Line 4 in the iteration, is $(s, v_1, v_2, \ldots, v_p, t)$, and
- the exception that prevented the packet from reaching t in the iteration is $x_i = (v_{r-l}, v_{r-l+1}, \ldots, v_r, v_{r+1})$, which consists of $l + 1$ edges.

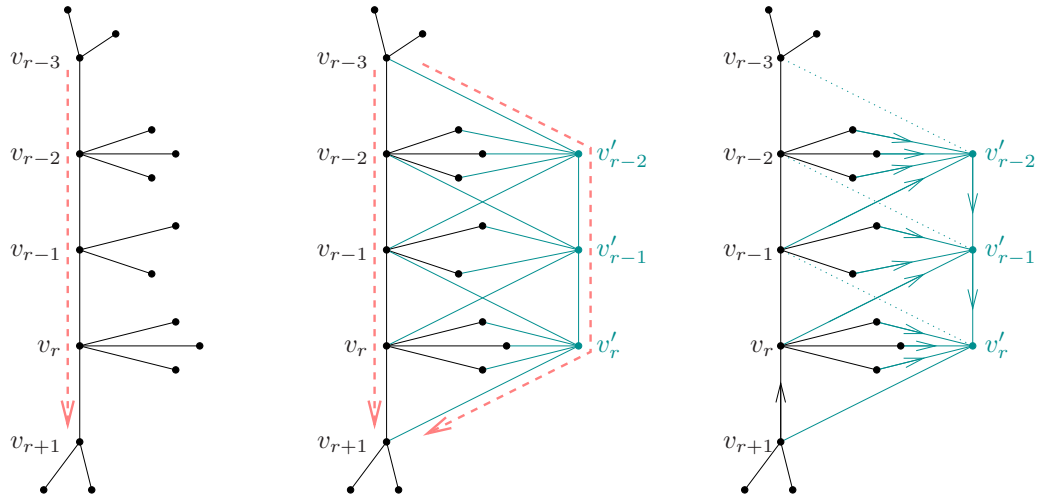
6.2.1 Modifying the graph

The modification of G_{i-1} into G_i (Line 6) in the ith iteration eliminates exception x_i while preserving all the x_i-avoiding paths in G_{i-1}. We do the modification in two steps.

In the first step, we create a graph G'_{i-1} by replicating the intermediate vertices of x_i (i.e., the vertices $v_{r-l+1}, v_{r-l+2}, \ldots, v_r$). We also add appropriate edges to the replica v' of a vertex v. Specifically, when we add v' to G_{i-1}, we also add the edges of appropriate weights between v' and the neighbors of v. It is easy to see that if a path in G_{i-1} uses $l' \leq l$ intermediate vertices of x_i, then there are exactly $2^{l'}$ copies of the path in G'_{i-1}. We say that a path in G'_{i-1} is x_i-avoiding if it contains none of the 2^l copies of x_i.

In the second step, we build a spanning subgraph G_i of G'_{i-1} by deleting a few edges from G'_{i-1} in such a way that all copies of x_i in G'_{i-1} are eliminated, but all x_i-avoiding paths in G'_{i-1} remain unchanged. To build G_i from G'_{i-1}, we delete the edges (v_{j-1}, v'_j) and (v'_j, v_{j-1}) for all $j \in [r-l+1, r]$. We also delete the edge (v_r, v_{r+1}), all the outgoing edges from v'_r except (v'_r, v_{r+1}), and all the outgoing edges from v'_j except (v'_j, v'_{j+1}) for all $j \in [r-l+1, r-1]$. Figure 6.2 shows how the "neighborhood" of an exception changes from G_{i-1} to G_i. As before, the undirected edges in the figure are bidirectional.

Observation 6.1. *Graph G_i has no copy of x_i.*



(a) G_{i-1} (b) G'_{i-1} (c) G_i

Figure 6.2: Modifying G_{i-1} to G_i: (a) The part of G_{i-1} at an exception $x_i = (v_{r-3}, v_{r-2}, v_{r-1}, v_r, v_{r+1})$, with $l = 3$. (b) Replicating vertices to create G'_{i-1}. The dashed paths show two of the 8 copies of the exception. (c) Deleting edges to create G_i. The dotted lines denote deleted edges. Note that the edge (v_r, v_{r+1}) has been deleted, but (v_{r+1}, v_r) remains.

Observation 6.2. *(i) Graph G_i has no negative weight cycle.*

(ii) If the edge-weights in G_{i-1} are non-negative, so are the edge-weights in G_i.

(iii) If G_{i-1} is a directed acyclic graph, so is G_i. Moreover, by inserting the sequence $(v'_{r-l+1}, v'_{r-l+2}, \ldots, v'_r)$ right before v_r in a topological order of the vertices in G_{i-1}, we get a topological order of the vertices in G_i.

In Section 6.3.1 we will prove that G_i still contains all the x_i-avoiding paths of G_{i-1}.

The vertices in G_i $[G'_{i-1}]$ that exist also in G_{i-1} (i.e., the ones that are not replica vertices) are called the *old vertices of G_i [respectively G'_{i-1}]*. Note that the vertices of G_0 exist in G_i for all $i \geq 0$. These vertices are called the *original vertices of G_i*.

6.2.2 Constructing the tree

In Line 7 of our algorithm we construct a tree T_i that contains a shortest x_i-avoiding path from s to every other vertex in G_{i-1}. Tree T_i is rooted at s, and its edges are directed away from s. Not every shortest path tree rooted at s in G_i will work. In order to guarantee termination of the algorithm, T_i must be similar to T_{i-1}, specifically, every x_i-avoiding path from s in T_{i-1} must be present in T_i. The necessity of this restriction is explained in Section 6.3.3.

We construct the required T_i by preserving as much of T_{i-1} as possible, and then applying the relaxation process (discussed in Section 6.1.2) starting from the preserved part of T_{i-1}. We have to modify the initialization step of the standard relaxation process in the manner used in Lemma 6.1. To be precise, let W_i be the set of vertices that are either replica vertices in G_i

or descendants of v_{r+1} in T_{i-1}, and let U_i be the set of all other vertices in G_i. The part of T_{i-1} that we preserve spans U_i. Our initialization step temporarily sets $T_i = T_{i-1} - W_i$, and then sets the estimated distance $d(v)$ to infinity for all $v \in W_i$. For all $v \in U_i$, $d(v)$ remains unchanged from the previous iteration (i.e., $d(v)$ denotes the s-v distance in the preserved part of T_{i-1}). After initialization, we run the main iterative part of the standard relaxation process: we repeatedly relax the edges until $d(v) \leq d(u) + w(u, v)$ is true for every edge (u, v). The process terminates because G_i has no negative weight cycle (Observation 6.2(i)). It makes the temporary T_i "grow" into a spanning tree in G_i. The order in which the edges are relaxed depends on the class of G_i (i.e., whether G_i has a cycle and/or a negative edge-weight), and we follow the order used in one of the classical single-source shortest path algorithms. We will discuss this in detail in Section 6.4.

6.3 Correctness and analysis

6.3.1 Justifying the graph modification

In this section we prove the following lemma, which uses the notion of a *corresponding path*. Consider any path P_i in G_i. By substituting every vertex in P_i that is not present in G_{i-1} with the corresponding old vertex in G_{i-1}, we get the corresponding path P_{i-1} in G_{i-1}. This is possible because any "new" edge in G_i is a replica of an edge in G_{i-1}. We define the corresponding path P_j in G_j for all $j < i$ by repeating this argument.

Lemma. *If P_i is a shortest path from s to an original vertex v in G_i, P_0 is a shortest $\{x_1, x_2, \ldots, x_i\}$-avoiding path from s to v in G_0.*

To prove the above lemma (repeated as Lemma 6.4 below), we will first prove that x_i-avoiding paths in G_{i-1} are preserved in G_i (Lemma 6.3), using the following characteristic of an x_i-avoiding path in the intermediate graph G'_{i-1}:

Lemma 6.2. *For any x_i-avoiding path P from s to v that uses only the old vertices in G'_{i-1}, there exists a copy of P in G_i that starts and ends at the old vertices s and v respectively, and possibly passes through the corresponding replicas of its intermediate vertices.*

Proof. We prove this lemma by re-routing any portion of P that uses the directed edge (v_r, v_{r+1}) to use the replica edge (v'_r, v_{r+1}) instead.

Graph G_i contains all the edges between pairs of old vertices in G'_{i-1} except for the directed edge (v_r, v_{r+1}). Thus P can remain unchanged if it does not use this directed edge. Otherwise we will re-route any portion of P that uses the directed edge (v_r, v_{r+1}) to use the replica edge (v'_r, v_{r+1}) instead. Let $P = (s = w_1, w_2, \ldots, w_q = v)$, and (w_j, w_{j+1}) be an occurrence of (v_r, v_{r+1}) in P. Tracing P backwards from w_j, let $h \leq j$ be the minimum index such that $(w_h, w_{h+1}, \ldots, w_{j+1})$ is a subpath of x_i. Because P is x_i-avoiding, w_h must be an intermediate vertex of x_i. This implies that $h > 1$, since $s = w_1$ is not an intermediate vertex of x_i because of the following reasons: (i) x_i is a path in the shortest path tree rooted at s in G_{i-1}, and (ii) there is no replica of s in G_i. Therefore w_{h-1} exists. We will reroute the portion of P between w_{h-1} and w_{j+1} by using the corresponding replica vertices in place of the subpath $(w_h, \ldots, w_j)$

of x_i. Note that the required edges exist in G_i (since P does not contain the whole exception x_i), and that the portions of P that we re-route are disjoint along P. Moreover, P starts and ends at the old vertices s and v respectively. □

Lemma 6.3. *Any x_i-avoiding path from s to v in G_{i-1} has a copy in G_i that starts and ends at the old vertices s and v respectively, and possibly goes through the corresponding replicas of its intermediate vertices.*

Proof. Let P be the x_i-avoiding path in G_{i-1}. As we do not delete any edge to construct G'_{i-1} from G_{i-1}, P remains unchanged in G'_{i-1}. Moreover, P uses no replica vertex in G'_{i-1}. So, Lemma 6.2 implies that P exists in G_i with the same old vertices at the endpoints, possibly going through the corresponding replicas of the intermediate vertices. □

Lemma 6.3 along with Observation 6.1 implies that we can find a shortest x_i-avoiding path in G_{i-1} using a shortest path in G_i. The main lemma of this section proves a stronger version of this claim that covers all the modifications done from G_0 to G_i:

Lemma 6.4. *If P_i is a shortest path from s to an original vertex v in G_i, P_0 is a shortest $\{x_1, x_2, \ldots, x_i\}$-avoiding path from s to v in G_0.*

Proof. For any $j \in [0, i]$, let $X_j = \{x_{j+1}, x_{j+2}, \ldots, x_i\}$. We show that for any j, if P_j is a shortest X_j-avoiding path in G_j, then P_{j-1} is a shortest X_{j-1}-avoiding path in G_{j-1}. The lemma then follows by induction on j, with basis $j = i$, because $X_i = \emptyset$ and thus P_i is a shortest X_i-avoiding path in G_i.

If P_j is a shortest X_j-avoiding path in G_j, P_j is X_{j-1}-avoiding because P_j is x_j-avoiding by Observation 6.1, and $X_j \cup \{x_j\} = X_{j-1}$. So, the corresponding path P_{j-1} is also X_{j-1}-avoiding. If we assume by contradiction that P_{j-1} is not a shortest X_{j-1}-avoiding path in G_{j-1}, then there exists another path P'_{j-1} from s to v in G_{j-1} which is X_{j-1}-avoiding and is shorter than P_{j-1}. Since $x_j \in X_{j-1}$, P'_{j-1} is x_j-avoiding, and hence by Lemma 6.3, there is a copy P'_j of path P'_{j-1} in G_j which has the same original vertices at the endpoints. As P'_{j-1} is X_j-avoiding, P'_j is also X_j-avoiding. This is impossible because P'_j is shorter than P_j. Therefore, P_{j-1} is a shortest X_{j-1}-avoiding path in G_{j-1}. □

6.3.2 Justifying the tree construction

To show that the "incremental" approach used in Section 6.2.2 to construct T_i is correct, we first show that the part of T_{i-1} that we keep unchanged in T_i is composed of shortest paths in G_i:

Lemma 6.5. *For every vertex $v \in U_i$, the path P from s to v in T_{i-1} is a shortest path in G_i.*

Proof. First we show that P exists in G_i. Every vertex in T_{i-1} exists in G_i as an old vertex. So, P exists in G_i through the old vertices if no edge of P gets deleted in G_i. The only edge between a pair of old vertices in G_{i-1} that gets deleted in G_i is (v_r, v_{r+1}). Since v is not a descendant of v_{r+1} in T_{i-1}, P does not use the edge (v_r, v_{r+1}). Therefore, no edge of P gets deleted in G_i. So, P exists in G_i through the old vertices.

Neither the modification from G_{i-1} to G'_{i-1} nor the one from G'_{i-1} to G_i creates any "shortcut" between any pair of vertices. So, there is no way that the distance between a pair of old vertices decreases after these modifications. Since these modifications do not change P, which is a shortest path in G_{i-1}, P is a shortest path in G_i. □

Lemma 6.6. *The tree T_i is a shortest path tree in G_i.*

Proof. For every vertex $v \in U_i$, the path P from s to v in T_i is the same as the one in T_{i-1} and hence, a shortest path in G_i (Lemma 6.5). The lemma then follows from Lemma 6.1. □

Lemmas 6.4 and 6.6 together prove that our algorithm is correct provided it terminates, which we establish in the next section.

6.3.3 Analysis of timing

Although in every iteration we eliminate one exception by modifying the graph, we introduce copies of certain other exceptions through vertex replication. Still our algorithm does not iterate indefinitely because, as we will show in this section, the incremental construction of the shortest path tree (Section 6.2.2) guarantees that we do not discover more than one copy of any exception. We first show that any exception in G_{i-1} has at most two copies in G_i (Lemma 6.7), and then prove that one of these two copies is never discovered in the future (Lemma 6.8):

Lemma 6.7. *Let $y \neq x_i$ be any exception in G_{i-1}. If the last vertex of y is* not *an intermediate vertex of x_i, then G_i contains exactly one copy of y. Otherwise, G_i contains exactly two copies of y. In the latter case, one copy of y in G_i ends at the old vertex v and the other copy ends at the corresponding replica v'.*

Proof. Let $\pi = (w_1, w_2, \ldots, w_j)$ be a maximal sequence of vertices in y that is a subsequence of $(v_{r-l+1}, v_{r-l+2}, \ldots, v_r)$. Let w'_j be the replica of w_j in G_i. We will first show that if there is a vertex v in y right after π, then exactly one of the edges (w_j, v) and (w'_j, v) exists in G_i. Consider the subgraph of G_i induced on the set of replica vertices $\{v'_{r-l+1}, v'_{r-l+2}, \ldots, v'_r\}$: this subgraph is a directed path from v'_{r-l+1} to v'_r, and the only edge that goes out of this subgraph is (v'_r, v_{r+1}). Therefore, (i) when $(w_j, v) = (v_r, v_{r+1})$, $(w'_j, v) \in G_i$ and $(w_j, v) \notin G_i$, and (ii) otherwise, $(w_j, v) \in G_i$ and $(w'_j, v) \notin G_i$.

Now G_i has exactly two copies of π: one through the old vertices, and another through the replicas. The above claim implies that when there is a vertex v in y right after π, G_i has at most one copy of the part of y from w_1 to v. However, when π is a suffix of y, G_i has both the copies of the part of y from w_1 to w_j. The lemma then follows because any part of y that contains no intermediate vertex of x_i has exactly one copy in G_i. □

Lemma 6.8. *Let $y \neq x_i$ be any exception in G_{i-1} such that the last vertex of y is an intermediate vertex v of x_i. The copy of y that ends at the old vertex v in G_i is not discovered by the algorithm in any future iteration.*

Proof. The copy of the path $(s, v_1, v_2, \ldots, v_r)$ through the old vertices in G_i contains v. Let P be the part of this path from s to v. Clearly, $P \in T_{i-1}$, and P does not contain any exception because the oracle returns the exception with the earlier last vertex. So, the way we construct T_j from T_{j-1} for any iteration $j \geq i$ ensures that $P \in T_j$.

Let y_1 be the copy of y that ends at v. Now y_1 is not a subpath of P because P does not contain any exception. For any $j \geq i$, $P \in T_j$, and both P and y_1 end at the same vertex, therefore $y_1 \notin T_j$. So, a packet in iteration j will not follow y_1, and y_1 will not be discovered in that iteration. □

Lemma 6.9. *The **while** loop iterates at most $k = |X|$ times.*

Proof. For any iteration i, G_{i-1} contains x_i, and G_i does not contain x_i. Every exception other than x_i in G_{i-1} has either one or two copies in G_i (Lemma 6.7). By Lemma 6.8, if an exception has two copies in G_i, only one of them is relevant in the future. Thus the number of exceptions effectively decreases by one in each iteration. The lemma then follows. □

6.3.4 Relaxing the edges efficiently

It follows from Lemma 6.9 that our algorithm terminates in polynomial time provided that in every iteration $i \in [1, k]$, we relax the edges of G_i a polynomial number of times. Although the exact number depends on the graph class of G_i (i.e., whether G_i has a cycle and/or a negative edge-weight), this section makes the relaxation process efficient for all graphs we consider.

It is obvious that each iteration of our algorithm increases both the number of vertices and the number of edges, which implies that the time required for iteration i will increase with i. However, the shortest paths that are already known in the ith iteration can be utilized in the following way to suppress the increase in time-per-iteration.

We relax the edges in G_i in such an order that the number of times we relax the edges depends only on n and m even though the size of G_i is a function of n, m, d and the total size of the exceptions discovered so far. (G_i can have as many as $n + L$ vertices and as many as $m + dL$ edges, as we will show in the proof of Lemma 6.10.) We achieve this by relaxing the edges of G_i in the following two phases. The first phase, which is independent of the graph class of G_i, relaxes all the edges (u, v) such that $u \in U_i$ and $v \in W_i$. The second phase relaxes the edges connecting two vertices in W_i, by running a classical single-source shortest path algorithm on the subgraph of G_i induced on W_i. Observe that a shortest path from s to any vertex $v \in W_i$ consists of a sequence of zero or more edges connecting two vertices in U_i, followed by an edge connecting a vertex in U_i to a vertex in W_i, and finally followed by a sequence of zero or more edges connecting two vertices in W_i. The edges in the first sequence do not need to be relaxed because $d(u)$ is the already actual shortest distance for all $u \in U_i$ (Lemma 6.5). So we can relax the edges in the above two phases without compromising the correctness of our algorithm.

The above technique ensures that the time requirement for each iteration i of the **while** loop, and in particular the second phase of relaxation in iteration i, is independent of i even though the graph gets bigger with i. Let $T(n, m)$ and $S(n, m)$ be respectively the time and the *extra* space used in the second phase of relaxation in every iteration. (We will soon compute $T(n, m)$ and $S(n, m)$ for specific classes of input graphs.)

Lemma 6.10. *We can compute a shortest X-avoiding path from s to t in $O(k(n+m+T(n,m)))$ time and $O(n+m+dL+S(n,m))$ space.*

Proof. The correctness of the algorithm follows from Lemmas 6.4 and 6.6.

Let l_i be the number of intermediate vertices of x_i—the exception discovered at the ith iteration (thus the size of the exception is l_i+2). We first analyze space. The ith iteration constructs G_i by adding l_i vertices and at most dl_i edges to G_{i-1}. Since the algorithm iterates at most k times (Lemma 6.9), there are $n+\sum_{i=1}^{k} l_i < n+L$ vertices and at most $m+\sum_{i=1}^{k} dl_i < m+dL$ edges in G_i. So, any intermediate graph and the associated estimated distances takes $O(n+m+dL)$ space. The space requirement then follows.

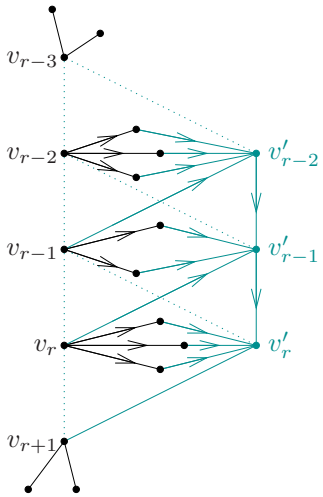


Figure 6.3: The graph in Figure 6.2(c) without the incoming edges of the intermediate vertices of x_i.

Before we analyze the time requirement, we first claim that in the ith iteration, each phase of relaxation considers at most m edges of G_i, although G_i can have as many as $m+dL$ edges. Observe that none of the phases relaxes the incoming edges of the vertices in U_i, and that all the intermediate vertices of x_i are in U_i. If we (temporarily) omit the incoming edges of the intermediate vertices of x_i as shown in Figure 6.3, the number of *new* edges in G_i becomes less than the number of those edges that are either deleted from G_{i-1} or omitted in this figure. The claim then follows.

Now, in the ith iteration the construction of G_i takes $O(l_i+m) = O(n+m)$ time, and initializing the relaxation process takes $O(n)$ time (since $|W_i| \le n$). The above claim together with the inequality $|W_i| \le n$ implies that the first phase of relaxation takes $O(n+m)$ time. So every iteration takes $O(n+m+T(n,m))$ time. The total time requirement then follows from Lemma 6.9. □

6.4 Algorithms for specific graph classes

The only part of our algorithm that requires further discussion is the second phase of relaxation in each iteration of the main loop. As we have mentioned before, we use a classical single-source

shortest path algorithm, although in a slightly modified form. Our choice of single-source shortest path algorithm depends on the graph class of G (i.e., whether G has a cycle and/or a negative edge-weight), as follows. Note that our graph modification preserves the graph class (Observation 6.2).

(i) When G is a directed acyclic graph, we relax the outgoing edges of each vertex while iterating through the vertices in a topological order [34, Section 24.2].

(ii) When the edge-weights in G are non-negative, we use Dijkstra's algorithm with a Fibonacci heap [34, Section 24.3].

(iii) When negative edge-weights are present, we use the Bellman-Ford algorithm [34, Section 24.1].

Each of these classical algorithms starts with the standard initialization step of the relaxation process. Recall that the only task performed in this step is to set the estimated distance of vertex v to zero if v is the source, and to infinity otherwise. The only modification we make to the classical algorithm is that we omit this initialization step. In each iteration, our modified initialization step of relaxation (discussed in Section 6.2.2) followed by the first phase of relaxation (discussed in Section 6.3.4) initializes the estimated distances in a way appropriate for our purpose. Then the second phase of relaxation is performed by using one of these algorithms without the standard initialization step.

This modification raises a correctness issue. The correctness of our generic algorithm in Section 6.2 relies on the condition that at the end of the second phase relaxation in each iteration i, we have $d(v) \leq d(u) + w(u, v)$ for every edge (u, v) that is reachable from s in G_i. None of the three classical shortest path algorithms *explicitly* checks before termination whether the condition is true, but the condition holds at termination in each case. It is not clear whether this is the case even after our modified initialization step. We will prove that the condition is still satisfied in our algorithms.

Theorem 6.1. *When G is a directed acyclic graph, we can compute a shortest X-avoiding path from s to t in $O(k(n+m))$ time and $O(n+m+dL)$ space, where d is the largest degree of a vertex in G, k is the number of exceptions in X, and L is the total size of all exceptions in X.*

Proof. We first prove that at the end of the second phase relaxation in each iteration i, we have $d(v) \leq d(u) + w(u, v)$ for any edge (u, v) that is reachable from s in G_i. Since in the second phase we relax the outgoing edges of each vertex while iterating through the vertices in a topological order, we relax the edge (u, v) after relaxing all the incoming edges of u. This implies that $d(u)$ cannot change after we relax (u, v). Because $d(v) \leq d(u) + w(u, v)$ at this point of time, and relaxation never increases $d(v)$, the inequality remains true till the end of the second phase.

The theorem then follows from Lemma 6.10, as we have $T(n, m) = O(n+m)$ and $S(n, m) = O(1)$ in this case [34, Section 24.2]. □

Theorem 6.2. *When G is a graph with non-negative edge-weights, we can compute a shortest X-avoiding path from s to t in $O(kn \log n + km)$ time and $O(n+m+dL)$ space, where d is*

the largest degree of a vertex in G, k is the number of exceptions in X, and L is the total size of all exceptions in X.

Proof. As in the case of the proof of Theorem 6.1, we first show that at the end of the second phase relaxation in each iteration i, we have $d(v) \leq d(u) + w(u, v)$ for any edge (u, v) that is reachable from s in G_i. We use Dijkstra's algorithm, where we relax the edge (u, v) after removing (dequeuing) u from the priority queue. Because any vertex u' dequeued after u has $d(u') \geq d(u)$, estimated distance $d(u)$ remains unchanged after we dequeue u. Relaxing (u, v) makes $d(v) \leq d(u) + w(u, v)$, and the inequality remains true till the end of the second phase since relaxation never increases $d(v)$.

The theorem then follows from Lemma 6.10, as we have $T(n, m) = O(n \log n + m)$ and $S(n, m) = O(n)$ in this case [34, Section 24.3]. □

Theorem 6.3. *When G contains negative edge-weights but no negative weight cycle, we can compute a shortest X-avoiding path from s to t in $O(knm)$ time and $O(n + m + dL)$ space, where d is the largest degree of a vertex in G, k is the number of exceptions in X, and L is the total size of all exceptions in X.*

Proof. We first prove that at the end of the second phase relaxation in each iteration i, $d(v)$ is the shortest distance from s to v for each vertex v in G_i. This is true for all vertices in U_i by Lemma 6.5. For every $v \in W_i$ that is reachable from s in G_i, there is a shortest path $(u_0 = s, u_1, u_2, \ldots, u_a, w_1, w_2, \ldots, w_b = v)$ such that $u_j \in U_i$ for all $j \in [0, a]$ and $w_j \in W_i$ for all $j \in [1, b]$. The first phase of relaxation sets $d(w_1)$ to the shortest s-w_1 distance. In the second phase, every iteration of the outer loop in the Bellman-Ford algorithm relaxes *every* edge between the vertices in W_i. Clearly, the first $b - 1$ iterations of this loop relax the edges of the path $(w_1, w_2, \ldots, w_b)$ in the order from (w_1, w_2) to (w_{b-1}, w_b). Therefore, $d(w_b = v)$ is set to the shortest s-v distance at the end of the $(b - 1)$th iteration of this loop.

It then follows from the triangle inequality that $d(v) \leq d(u) + w(u, v)$ for any edge (u, v) of G_i. The theorem then follows from Lemma 6.10, as we have $T(n, m) = O(nm)$ and $S(n, m) = O(1)$ in this case [34, Section 24.1]. □

Interestingly, when X is known explicitly, we can solve the shortest exception avoiding path problem by modifying our generic algorithm in a straightforward manner as follows. We can make a deterministic finite automaton (DFA) from the set of forbidden paths so that we can determine in linear time whether a path contains an exception or not. Using this approach, we solve this case in $O(dL + k(n + m + L + T(n, m)))$ preprocessing time. Recall that Villeneuve and Desaulniers [99] solved this case in $O(dL + T(n + L, m + dL))$ preprocessing time, which is faster in general. There are, however, a few very restricted settings in which our approach is better. One such case is: $k = O(1)$, $L = \Theta(n)$ and $m = o(dn)$ (intuitively, when there are a few long exceptions, and the average degree of a vertex is much smaller than the largest degree).

6.5 Computing shortest paths to all vertices

The generic algorithm in Section 6.2 can be extended to compute a shortest path from s to every other vertex in G. Of course we can run the generic algorithm once with every vertex

in G as a destination, but we are then likely to encounter some of the exceptions multiple times. We avoid this repetitive discovery of exceptions by saving some information in between the iterations as follows: for every destination vertex (except of course the first one), we start with the graph and the shortest path tree constructed for the previous destination vertex. The algorithm is as follows:

```
1  construct the shortest path tree T_0 rooted at s in G_0 = G;
2  let i = 1;
3  for each vertex v ∈ V do
4      send a packet from s to v through the path in T_{i-1};
5      while the packet fails to reach v do
6          let x_i be the exception that caused the failure;
7          construct G_i from G_{i-1} by replicating the intermediate vertices of x_i and then
           deleting selected edges;
8          construct the shortest path tree T_i rooted at s in G_i using T_{i-1};
9          send a packet from s to v through the path in T_i;
10         let i = i + 1;
11     end
12 end
```

Lemma 6.11. *We can preprocess the graph in $O(k(n + m + T(n, m)))$ time and $O(n + m + dL + S(n, m))$ space so that we can find a shortest X-avoiding path from s to any vertex in $O(n + L)$ time.*

Proof. Since every exception in X is handled at most once, the **while** loop still iterates at most k times, and therefore, the time and space requirement follows from the proof of Lemma 6.10. □

We then have the following theorems for the three classes of graphs:

Theorem 6.4. *When G is a directed acyclic graph, we can preprocess the graph in $O(k(n+m))$ time and $O(n + m + dL)$ space so that we can find a shortest X-avoiding path from s to any vertex in $O(n + L)$ time.*

Theorem 6.5. *When G is a graph with non-negative edge-weights, we can preprocess the graph in $O(kn \log n + km)$ time and $O(n + m + dL)$ space so that we can find a shortest X-avoiding path from s to any vertex in $O(n + L)$ time.*

Theorem 6.6. *When G contains negative edge-weights but no negative weight cycle, we can preprocess the graph in $O(knm)$ time and $O(n + m + dL)$ space so that we can find a shortest X-avoiding path from s to any vertex in $O(n + L)$ time.*

6.6 Handling a weaker oracle

The oracle we considered so far returns an exception that ends earliest in the query path. We now consider a weaker oracle that instead returns *any* exception on the query path. The

easiest way to handle this weaker oracle is to modify the algorithm in Section 6.2 as follows: whenever the path P from s to t in the current shortest path tree fails (Line 4), we repeatedly query the weaker oracle with appropriate subpaths of P to locate an exception that ends earliest (intuitively, we "compensate" for the weakness of the oracle by querying the oracle more often). More precisely, we perform a binary search on the length of a prefix of P to find the smallest prefix that contains an exception. Any exception in the smallest prefix is then used as exception x_i in Line 5 of our algorithm.

Lemma 6.12. *We can preprocess the graph in $O(k(n+m+\log L+T(n,m)))$ time and $O(n+m+dL+S(n,m))$ space so that we can find a shortest X-avoiding path from s to any vertex in $O(n+L)$ time.*

Proof. The only difference between the algorithm in Section 6.5 and the current algorithm is that in each iteration, locating exception x_i in the path P from s to t in T_{i-1} takes $O(1)$ time in the previous algorithm, and $O(\log(n+L))$ time in the current algorithm (since P can have at most $n+L$ vertices). So, we use $O(k\log(n+L))$ more time in preprocessing in the current algorithm. The lemma then follows Lemma 6.11. □

We thus have the following running time for graphs with negative edge-weights:

Theorem 6.7. *When G contains negative edge-weights but no negative weight cycle, we can preprocess the graph in $O(k(nm+\log L))$ time and $O(n+m+dL)$ space so that we can find a shortest X-avoiding path from s to any vertex in $O(n+L)$ time.*

For each of the other two classes of graphs, i.e., directed acyclic graphs and graphs with non-negative edge-weights, we can do better than the generic algorithm. We instead have a "specialized" algorithm that is faster than the one derived from the above generic algorithm. In each case, the intuition is that we "merge" the **while** loop of our generic algorithm with the outer loop of the classical single-source shortest path algorithm. This is possible because the classical algorithm starts with an empty shortest path tree and then adds one vertex to the current shortest path tree in each iteration of the outer loop. We effectively stop the growth of a tree containing forbidden paths by trying the path (i.e., querying the oracle with the path) from s to the vertex being added to the tree in the beginning of each iteration. We will discuss each algorithm below.

For directed acyclic graphs, we modify the classical algorithm slightly as follows. Inside the outer loop (i.e., the loop that considers the vertices in a topological order), we first try the path from s to the current vertex v in the current shortest path tree. If the path is exception avoiding, we relax the outgoing edges of v as in the classical algorithm. Otherwise, we first modify the graph using the exception returned by the oracle, as described in Section 6.2.1. We then relax all the incoming edges of v and the replica vertices, and update the topological order used in the outer loop to include the replica vertices. Finally we continue the outer loop from the first replica vertex in the topological order.

Theorem 6.8. *When G is a directed acyclic graph, we can preprocess the graph in $O(n+m+dL))$ time and $O(n+m+dL)$ space so that we can find a shortest X-avoiding path from s to any vertex in $O(n+L)$ time.*

Proof. Let $(x_1, x_2, \ldots)$ be the sequence of exceptions discovered by the algorithm. To be consistent with the notation in Section 6.2, let $G_0 = G$, and G_i be the modified graph after the discovery of the ith exception. The algorithm above maintains a partial shortest path tree rooted at s, but does not define T_i. We define T_i be a shortest path tree in G_i that contains the current partial shortest path tree maintained by the algorithm.

The same argument as in the proof of Lemma 6.9 implies that the algorithms discovers at most k exceptions. By Observation 6.2(iii), the outer loop iterates in a topological order of the vertices in the current graph. The correctness of the algorithm then follows from that of the classical algorithm.

Let l_i be the number of intermediate vertices of exception x_i. For each i, there are $n + \sum_{i=1}^{k} l_i < n + L$ vertices and at most $m + \sum_{i=1}^{k} dl_i < m + dL$ edges in G_i. So, any intermediate graph and the associated estimated distances takes $O(n + m + dL)$ space. The outer loop iterates at most $n + \sum_{i=1}^{k}(l_i + 1) < n + 2L$ times, and the number of times we relax the edges is at most the number of edges, which is less than $m + dL$. The preprocessing time then follows. □

For graphs with non-negative edge-weights, we modify Dijkstra's algorithm in a similar manner: inside the outer loop (i.e., the loop that removes vertices from the priority queue one by one), we first try the path from s to the current vertex v in the current shortest path tree, and then relax the outgoing edges of v as in the classical algorithm if the path is exception avoiding. Otherwise, we first modify the graph using the exception returned by the oracle as before. We then relax all the incoming edges of v and the replica vertices, and insert v and the replica vertices into the priority queue. Finally we continue the outer loop.

Theorem 6.9. *When G is a graph with non-negative edge-weights, we can preprocess the graph in $O((n+L)\log(n+L)+m+dL)$ time and $O(n+m+dL)$ space so that we can find a shortest X-avoiding path from s to any vertex in $O(n+L)$ time.*

Proof. The proof is similar to that of Theorem 6.8 except that the correctness follows from Dijkstra's algorithm, and each iteration of the outer loop takes $O(\log(n + L))$ extra time for queue operations. □

Each of the specialized algorithms in Theorems 6.8 and 6.9 is faster than the corresponding generic algorithm (Lemma 6.12). However, the specialized algorithm makes as many as $n + L$ queries to the oracle versus at most k oracle queries for the generic one. The generic algorithm is faster only in the special case $k = O(1)$.

It is straightforward to extend the above algorithms to handle an even weaker oracle that does not reveal the location of an exception, but instead returns a boolean value indicating whether the query path is exception avoiding or not.

6.7 Conclusion

In this chapter we have presented an algorithm to compute shortest exception avoiding paths in the model where exceptions are not known a priori. Our algorithm is generic and depends on a

(conventional) shortest path algorithm, such as Dijkstra's algorithm in the case of non-negative edge-weights. The algorithm handles forbidden paths by coupling the vertex replication technique with the "growth" of a shortest path tree in the graph. We have adapted the generic algorithm to three classes of graphs, and two types of oracles that reveal different amount of information when queried about the feasibility of a path.

Our work in this chapter presents four avenues for future research.

We modeled forbidden paths by means of an oracle that—when queried—informs the algorithm of relevant forbidden paths. This is appropriate for the optical network routing problem, but the traffic routing application discussed in Section 6.1.1 seems to call for a somewhat different model. One can imagine the dispatcher, in a more passive role, receiving updates about road closures and re-openings. In this situation, the set of forbidden paths is a dynamic set, and the algorithm is given updates when a path is added to, or deleted from, the forbidden path set. This is an interesting problem for which our techniques offer nothing beyond the naive approach of re-starting the algorithm from scratch at most updates.

When the forbidden paths in a road network are static but unknown, our approach applies only to the restricted case that vehicles are dispatched repeatedly from a service provider. The more natural problem of finding a shortest path for a vehicle in such a road network seems significantly harder than finding such a path for a network packet. This is because upon discovering a forbidden path, a vehicle needs to find a detour *from the point of failure* (rather than *from the source* as in the case of a network packet). It is easy for an adversary to "fool" the vehicle into entering a part of the network from which the destination is unreachable, and it is not clear to us how to handle this issue.

Another interesting avenue is to explore other "traditional" optimization problems in the presence of forbidden paths. Two examples are the all-pairs shortest path problem and the bottleneck shortest path problem. One could also extend the notion of a spanning tree by asking for the minimum cost subset of edges that includes a path (avoiding forbidden paths) between any two vertices. Or one might ask for minimum weight strongly connected or biconnected subgraphs.

We mention one final extension. Rather than forbidding some paths, a more general approach is to *penalize* some paths. More precisely, given a set of penalized paths, each having a positive weight (a *penalty*) in addition to the total weight of its edges, define the weight of a path P to be the sum of the weights of the edges in P plus the sum of the penalties of the penalized subpaths in P. For example, in a road network where an edge-weight denotes the travel time along the edge, the extra time (delay) needed for certain left turns during rush hours can be modeled as penalties of corresponding two-edge paths. A penalized path generalizes the "non-transitive behavior" modeled by forbidden paths because a penalized path with a very high penalty is equivalent to a forbidden path. Our construction can be generalized to handle non-overlapping penalized paths, but we do not see how to avoid an exponential blow-up in the general case.

Chapter 7

Conclusion

In this thesis we studied the problem of finding shortest paths subject to various feasibility constraints arising in practice. We focused on shortest paths in two different settings: in terrains and in graphs.

In Chapters 3 to 5, we explored terrain shortest paths subject to certain height-related constraints:

- In Chapter 3 we studied the shortest descending path (SDP) problem. We gave a full characterization of the bend angles of an SDP, and then reduced the SDP problem to the apparently simpler problem of finding an SDP through a given sequence of faces. We devised two approximation algorithms to find SDPs through given faces, and two polynomial time (exact) algorithms to find SDPs in two special classes of terrains.
- In Chapter 4 we presented two $(1+\epsilon)$-approximation algorithms for SDPs in general terrains. Both algorithms are based on the Steiner point approach, and are simple, robust and easy to implement. The running times are not comparable in general, but we can choose the one that is faster for a given terrain.
- In Chapter 5 we generalized the SDP problem to the shortest gently descending path (SGDP) problem. We explored some properties of SGDPs, and devised using the Steiner point approach two easy-to-implement $(1+\epsilon)$-approximation algorithms for the problem. We showed that finding an SGDP with a limited number of bends or a limited total turn-angle is NP-hard.

In Chapter 6 we focused on the graph shortest path problem of finding shortest paths avoiding forbidden subpaths. We presented an algorithm to compute such paths in the model where forbidden subpaths are not known a priori. We adapted the algorithm to three classes of graphs, and explored two types of oracles that reveal different amount of information when queried about forbidden subpaths.

In terrains, the most challenging open question seems to be the complexity of any of the following problems: the SDP problem, the SGDP problem, the weighted region problem, or the shortest anisotropic path problem which generalizes the other three problems. Our results imply that the SDP problem (and hence the SGDP problem) remains open even when we know

the sequence of faces used by the shortest path. For the weighted region problem, numerical issues in finding a shortest path through given faces remain unsolved 18 years after the issues were first encountered. Hence a more "promising" open problem seems to be to prove the hardness of any of the four problems, perhaps for the case of a given face sequence. One step in this direction may be showing the hardness of a related bicriteria shortest path problem (e.g., SDPs with at most k bends). We gave the hardness of two bicriteria problems for SGDPs (and for shortest anisotropic paths). For other related bicriteria problems the hardness may be non-trivial. Even in a simpler 2D setting, the complexity of the bicriteria problem of finding a shortest k-link path in a polygon with holes has been open for many years.

Another interesting direction for further research is characterizing the bend angles and other structural properties of SGDPs and shortest anisotropic paths in order to identify terrains in which the problems are solvable in polynomial time, or to identify easy-to-handle cost models for the shortest anisotropic path problem. Note that only approximation algorithms are known for the shortest anisotropic path problem, and they are based on a restricted cost model.

Exploring structural properties of SGDPs and shortest anisotropic paths may also help in devising faster approximation schemes. The fastest approximation scheme for the weighted region problem is the one by Aleksandrov et al. [15], which places Steiner points on bisectors of face angles, and relies on properties of locally shortest paths for error analysis. This seems to be the only algorithm among all the Steiner point based algorithms for the weighted region problem, the SDP problem, the SGDP problem, and the shortest anisotropic path problem to utilize properties of locally shortest paths. Our preliminary effort to adapt the idea of Aleksandrov et al. for the SDP problem did not work out, but it is conceivable that utilizing properties of locally shortest paths in some way should yield a Steiner point graph smaller than one built without using any structural properties of the desired path. This demands further investigation, particularly for the shortest anisotropic path problem where even approximation schemes are not known for the general cost model.

In graphs, one interesting problem is find a shortest path when the set of forbidden paths is a dynamic set, which means that forbidden paths can be added to, or deleted from, the set. For this problem our techniques offer nothing beyond the naive approach of re-starting the algorithm from scratch at most changes to the set.

For the case of static but unknown forbidden paths, an interesting research direction is adapting our technique for the problem of finding a shortest path for a vehicle in a road network, where the discovery of a forbidden path should cause a detour *from the point of failure*. The problem is non-trivial because it is easy for an adversary to "fool" the vehicle into entering a part of the network from which the destination is unreachable.

We can also generalize the idea of forbidden paths to *penalized* paths each having a positive weight (a *penalty*) representing, for example, the extra time (delay) needed for certain left turns during rush hours in a road network. Our construction can be generalized to handle non-overlapping penalized paths, but it is not clear how to avoid an exponential blow-up when we have overlapping penalized paths.

References

[1] Peyman Afshani, Jeremy Barbay, and Timothy M. Chan. Instance-optimal geometric algorithms. In *Proceedings of the 50th Annual Symposium on Foundations of Computer Science*, page (to appear), 2009. 1

[2] Pankaj K. Agarwal, Sariel Har-Peled, and Meetesh Karia. Computing approximate shortest paths on convex polytopes. *Algorithmica*, 33(2):227–242, 2002. 2, 2.1, 4.1

[3] Pankaj K. Agarwal, Sariel Har-Peled, Micha Sharir, and Kasturi R. Varadarajan. Approximating shortest paths on a convex polytope in three dimensions. *J. ACM*, 44(4):567–584, 1997. 2

[4] Mustaq Ahmed, Sandip Das, Sachin Lodha, Anna Lubiw, Anil Maheshwari, and Sasanka Roy. Approximation algorithms for shortest descending paths in terrains. *J. Discrete Alg.*, May 2009. Accepted for publication. http://dx.doi.org/10.1016/j.jda.2009.05.001. 3.1, 1, 1

[5] Mustaq Ahmed and Anna Lubiw. Shortest descending paths through given faces. In *Proceedings of the 18th Canadian Conference on Computational Geometry (CCCG)*, pages 35–38, August 2006. 1

[6] Mustaq Ahmed and Anna Lubiw. An approximation algorithm for shortest descending paths. *CoRR*, 0705.1364v1 [cs.CG], May 2007. 1, 4.2.1

[7] Mustaq Ahmed and Anna Lubiw. Properties of shortest descending paths. The 17th Fall Workshop on Computational and Combinatorial Geometry (FWCG), IBM T.J. Watson Research Center, Hawthorne, New York, November 2007. 1

[8] Mustaq Ahmed and Anna Lubiw. Shortest anisotropic paths with few bends is NP-complete. In *The 18th Fall Workshop on Computational Geometry (FWCG): Abstracts*, pages 28–29, Rensselaer Polytechnic Institute, Troy, New York, October 2008. 2

[9] Mustaq Ahmed and Anna Lubiw. Shortest descending paths through given faces. *Comput. Geom. Theory Appl.*, 42(5):464–470, July 2009. Special issue on selected papers from Canadian Conference on Computational Geometry (CCCG) 2005 and 2006. http://dx.doi.org/10.1016/j.comgeo.2007.10.011. 3.1, 1

[10] Mustaq Ahmed and Anna Lubiw. Shortest paths avoiding forbidden subpaths. In *Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 63–74, February 2009. 1

[11] Mustaq Ahmed, Anna Lubiw, and Anil Maheshwari. Shortest gently descending paths. In *Proceedings of the Third Annual Workshop on Algorithms and Computation (WALCOM)*, volume 5431 of *Lecture Notes in Computer Science*, pages 59–70. Springer-Verlag, February 2009. 1

[12] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975. 6.1.4

[13] Lyudmil Aleksandrov, Mark Lanthier, Anil Maheshwari, and Jörg-Rüdiger Sack. An ϵ-approximation algorithm for weighted shortest paths on polyhedral surfaces. In *Proceedings of the Sixth Scandinavian Workshop on Algorithm Theory*, volume 1432 of *Lecture Notes in Computer Science*, pages 11–22, Berlin, Germany, 1998. Springer-Verlag. 2, 2.3.1, 2.3.1, 2.3.1, 4.2.1, 4.4.1

[14] Lyudmil Aleksandrov, Anil Maheshwari, and Jörg-Rüdiger Sack. Approximation algorithms for geometric shortest path problems. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 286–295, New York, NY, USA, 2000. ACM Press. 2, 2.3.1, 2.3.1, 2.3.1, 2.3.1, 4.2.1

[15] Lyudmil Aleksandrov, Anil Maheshwari, and Jörg-Rüdiger Sack. Determining approximate shortest paths on weighted polyhedral surfaces. *J. ACM*, 52(1):25–53, 2005. 2, 2.3.1, 2.3.1, 2.3.1, 4.1, 4.2.1, 4.5, 7

[16] Esther M. Arkin, Joseph S. B. Mitchell, and Christine Piatko. Bicriteria shortest path problems in the plane. In *Proceedings of the Third Canadian Conference on Computational Geometry*, pages 153–156, August 1991. See also the PhD thesis of Christine Piatko. 5.1

[17] Peter Ashwood-Smith. Personal communication, 2007. 6.1.1

[18] Peter Ashwood-Smith, Don Fedyk, and Vik Saxena. Link viability constraints requirements for GMPLS-enabled networks. `http://tools.ietf.org/html/draft-ashwood-ccamp-gmpls-constraint-reqts-00`, July 2005. Internet draft, work in progress. 6.1.1

[19] Mikhail J. Atallah and Susan Fox, editors. *Algorithms and Theory of Computation Handbook*. CRC Press, Inc., Boca Raton, FL, USA, 1998. 1, 3.3.3

[20] Chanderjit Bajaj. The algebraic complexity of shortest paths in polyhedral spaces. In *Proceedings of the 23rd Allerton Conference on Communication, Control and Computing*, pages 510–517, 1985. 2.1, 3.6

[21] Chanderjit Bajaj. The algebraic degree of geometric optimization problems. *Discrete Comput. Geom.*, 3(2):177–191, 1988. 2.1, 3.6, 4.1

[22] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004. 3.2, 3.3.2, 3.3.3

[23] Sergio Cabello. Many distances in planar graphs. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1213–1220, New York, NY, USA, 2006. ACM Press. 6.1.4

[24] John F. Canny and John H. Reif. New lower bound techniques for robot motion planning problems. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 49–60, 1987. 2.2.3, 3.1, 5.6

[25] Edward P. F. Chan and Yaya Yang. Shortest path tree computation in dynamic graphs. *IEEE Trans. Computers*, 58(4):541–557, 2009. 6.1.4

[26] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991. 2.1

[27] Bernard Chazelle, Ding Liu, and Avner Magen. Sublinear geometric algorithms. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 531–540, New York, NY, USA, 2003. ACM. 2

[28] Jindong Chen and Yijie Han. Shortest paths on a polyhedron. I. Computing shortest paths. *Internat. J. Comput. Geom. Appl.*, 6(2):127–144, 1996. 2, 2.2.2, 3.1, 3.4, 3.4.1, 3.5.2

[29] Siu-Wing Cheng, Hyeon-Suk Na, Antoine Vigneron, and Yajun Wang. Approximate shortest paths in anisotropic regions. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 766–774, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. 4.5

[30] Siu-Wing Cheng, Hyeon-Suk Na, Antoine Vigneron, and Yajun Wang. Querying approximate shortest paths in anisotropic regions. In *Proceedings of the 23rd Annual Symposium on Computational Geometry*, pages 84–91, New York, NY, USA, 2007. ACM. 2.3.2

[31] Siu-Wing Cheng, Hyeon-Suk Na, Antoine Vigneron, and Yajun Wang. Approximate shortest paths in anisotropic regions. *SIAM J. Comput.*, 38(3):802–824, 2008. 2, 2.3.2, 4.2.1, 5.1

[32] Joonsoo Choi, Jürgen Sellen, and Chee-Keng Yap. Approximate Euclidean shortest path in 3-space. In *Proceedings of the 10th Annual Symposium on Computational Geometry*, pages 41–48, New York, NY, USA, 1994. ACM. 2.2.3

[33] Kenneth L. Clarkson. Approximation algorithms for shortest path motion planning. In *Proceedings of the 19th Annual ACM Conference on Theory of Computing*, pages 56–65, New York, NY, USA, 1987. ACM Press. 2.3.1

[34] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001. 1, 6.1.3, 6.1.3, 6.1.4, i, ii, iii, 6.4, 6.4, 6.4

[35] Ovidiu Daescu, Joseph S. B. Mitchell, Simeon C. Ntafos, James D. Palmer, and Chee-Keng Yap. k-link shortest paths in weighted subdivisions. In *Proceedings of the Ninth Workshop on Algorithms and Data Structures*, volume 3608 of *Lecture Notes in Computer Science*, pages 325–337, Berlin, Germany, 2005. Springer-Verlag. 5.1

[36] Mark de Berg, Matthew J. Katz, A. Frank van der Stappen, and Jules Vleugels. Realistic input models for geometric algorithms. *Algorithmica*, 34(1):81–97, 2002. 1

[37] Mark de Berg and Marc J. van Kreveld. Trekking in the Alps without freezing or getting tired. *Algorithmica*, 18(3):306–323, 1997. 1.1, 3.1

[38] Mark de Berg, Marc J. van Kreveld, Mark Overmars, and Otfried Cheong. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000. 2.1

[39] Ernesto de Queiros Vieira Martins. An algorithm for ranking paths that may contain cycles. *European Journal of Operational Research*, 18(1):123–130, October 1984. 6.1, 6.1.4, 6.2

[40] Erik D. Demaine, Joseph S. B. Mitchell, and Joseph O'Rourke. The open problems project. http://maven.smith.edu/~orourke/TOPP/. 1

[41] Camil Demetrescu, Stefano Emiliozzi, and Giuseppe F. Italiano. Experimental analysis of dynamic all pairs shortest path algorithms. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 369–378, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics. 6.1.4

[42] Camil Demetrescu, Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni. Maintaining shortest paths in digraphs with arbitrary arc weights: an experimental study. In *Proceedings of the Fourth International Workshop on Algorithm Engineering*, pages 218–229, London, UK, 2001. Springer-Verlag. 6.1.4

[43] Camil Demetrescu, Mikkel Thorup, Rezaul A. Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008. 6.1.4

[44] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. 2.2.1, 2.2.3

[45] David Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673, 1999. 6.1.4

[46] Leila De Floriani, Paola Magillo, and Enrico Puppo. Applications of computational geometry to geographic information systems. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 333–388. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 1998. 1

[47] Michael L. Fredman and Robert E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. 4.3.2, 4.4.2

[48] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, NY, USA, 1990. 5.2, 5.6.6

[49] Andrew V. Goldberg and Chris Harrelson. Computing the shortest path: A^* search meets graph theory. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 156–165, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics. 6.1.4

[50] Andrew V. Goldberg, Haim Kaplan, and Renato Fonseca F. Werneck. Better landmarks within reach. In *Proceedings of the Sixth International Workshop on Experimental Algorithms*, volume 4525 of *Lecture Notes in Computer Science*, pages 38–51, Berlin, Germany, 2007. Springer-Verlag. 6.1.4

[51] Luis Gouveia, Pedro Patrício, Amaro de Sousa, and Rui Valadas. MPLS over WDM network design with packet level QoS constraints based on ILP models. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, April 2003. 6.1.1, 6.1.4

[52] Sariel Har-Peled. Approximate shortest paths and geodesic diameter on a convex polytope in three dimensions. *Discrete Comput. Geom.*, 21(2):217–231, 1999. 2

[53] Sariel Har-Peled. Constructing approximate shortest path maps in three dimensions. *SIAM J. Comput.*, 28(4):1182–1197, 1999. 2

[54] John Hershberger and Subhash Suri. Practical methods for approximating shortest paths on a convex polytope in $\mathbb{R}^3$. *Comput. Geom. Theory Appl.*, 10(1):31–46, 1998. 2

[55] John Hershberger and Subhash Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Comput.*, 28(6):2215–2256, 1999. 2, 3.1

[56] John Hershberger and Subhash Suri. Vickrey prices and shortest paths: what is an edge worth? In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, pages 252–259, 2001. 1

[57] John Hershberger, Subhash Suri, and Amit Bhosle. On the difficulty of some shortest path problems. *ACM Trans. Algorithms*, 3(1):5, 2007. 6.1.4

[58] Frederick S. Hillier and Gerald J. Lieberman. *Introduction to Operations Research*. McGraw-Hill, 8th edition, 2005. 1

[59] Martin Holzer, Frank Schulz, Dorothea Wagner, and Thomas Willhalm. Combining speed-up techniques for shortest-path computations. *J. Exp. Algorithmics*, 10:2.5, 2005. 6.1.4

[60] Biliana Kaneva and Joseph O'Rourke. An implementation of Chen & Han's shortest paths algorithm. In *Proceedings of the 12th Canadian Conference on Computational Geometry*, pages 139–146, August 2000. 2.2.2, 3.6

[61] Sanjiv Kapoor. Efficient computation of geodesic shortest paths. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 770–779, New York, NY, USA, 1999. ACM Press. 2

[62] Samir Khuller, Kwangil Lee, and Mark A. Shayman. On degree constrained shortest paths. In *Proceedings of the 13th Annual European Symposium on Algorithms*, pages 259–270, 2005. 6.1.4

[63] Mark Lanthier, Anil Maheshwari, and Jörg-Rüdiger Sack. Approximating weighted shortest paths on polyhedral surfaces. In *Proceedings of the 13th Annual Symposium on Computational Geometry*, pages 274–283, New York, NY, USA, 1997. ACM Press. 2, 2.3.1, 2.3.1, 2.3.1

[64] Mark Lanthier, Anil Maheshwari, and Jörg-Rüdiger Sack. Shortest anisotropic paths on terrains. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming*, pages 524–533, London, UK, 1999. Springer-Verlag. 2.3.2, 4.2.1, 4.2.2, 5.1

[65] Aaron W. F. Lee, David Dobkin, Wim Sweldens, and Peter Schröder. Multiresolution mesh morphing. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 343–350, New York, NY, USA, August 1999. ACM Press/Addison-Wesley Publishing Co. 1

[66] Kwangil Lee and Mark A. Shayman. Optical network design with optical constraints in IP/WDM networks. *IEICE Trans. on Communications*, E88-B(5):1898–1905, 2005. 6.1.1, 6.1.4

[67] Miguel S. Lobo, Lieven Vandenberghe, Stephen Boyd, and Hervé Lebret. Applications of second-order cone programming. *Linear Algebra and its Applications*, 284:193–228, 1998. 3.3.3

[68] Andrew McGregor and Bruce Shepherd. Island hopping and path colouring with applications to WDM network design. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithm*, pages 864–873, Philadelphia, PA, USA, January 2007. Society for Industrial and Applied Mathematics. 6.1.4

[69] Joseph S. B. Mitchell. Geometric shortest paths and network optimization. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 633–701. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000. 1, 3.1, 5.1

[70] Joseph S. B. Mitchell, David M. Mount, and Christos H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16(4):647–668, 1987. 2, 2.1, 2.2.1, 2.2.2, 2.3.1, 3.1, 3.2, 3.3, 3.3.1, 3.3.1, 3.4.1

[71] Joseph S. B. Mitchell and Christos H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *J. ACM*, 38(1):18–73, 1991. 2, 2.3.1, 2.3.1, 2.3.2, 3.1, 3.3.2, 3.6

[72] Daniel Y. Mo, Raymond K. Cheung, Allen W. Lee, and Gil K. Law. Flow diversion strategies for routing in integrated automatic shipment handling systems. *IEEE Trans. on Automation Science and Engineering*, 6(2):377–384, April 2009. 1

[73] Esther Moet, Marc van Kreveld, and A. Frank van der Stappen. On realistic terrains. In *Proceedings of the 22nd Annual Symposium on Computational Geometry*, pages 177–186, New York, NY, USA, 2006. ACM Press. 4.5

[74] David M. Mount. On finding shortest paths on convex polyhedra. Technical Report 1495, Department of Computer Science, University of Maryland, 1985. 2.2.1

[75] Joseph O'Rourke, Subhash Suri, and Heather Booth. Shortest paths on polyhedral surfaces. In *Proceedings of the Second Annual Symposium on Theoretical Aspects of Computer Science*, volume 182 of *Lecture Notes in Computer Science*, pages 243–254. Springer, 1985. 2

[76] Christos H. Papadimitriou. An algorithm for shortest-path motion in three dimensions. *Inform. Process. Lett.*, 20:259–263, 1985. 2, 2.2.3, 2.2.3, 2.3.1, 2.3.1

[77] Valentin Polishchuk and Joseph S. B. Mitchell. Touring convex bodies—a conic programming solution. In *Proceedings of the 17th Canadian Conference on Computational Geometry*, pages 290–293, 2005. 3.3.3

[78] Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985. 1

[79] Rajiv Ramaswami and Kumar N. Sivarajan. *Optical Networks: A Practical Perspective*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002. 6.1.4

[80] John H. Reif and Zheng Sun. Movement planning in the presence of flows. *Algorithmica*, 39(2):127–153, 2004. 2.3.2

[81] Neil C. Rowe and Ron S. Ross. Optimal grid-free path planning across arbitrarily-contoured terrain with anisotropic friction and gravity effects. *IEEE Trans. Robot. Autom.*, 6(5):540–553, 1990. 1, 2.3.2, 4.2.1

[82] Sasanka Roy. *Algorithms for some geometric facility location and path planning problems*. PhD thesis, Indian Statistical Institute, Kolkata, India, 2008. 3.1

[83] Sasanka Roy, Sandip Das, and Subhas C. Nandy. Shortest monotone descent path problem in polyhedral terrain. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science*, volume 3404 of *Lecture Notes in Computer Science*, pages 281–292, Berlin, Germany, 2005. Springer-Verlag. 3.1, 3.3.1, 3.3.1

[84] Sasanka Roy, Sandip Das, and Subhas C. Nandy. Shortest monotone descent path problem in polyhedral terrain. *Comput. Geom. Theory Appl.*, 37(2):115–133, 2007. 3.1, 3.3.1

[85] Sasanka Roy, Sachin Lodha, Sandip Das, and Anil Maheshwari. Approximate shortest descent path on a terrain. In *Proceedings of the 19th Canadian Conference on Computational Geometry*, pages 189–192, August 2007. 1

[86] Yevgeny Schreiber. Shortest paths on realistic polyhedra. In *Proceedings of the 23rd Annual Symposium on Computational Geometry*, pages 74–83, New York, NY, USA, 2007. ACM. 3.1

[87] Yevgeny Schreiber and Micha Sharir. An optimal-time algorithm for shortest paths on a convex polytope in three dimensions. In *Proceedings of the 22nd Annual Symposium on Computational Geometry*, pages 30–39, New York, NY, USA, 2006. ACM. 2, 3.1

[88] Micha Sharir and Amir Schorr. On shortest paths in polyhedral spaces. *SIAM J. Comput.*, 15(1):193–215, 1986. 2, 2.2.1

[89] Zheng Sun and Tian-Ming Bu. On discretization methods for approximating optimal paths in regions with direction-dependent costs. *Inform. Process. Lett.*, 97(4):146–152, 2006. 2.3.2, 4.2.1, 4.2.2, 5.1

[90] Zheng Sun and John H. Reif. BUSHWHACK: An approximation algorithm for minimal paths through pseudo-Euclidean spaces. In *Proceedings of the 12th International Symposium on Algorithms and Computation*, pages 160–171, London, UK, 2001. Springer-Verlag. 2.3.1

[91] Zheng Sun and John H. Reif. On finding energy-minimizing paths on terrains. *IEEE Trans. on Robotics*, 21(1):102–114, 2005. 2.3.2, 2.3.2, 4.2.1, 4.2.2, 5.1, 5.4.1

[92] Zheng Sun and John H. Reif. On finding approximate optimal paths in weighted regions. *J. Algorithms*, 58(1):1–32, 2006. 2, 2.3.1, 2.3.1, 4.2.1

[93] Vitaly Surazhsky, Tatiana Surazhsky, Danil Kirsanov, Steven J. Gortler, and Hugues Hoppe. Fast exact and approximate geodesics on meshes. *ACM Trans. Graph.*, 24(3):553–560, July 2005. 1

[94] Stefan Szeider. Finding paths in graphs avoiding forbidden transitions. *Discrete Appl. Math.*, 126(2-3):261–273, 2003. 6.1, 6.1.4

[95] Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM*, 46(3):362–394, 1999. 6.1.4

[96] Mikkel Thorup. Equivalence between priority queues and sorting. *J. ACM*, 54(6):28:1–28:27, 2007. 6.1.4

[97] Chris Upchurch, Michael Kuby, Michael Zoldak, and Anthony Barranda. Using GIS to generate mutually exclusive service areas linking travel on and off a network. *Journal of Transport Geography*, 12(1):23–33, 2004. 1

[98] Kasturi R. Varadarajan and Pankaj K. Agarwal. Approximating shortest paths on a nonconvex polyhedron. *SIAM J. Comput.*, 30(4):1321–1340, 2000. 2

[99] Daniel Villeneuve and Guy Desaulniers. The shortest path problem with forbidden paths. *European Journal of Operational Research*, 165(1):97–07, 2005. 6.1, 6.1.4, 6.4

[100] Jiantao Wang, Lun Li, Steven H. Low, and John C. Doyle. Can shortest-path routing and TCP maximize utility. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 2049–2056, March–April 2003. 1

[101] Wilbert E. Wilhelm, Ivette Arambula, and Neil N. D. Choudhry. Optimizing picking operations on dual-head placement machines. *IEEE Trans. on Automation Science and Engineering*, 3(1):1–15, January 2006. 1

[102] Dianna Xu. Shortest paths on polytopes, 1996. Undergraduate thesis, Smith College. 3.6

[103] Uri Zwick. Exact and approximate distances in graphs—a survey. In *Proceedings of the Ninth Annual European Symposium on Algorithms*, pages 33–48, London, UK, 2001. Springer-Verlag. 6.1.4