Computing Realistic Terrains from Imprecise Elevations

by

Graeme Stroud

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the 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.

Statement of Contributions

Graeme Stroud was the sole author for Chapters 1, 2, 4, 5, and Section 3.1 of Chapter 3. These chapters were all written under the supervision of Anna Lubiw.

Sections 3.2 and 3.3 of this thesis were written for publication, see [4], and the full version of the paper [5]. I was the primary authour responsible for these sections. Anna Lubiw helped reformat these sections, including the diagrams.

Abstract

It is ideal for triangulated terrains to have characteristics or properties that are realistic. In the imprecise terrain model, each vertex of a triangulated terrain has an imprecise elevation value only known to lie within some interval. Under some objective function, the goal is to compute a precise terrain by assigning a single elevation value to each point, so that the objective function is optimized.

This thesis examines various objectives, such as minimizing the number of local extrema and minimizing the terrain's surface area. We give algorithms in some cases, hardness results in other cases. Specifically, we consider four objectives: (1) minimizing the number of local extrema; (2) optimizing coplanar features; (3) minimizing the surface area; (4) minimizing the maximum steepness.

Problem (1) is known to be NP-hard, but we give an algorithm for a special case. For problem (2) we give an NP-hardness proof for the general case and a positive result for a special case. Meanwhile, problems (3) and (4) can be approximated using Second Order Cone Programming. We also consider versions of these problems for terrains one dimension down, where the output is a polyline. Here we give very efficient algorithms for all objective functions considered.

Finally, we go beyond terrains and briefly consider the distant representatives problem, where the goal is to choose precise points from segments to be as far from each other as possible. For this problem, we give a parameterized algorithm for vertical segments, prove NP-hardness for unit horizontal segments, and show hardness of approximation for vertical and horizontal segments.

Acknowledgements

I would like to thank Professor Anna Lubiw for her guidance throughout this challenging time, and for the many comments and suggestions that made this thesis stronger. I would like to thank to Professor Craig Kaplan and Professor Therese Biedl for agreeing to read my thesis and providing feedback.

Dedication

I would like to dedicate this thesis to my family, for their unconditional love and support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In computational geometry, a "terrain" is a surface that is intersected at most once by any vertical line. A natural problem that arises in Geographic Information Systems is to compute a triangulated terrain in 3D space that has "nice" or "realistic" properties. There is no correct or best property, as it will depend on context. In the study of erosion and hydrology, it is generally accepted that pits in a triangulated terrain are artifacts of imprecision, due to the unrealistic occurrence of water accumulation in flow simulations [22]. Meanwhile, flat terrains better represent certain real life landscapes, like plains, shields, while smooth terrains better represent landscapes like dunes.

There are certain features of a terrain we can measure that are relevant in GIS, such as the quantity or size of peaks or valleys, or the steepness of part of the terrain. Gray et al. [24] explore another feature: "Measures on spatial angles, in particular the angle between the surface normals of adjacent triangles, are known to be important for several GIS applications. In particular, they have been shown to be good for improving the quality of the approximation, for achieving good slope characteristics and for flow modeling".

A triangulated terrain is often computed from real elevation data. It is usually assumed that the data is accurate, however data acquisition can be complex and potentially prone to errors. It may be appropriate to model the input data as coming from a possible range of values to account for this uncertainty. This uncertainty motivated Gray et al [21] to formulate the imprecise 2.5D terrain model. In this model, the x, y-coordinates of points are given as input, in addition to a triangulation of the points when projected to the xy plane, but, the z-coordinate (elevation) of each point is only known "imprecisely" within some interval of possible values. The x, y data could also be considered to be imprecise, but it is a simplification to only have the elevation values to be imprecise. One justification for this model is that allowing inaccuracy in the z-coordinate can compensate for inaccuracy in the x-y coordinates.

A precise 2.5D terrain can be computed by choosing a precise elevation from each uncertainty interval and connecting the points together according to the input triangulation. In order to determine whether one choice is better or nicer than another, one should come

up with a "niceness" objective function to optimize over all possible choices of the elevation values. Multiple criteria have been considered in the past such as minimizing the number of local extrema [22], or minimizing the length of the shortest path along the terrain from one point to another [21] [30]. Many of these formulations lead to NP-hard problems. When these problems are NP-hard or have unknown computational complexity for 2.5D terrains, researchers have gone down a dimension and considered terrains in 1.5D, see Gray et al. [24] for example. Here, the x-coordinates are precise, and the elevations are the y-coordinates, which are given imprecisely.

The main goal of this thesis to explore various niceness objective functions in the imprecise terrain model. There are four objective functions we will consider. One of these has been studied before, and to the best of our knowledge, the other problems have yet to be explored. We will state the four objectives, and briefly discuss the properties and applications.

Objective #1: Minimizing the number of local extrema. A local extremum is a local maximum or minimum compared to its neighbours in the triangulation. In a terrain these correspond to peaks or valleys. The problem of minimizing the number of local extrema was considered by Gray et al. [22]. As mentioned by Gray et al., local extrema do not appear frequently in terrains, they usually suggest some form of elevation error. For instance, in the study of hydrology, pits are unnatural since water always flows downhill and erodes the land. The definition of extrema used by Gray et al. counts all minima (similarly maxima) that are connected together and are all at equal elevation as one minimum (similarly maximum), which they justify for applications. We will use this definition of extrema as well.

Objectives #2a and #2b: Optimizing coplanar features. In order to have a terrain that is smooth, ideally we would like triangles to be coplanar to adjacent triangles if possible. This idea results in two similar objectives.

Objective #2a: Minimizing the number of patches (links). The intention of this objective is to have as many large and smooth patches on the terrain as possible. A ***patch*** is defined to consist of a maximal set of connected triangles that are all coplanar. This is natural in cases where the data is obtained from smooth or flat real-life terrains, such as plains or dunes. Note that a 1.5D terrain does not consist of triangles, so this definition does not apply in the imprecise 1.5D terrain model. Instead, we define a ***link*** to be a maximal set of edges of the terrain that are connected and are collinear.

Objective #2b: Minimizing the number of bends. Minimizing the number of patches is similar to the objective of minimizing the number of ***bends*** at edges, where an edge is a bend if the dihedral angle between the two triangles sharing this edge is not equal to π. In 1.5D, minimizing the number of links is the same objective as minimizing the number of bends. In 2.5D, if the triangulation of the points gives an outerplanar graph, then these

objectives are also equivalent. These are not equivalent objective function in general, as we will see in Section 4.2. But, they both have a similar goal of returning a terrain that has lots of connected coplanar triangles, so we will explore both objectives.

Objective #3: Minimizing the total surface area (length). Minimizing the total surface area is a very natural objective function. This is similar to the topic of minimal surfaces, which is a well studied topic in mathematics. However, a 2.5D terrain is only a piecewise approximation of an actual minimal surface. Optimizing our objective can ensure that the 2.5D terrain we compute approximates a minimal surface as best as possible. In 1.5D, we instead look at the *total length* of the 1.5D terrain. In a similar vein, approximation algorithms for computing a minimal surface from a set of points have been considered [38].

Objectives #4a and #4b: Optimizing steepness. We look at two objectives for optimizing steepness.

Objective #4a: Minimizing the maximum steepness. This objective prefers the segments of the 1.5D terrain to have a slope as close to 0 as possible, so the goal is not only to have a smooth terrain, but to have a terrain that is as flat as possible. There is a standard notion for the steepness of a plane (or triangle) in 3D space, which is the L_2 norm of the gradient of the plane. This is a very natural objective for real life terrains that are flat. This is similar to the third objective function, which also ends up trying to keep the triangles flat.

Objective #4b: Maximizing the minimum steepness. We can also look at the problem of maximizing the minimum steepness. The resulting terrain is not flat, and may have sharp bends, so it is not a very good niceness objective. The main reason we will consider this problem is because it ties in nicely to another class of problems in the imprecise points model, known as *dispersion* problems [2], which try to place points as far from each other as possible.

1.1 Overview of results

In Chapter 2, we start with a warm-up and explore these objectives one dimension down. In the imprecise 1.5D terrain model, the x-coordinates of the points are specified, and each y-coordinate may be chosen from some given interval. After their elevations have been chosen, connecting points consecutive in the x order gives a precise 1.5D terrain, i.e., an x-monotone polyline. This is an easier setting than in 2.5D, but it is still interesting to examine the objectives in this model. A summary of these results is given in the first column of Table 1.1.

Objective function	1.5D terrain	2.5D Path	2.5D General
#1 minimize number of extrema	$O(n)$ [§2.1]	$O(n^4)$ [§4.1]	NP-hard [22]
#2a,b coplanar features	**2-approx** [§2.2]	**5-approx*** [§4.2.2]	**NP-hard** [§4.2.3]
#3 minimize length (1.5D) / surface area (2.5D)	$O(n)$ [§2.3]	**SOCP**	**SOCP** [§4.3]
#4a minimize max steepness	$O(n)$ [§2.4.1]	**SOCP**	**SOCP** [§4.4]
#4b maximize min steepness	$O(n^3)$ [§2.4.2]	Unknown	Unknown

Table 1.1: Algorithm or hardness results for problems involving imprecise points. Results for 1.5D terrains are given in the first column, and results for 2.5D terrains are given in the last column. The middle column gives results for 2.5D restricted to path triangulations. Bold results are original to this thesis.
*The terrain is assumed to lie in a strip for this algorithm. See Section 4.2 for more details.

In Chapter 4, we focus on the imprecise 2.5D terrain model. See the last two columns of Table 1.1 for a summary of our results. For the problem of minimizing the number of extrema (Objective #1), Gray et al. [22] have shown that it is NP-hard in the general case. In hopes of obtaining a positive result, we considered a special case based on the type of input triangulation. We will look at inputs where the triangulation is a path of triangles, and give a polynomial time algorithm for this problem. See Section 4.1.

For Objectives #2a and #2b, we show that they are both NP-hard in the general case (Section 4.2.3), but there is a constant factor approximation algorithm when the triangulation is restricted to a path of triangles within a strip (Section 4.2.2). For the problems of minimizing the total surface area (Objective #3) and minimizing the maximum steepness (Objective #4a), we provide Second Order Cone Programs (SOCPs) that solve the problems. Second Order Cone Programming is a type of convex optimization problem that can be solved quite efficiently (though not in polynomial time). See Sections 4.3 and 4.4.

In Section 3, we take a detour from terrains and look at another imprecise points problem for line segments. The problem we will look at is a type of dispersion problem called the distant representatives problem [18]. The goal is to choose one point per input object in order to maximize the minimum pairwise distance between the points. We look at a special case where the objects are segments, which makes the problem similar in flavour to imprecise 1.5D terrains—and is also a dispersion problem that is similar to Objective #4b.

1.2 Background

Imprecise points model

The imprecise points model has been explored for other objectives unrelated to terrains. In particular, given a class of objects in the plane as input, one can ask for a set of points (one per input object) that optimizes some objective function. For example, for classes like line segments or squares, Löffler and Kreveld [33] give algorithms or hardness results for the objectives of minimizing/maximizing the area or perimeter of the convex hull of the points. Other optimization objectives for imprecise points have been explored, including minimizing/maximizing the width, the area of the bounding box, and the diameter of the points [28] [34]. Another optimization objective is to compute the best Euclidean weight of a spanning tree [14], over all possible placement of points and over all possible trees of the points. A non-optimization objective was considered by Löffler and Snoeyink [32], which asks to compute a data structure to efficiently compute the Delaunay triangulation for any possible precise realization of the imprecise points. Even more general is the study of how imprecision affects the accuracy of geometric computations, which is called "epsilon geometry" [42].

Previous results for imprecise 1.5D and 2.5D terrains

Gray and Evans [21] were the first to consider the imprecise terrain model. The problem they considered was finding the shortest path from one point to another over all precise realizations of the terrain. This problem is shown to be NP-hard in 2.5D. Kholondyrev [30] also looked at the same problem, but the path must be restricted to edges of the terrain. This has a polynomial time algorithm.

Various other objective functions have been explored for imprecise 1.5D and 2.5D terrains. The problem of minimizing the number of extrema was explored by Gray et al. [22]. In the general 2.5D case, they showed that minimizing the number of extrema is NP-hard, and there is no $O(\log \log n)$ approximation algorithm unless P = NP. We can instead ask to just minimize the number of minima. They give an $O(n \log n)$ algorithm, but this relies on counting groups of adjacent points with the same elevation as a single minimum. Otherwise, the problem is shown to be NP-hard. Driemel et al. [16] considered the problem of determining whether water can flow between two points of an imprecise 2.5D terrain. Here, the assumption is water flows down the path of steepest descent. The problem is NP-hard in 2.5D.

Gray et al. [24] considered a few objectives that result in "smooth" terrains. The results are all for imprecise 1.5D terrains, however the objectives can be explored in the imprecise 2.5D terrain model. One of their objectives for imprecise 1.5D terrains is to minimize the total turn angle, that is, the sum of the turn angles between adjacent edges of the terrain. This is very similar to minimizing the number of bends, as we can simply apply a binary

indicator to each term of the sum to get our objective from theirs. Our objective function is more strict, because small bends are just as bad as large bends, where as the size of each bend matters for their objective. As a result, there may be very sharp turns with our objective, but regardless there will still be many coplanar triangles within the terrain. The benefit of our objective is that minimizing the number of bends is a discrete objective function, so it will be easier to deal with. For instance, we will see that our problem is in NP, but precision issues prevent their objective from being placed in NP. Additionally, Gray et al. [24] considered the objectives of maximizing the total turning angle, minimizing the largest turning angle, maximizing the smallest turning angle and minimizing the maximum slope *change*.

Comparing curve simplification with the links problem

The problem of minimizing the number of links/bends for an imprecise 1.5D terrain is related to curve simplification. Imai and Iri [27] provide an algorithm that, given as input an x-monotone polygonal line and $\varepsilon > 0$, computes another polygonal line that is within vertical distance ε of the input polyline, while containing as few (bend) points as possible. Although there is no input polyline for the links problem, the output will also be a polyline, and it will consist of as few points as possible. However, in the links/bends problem, the points are limited to the n possible x-coordinates given as input, but Imai and Iri's problem does not require the bend points to be at any specific locations. Other polyline distance metrics for simplification have been considered, see [46] for a recent result for the Fréchet and Hausdorff metrics. For a recent result on curve simplification under uncertainty, see [7]. For a nice survey about curve simplification, see Section 4.1 of [1].

Distant representatives

Distant representatives is a well-studied studied problem in the imprecise data model. It was first discussed by Fiala et al. [18], for input discs and squares. In the plane, various versions of the problem for different classes of objects are shown to be NP-hard [19] [41]. For line segments, Roeloffzen [41] looked at unit horizontal segments in the plane as input. He showed the problem is NP-hard, but did not take care of bit complexity issues in the reduction. See Section 3.2 for our reduction and how we handle bit complexity issues. We also give a similar argument to show the problem is NP-hard for the L_1, L_∞ norms. There are some known approximation algorithms as well, see [4, 8, 17].

Uncertainty over the triangulation

There are other ways to model uncertainty, such as the model where the points in 3D are precisely known, but the triangulation is not specified. This model has been explored with Objective #1 [13], and the problem is NP-hard. Outside of terrains, researchers have

considered problems of computing a triangulation with nice properties for a set of points in the 2D *plane*. Computing the Delaunay triangulation is a very famous example. Another problem is to find the triangulation with minimum weight (where weight is the sum of the edge lengths). This problem is NP-hard [36]. Famously, there is a dynamic programming algorithm for triangulating a simple *polygon* in the plane with minimum weight [20]. There is a Fixed Parameter Tractable algorithm when the boundary of the triangulation is given, and there are few points in the interior [26]. (The number of points is the parameter in this case.) Another problem in 2D is to minimize the maximum area over all triangles in a convex polygon, which can be solved in $O(n^2 \log n)$ time [29].

This model relates to the topic of surface reconstruction, which asks to construct an approximation of an unknown surface from a sample of points. See Chapter 35 of [45] for an extensive overview of this topic.

Chapter 2

1.5D Terrains

In this section we consider the problem of constructing a 1.5D terrain from imprecise data. Each point is given by a precise x-coordinate but with a range of possible y-coordinates. Our goal is to choose precise y-coordinates to optimize some objective function of the resulting terrain. More precisely:

Input: A set of intervals $I_1 = [b_1, t_1], \ldots, I_n = [b_n, t_n]$, and x-coordinates $x_1 < x_2 < \cdots < x_n$ which determine line segments $\ell_i = \{x_i\} \times I_i$. This input is called an ***imprecise 1.5D terrain***. The line segments are often called ***imprecise points*** [33].

Output: A set of coordinates $y_1 \in I_1, \ldots, y_n \in I_n$, determining points $p_i = (x_i, y_i)$. Adding line segments from p_i to $p_{i+1}, i = 1, \ldots, n-1$ gives a precise ***1.5D terrain***.

Objective function: Return a 1.5D terrain that optimizes some objective function.

The model for 1.5D terrains with imprecise elevations was introduced by Gray et al. [24], and has been solved for problems like minimizing the maximum turn angle, minimizing the total turn angle, and minimizing the maximum slope change. We examine our objectives for the imprecise 1.5D terrain model.

In Section 2.1, we explore the problem of minimizing the number of extrema (Objective #1), and give a linear time algorithm for the problem. In Section 2.2, we explore the problem of asking for a 1.5D terrain that takes as few links to describe as possible (Objective #2a). We will give a 2-approximation algorithm for this problem. This algorithm is also a 2-approximation algorithm for the problem of minimizing the number of bends (Objective #2b). In Section 2.3, we explore the problem of minimizing the total length (Objective #3), and give a linear time algorithm.

In Section 2.4.1, we explore the problem of minimizing the maximum (lexicographic) steepness (Objective #4a), and give a linear time algorithm. For this problem, lexicographic

Problem	Runtime
#1 minimize number of extrema	$\boldsymbol{O(n)}$ [§ 2.1]
#2 minimize number of links/bends	$\boldsymbol{O(n)}$ **(2-approx)** [§ 2.2]
#3 minimize total length	$\boldsymbol{O(n)}$ [§ 2.3]
#4a minimize maximum steepness	$\boldsymbol{O(n)}$ [§ 2.4.1]
#4b maximize minimum steepness	$\boldsymbol{O(n^3)}$ [§ 2.4.2]
minimize maximum turn angle	$O(\frac{n}{\varepsilon^4}\log\frac{1}{\varepsilon})$ with additive error ε [24]
minimize total turn angle	$O(n)$ [24]
maximize total turn angle	$O(n)$ [24]
minimize maximum slope change	$O(n^2)$ [24]

Table 2.1: Runtimes for various objectives for an input of n vertical segments. Bold results are original to this thesis.

means we minimize the maximum steepness, and subject to this, we minimize the second largest steepness, and so on. In Section 2.4, we look at the problem of maximizing the minimum (lexicographic) steepness (Objective #4b), and give a cubic time algorithm. For this problem, lexicographic means we maximize the minimum steepness, and subject to this, we maximize the second smallest steepness, and so on. Lexicographic objective functions have previously been explored for dispersion problems [3], as well as resource allocation [35] and facility location [37] problems.

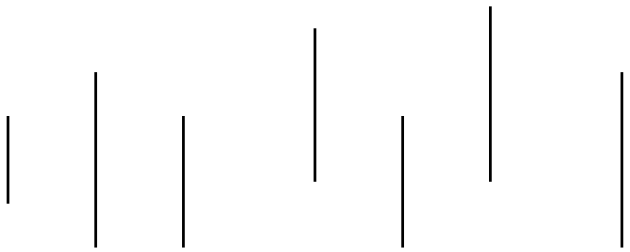
Notation and Definitions. Here we will state some common notation and definitions used throughout the four subsections.

1. Let $\mathcal{L}_i = \ell_1, \ldots, \ell_i$ denote the subproblem containing the first i vertical line segments.
2. Let $\mathcal{L}_{ij} = \ell_i, \ldots, \ell_j$ denote the subproblem containing the vertical line segments from i to j.
3. We define a polygon containing the segments as follows.

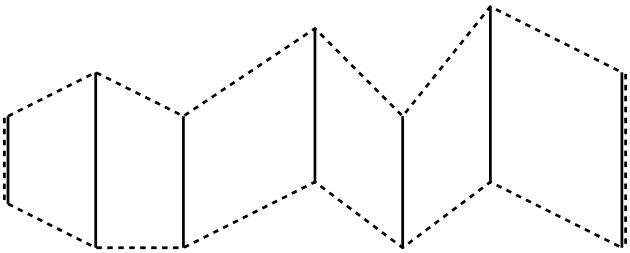
   Definition 2.0.1. Let $P_{i,j}$ denote the simple polygon, where the vertices are the top and bottom endpoints of the line segments $\ell_i, \ldots, \ell_j$, and for edges, we have the two edges ℓ_i, ℓ_j, edges connecting adjacent top endpoints, and edges connecting adjacent bottom endpoints.

   See Figure 2.1 for an example. We observe that any precise 1.5D terrain for the subproblem $\ell_i, \ldots, \ell_j$ lies within the polygon $P_{i,j}$.

   For the polygon $P_{i,j}$, we will use various algorithms which involve finding paths inside the polygon. These algorithms need a triangulation of the polygon. Thankfully, we do not need Chazelle's impractical linear time algorithm [9], since $P_{i,j}$ is composed of trapezoids each of which can be cut into two triangles.



(a) The input segments.



(b) Polygon $P_{1,n}$.

Figure 2.1: Constructing a simple polygon containing all of the vertical line segments.

2.1 Minimizing the number of local extrema

For our first problem, we will look at minimizing the number of local extrema along the 1.5D terrain. The definition we will use was first given for 2.5D terrains by Gray et al. [22].

Definition 2.1.1. Given elevations $y_1, \ldots, y_n$, a plateau is a collection of points $p_i, \ldots, p_j$ from intervals $\ell_i, \ldots, \ell_j$ so that $y_i = \cdots = y_j$.

Definition 2.1.2. Given elevations $y_1, \ldots, y_n$, a ***local minimum*** (symmetric for ***local maximum***) of a 1.5D terrain is a plateau whose points are lower than the points that are outside but adjacent to the plateau. That is, it is a collection of points $p_i, \ldots, p_j$ so that $y_i = \cdots = y_j$, $y_i < y_{i-1}$ if $i \neq 1$, and $y_j < y_{j+1}$ if $j \neq n$.

This definition means we will count adjacent points that have the same elevation only once. In 1.5D, we can identify each extremum by its rightmost point.

Definition 2.1.3. Given elevations $y_1, \ldots, y_n$ and the resulting 1.5D terrain, p_j is a ***local minimum [maximum] identifier*** if p_j is the rightmost (maximum index) point of the local minimum [maximum] p_j belongs to.

Objective #1: Minimizing the number of local minima and maxima (identifiers).

See Figure 2.2 for a potential input and a sample solution. The first and last points are always part of an extremum by definition, and the last point is always its own extremum's identifier. The first and last point may in fact be part of the same extremum if there is

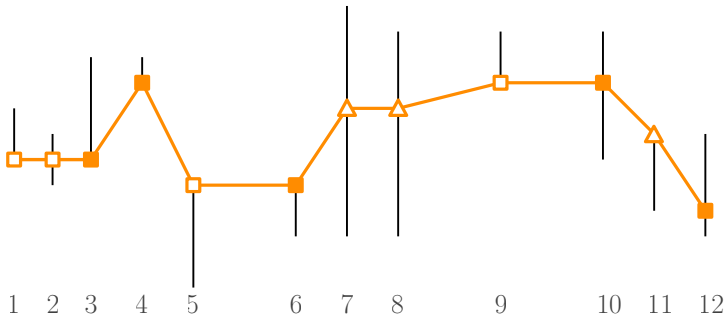


Figure 2.2: A 1.5D terrain with 5 local extrema. We associate each extremum with its rightmost point (the filled-in squares). Points represented with hollow triangles are not part of an extremum. This solution is optimal: because $t_2 < b_4 > t_5 < b_9 > t_{11}$, we require 5 extrema by Claim 2.1.1.

a single plateau solution covering all of the segments. The extremum is considered both a local minimum and maximum in this case, but we only count it as one extremum.

Our result for this problem is:

Theorem 2.1.1. There exists an $O(n)$ time algorithm to minimize the number of extrema for an imprecise 1.5D terrain.

Gray et al. [22] showed that the problem of minimizing the total number of extrema is NP-hard for 2.5D terrains. One can also ask to minimize the total number of minima instead of asking to minimize the total number of extrema. Gray et al. [22] show that this problem can be solved in $O(n \log n)$ time, even for 2.5D terrains. Because the extrema alternate between minima and maxima in a 1.5D terrain, it is not too hard to use this algorithm to solve the problem of minimizing the total number of extrema in $O(n \log n)$ time. Of course as we will see, we can improve the time complexity.

Preliminaries. First, we will prove a claim that helps establish a lower bound on the minimum number of extrema.

Claim 2.1.1. Let $k \geq 1$. Let $l_1, \ldots, l_k$ be a subsequence of $[n]$ where $t_{l_{j-1}} < b_{l_j} > t_{l_{j+1}}$ for all even $j < k$. Then the number of extrema is lower bounded by k.

The statement is also true if instead we assume $b_{l_{j-1}} > t_{l_j} < b_{l_{j+1}}$ for all even $j < k$.

Proof. The claim is trivial for $k = 1$. For $k = 2$, since $t_{l_1} < b_{l_2}$, it is not possible to make a single plateau solution, so there must be at least two extrema. Let $k \geq 3$. Let y be an arbitrary solution. See Figure 2.2 for a running example. We note that $t_2 < b_4 > t_5 < b_9 > t_{11}$ for this input.

Because $t_2 < b_4 > t_5$, we must have $y_2 < y_4 > y_5$, therefore we must have a local maximum *strictly* between points p_2 and p_5. The figure above illustrates one example of

this, where p_4 is the maximum, although there are alternatives (for instance, p_3, p_4 can both be part of a maximum, or p_3 could be the maximum). In general, we have $y_{l_{j-1}} < y_{l_j} > y_{l_{j+1}}$ for all even $j < k$, so there is at least one maximum strictly between l_{j-1} and l_{j+1}. So there are at least $\lfloor k/2 \rfloor - 1$ extrema. Crucially, given the maximum from when $j = 2$, we see that this maximum is not the first extremum, because it is *strictly* between l_1 and l_3, so there is at least one extremum before this maximum.

Also, since $b_{l_{j-1}} > t_{l_j} < b_{l_{j+1}}$ for all odd $j > 1$, we have that $y_{l_{j-1}} > y_{l_j} < y_{l_{j+1}}$. Therefore, there is at least one minimum contained strictly between indices l_{j-1} and l_{j+1}. Going over all odd j, we have found at least $\lceil k/2 \rceil - 1$ new extrema. We also count the extremum containing the last point, which comes after the one that exists strictly between l_{k-2} and l_k.

So in total, we have found $\lfloor k/2 \rceil + \lfloor k/2 \rfloor = k$ distinct extrema. □

Greedy algorithm. We will now provide a linear time greedy algorithm for this problem. The solution computed by the algorithm will be denoted by y. Let m denote the number of extrema of y, and let $i_1, \ldots, i_m$ be the identifiers of y.

The intuition behind the algorithm is starting from left to right, we place the current local extremum's identifier as far right as possible. This way, we ensure in the long run the total number of local extrema is minimized. See Algorithm 1 for the pseudocode.

We will now give an overview of the algorithm. On line 1, we place the first extremum identifier at $i_1 = \max\{i \in [n] : \min\{t_1, \ldots, t_i\} \geq \max\{b_1, \ldots, b_i\}\}$. This is as far right as we can place the first identifier. We declare a binary variable *dir*, which denotes whether to choose the elevation of the next points in an increasing or decreasing fashion. We initially define *dir* to be decr (decreasing) if the point p_{i_1+1} has to be below p_{i_1} (condition on line 2), or define *dir* to be incr (increasing) if the point p_{i_1+1} has to be above p_{i_1} (condition on line 4). We continue to place points in a decreasing (or increasing) fashion until it is no longer possible. At this time, we find where to place the next identifier i_2. As long as we choose points to increase as little as possible (when increasing) or to decrease as little as possible (when decreasing), we can guarantee that i_2 is as far away from i_1 as possible. We continue to place identifiers in this greedy fashion, alternating between maximum and minimum identifiers. Also throughout the algorithm we declare variables $l_1, \ldots, l_m$ that will be used when we appeal to Claim 2.1.1 for correctness.

We see variable *dir* flips from incr to decr when we switch from placing points in an increasing fashion to a decreasing fashion (and visa versa). Also, j increases by 1 during these flips. In particular, the conditions for when variable *dir* flips are given on lines 15 and 22. What do these conditions mean?

Consider an iteration i between $i_j + 1$ and i_{j+1}, and assume *dir*=decr without loss of generality. On line 14, we set y_i to be as high as possible, so long as it is less than or equal to the previous point y_{i-1} (hence $y_i = \min(y_{i-1}, t_i)$). On line 15 during iteration i_{j+1}, we see that $y_{i_{j+1}} < b_{i_{j+1}+1}$, and so it is not possible to set $y_{i_{j+1}+1}$ to be below $y_{i_{j+1}}$, despite the fact

Algorithm 1 Minimizing the number of extrema

$i_1 \leftarrow \max\{i \in [n] : \min\{t_1, \ldots, t_i\} \geq \max\{b_1, \ldots, b_i\}\}$
if $t_{i_1+1} < \max\{b_1, \ldots, b_{i_1}\}$ **then**
    $dir \leftarrow$ decr
else if $b_{i_1+1} > \min\{t_1, \ldots, t_{i_1}\}$ **then**
    $dir \leftarrow$ incr
end if
for $i = 1$ up to i_1 **do**
    $y_i \leftarrow \min\{t_1, \ldots, t_{i_1}\}$ if $dir =$ incr, otherwise $y_i \leftarrow \max\{b_1, \ldots, b_{i_1}\}$
end for
Let $l_1 \in \{1, \ldots, i_1\}$ such that $y_{i_1} = t_{l_1}$ if $dir =$ incr, otherwise $l_1 \in \{1, \ldots, i_1\}$ such that $y_{i_1} = b_{l_1}$
$j = 1$
for $i = i_1 + 1$; $i \leq n$; i += 1 **do**
    if $dir =$ decr **then**
        $y_i \leftarrow \min(y_{i-1}, t_i)$ ▷ Invariant: $y_i = \min\{t_{i_j+1}, \ldots, t_i\}$
        if $i \neq n$ and $y_i < b_{i+1}$ **then**
            $dir =$ incr
            j += $1, i_j = i$
            Let $l_j \in \{i_{j-1} + 1, \ldots, i_j\}$ such that $y_{i_j} = t_{l_j}$
        end if
    else if $dir =$ incr **then**
        $y_i \leftarrow \max(y_{i-1}, b_i)$ ▷ Invariant: $y_i = \max\{b_{i_j+1}, \ldots, b_i\}$
        if $i \neq n$ and $y_i > t_{i+1}$ **then**
            $dir =$ decr
            j += $1, i_j = i$
            Let $l_j \in \{i_{j-1} + 1, \ldots, i_j\}$ such that $y_{i_j} = b_{l_j}$
        end if
    end if
end for
$i_j \leftarrow n, m \leftarrow j$
return y ▷ y has m local extrema

we placed $y_{i_{j+1}}$ as high up as possible. Therefore, we must flip variable *dir* before starting the next iteration.

Correctness. First, we elaborate on comments made in the algorithm (lines 14 and 21):

Invariant 2.1.1. Assume $b_{i_1+1} > \min\{t_1, \ldots, t_{i_1}\}$ without loss of generality. Then

- $y_i = \max\{b_{i_j+1}, \ldots, b_i\}$ for all $i \in \{i_j + 1, \ldots, i_{j+1}\}$ if j odd.
- $y_i = \min\{t_{i_j+1}, \ldots, t_i\}$ for all $i \in \{i_j + 1, \ldots, i_{j+1}\}$ if j even.

If instead $t_{i_1+1} < \max\{b_1, \ldots, b_{i_1}\}$, we swap the odd and even cases. This is a pretty simple proof by induction. The invariant itself is not too important, but the main takeaway is that what is done on lines 18 and 25 makes sense. We now give the proof of correctness for the algorithm.

Proof of correctness. Consider $i_1, l_1, \ldots, i_m, l_m$ computed by the algorithm. Clearly, y has m extrema. We will use Claim 2.1.1 to prove this is optimal.

Assume with loss of generality that $b_{i_1+1} > \max\{t_1, \ldots, t_{i_1}\}$. So the parity variable *dir* starts off being equal to incr, and *dir* = incr if and only if the variable j is odd. By lines 10, 18 and 25, $y_{i_j} = t_{l_j}$ if j is odd, and $y_{i_j} = b_{l_j}$ if j is even, and clearly $l_1 < l_2 < \cdots < l_m$.

It is also clear the algorithm chooses $y_{i_1}, y_{i_2}, \ldots, y_{i_m}$ as the extrema identifiers, so $y_{i_1} < y_{i_2} > y_{i_3} < y_{i_4} > \cdots$, and so $t_{l_1} < b_{l_2} > t_{l_3} < b_{l_4} > \cdots$. By Claim 2.1.1, there are at least m extrema for this input. Hence, y is optimal. □

2.2 Minimizing the number of links/bends

On any 1.5D terrain, we have $n - 1$ segments connecting the n points of the terrain. If a subsequence of the segments all lie on a line, we can really think of them as one long segment. We will refer to such a subsequence as a ***link*** of the terrain.

Objective function #2a: Minimizing the number of links.

Note: we will later generalize the notion of a link to a ***patch*** (see Section 4.2).

An equivalent way to phrase this objective function is to minimize the number of turn angles that are *not* equal to 0. We call the points where such turn angles happen ***bends***.

Objective function #2b: Minimizing the number of bends.

For a precise 1.5D terrain, the number of links is equal to the number of bends plus one.

We will prove the problems are in NP. See Section 2.2.1. The problems appear simple enough that they do not appear to be NP-hard, however efforts to find an exact algorithm were not successful. However, it is easy to provide a 2-approximation algorithm.

Theorem 2.2.1. There exists a linear time 2-approximation algorithm for the problem of minimizing the number of links (or bends).

The first problem is similar to the problem of finding a minimum link distance path in a polygon, for which Suri [44] gave a linear time algorithm. In Section 2.2.2, we will use Suri's algorithm to give a linear time 2-approximation algorithm. In Section 2.2.3, we will provide another 2-approximation (greedy) algorithm, which will inspire a similar approach for a special case in the 2.5D model (Section 4.2.2). We will prove both algorithms 2-approximate the optimal number of bends, which also proves that the algorithms 2-approximate the optimal number of links. In Section 2.2.4, we briefly compare the two algorithms. We give two example inputs where one algorithm returns a better solution than the other algorithm.

Preliminaries. Both algorithms build upon an algorithm by Suri [44]. The problem proposed by Suri is the following: given a simple polygon, compute a path between vertices u and v with as few links as possible. Given the polygon has n vertices, the runtime of this algorithm is $O(n)$ time.

Before we discuss how we use his algorithm, we will give a brief summary of how Suri's algorithm works. First, the algorithm finds the visibility polygon $V(u)$ of u. If v is not visible from u, then more than one link is needed. Next, an edge of $V(u)$ is computed, which is a chord of P, denoted the window $w(u)$. In successive steps his algorithm finds the visibility polygon of the window of the previous visibility polygon to get a new window. The algorithm stops when the current visibility polygon includes the vertex v.

Observation 2.2.1. Suri's algorithm can be modified to work with starting and ending segments e_1, e_2 of P, instead of vertices of P.

In particular, we will use this modification to compute the shortest link path from ℓ_1 to ℓ_n in $P_{1,n}$. We will now prove this result. After proving this observation, we found a result from Imai and Ira [27] that (implicitly) provides an algorithm to compute the path with the minimum link distance between two vertical edges of a polygon. They prove this result from scratch, but we will simply modify Suri's algorithm.

Proof. Because e_1 is a chord of P, and because the algorithm iteratively computes visibility polygons from chords of P, the algorithm can start with the chord e_1 instead of some vertex a. Furthermore, we can easily check whether e_2 intersects the current visibility polygon instead of whether some point b intersects the visibility polygon. □

We have a second algorithm, which needs to compute one link paths efficiently.

Observation 2.2.2. Let $i < n$. We can compute the largest index $j > i$ such that we can connect ℓ_i to ℓ_j with a line segment through $P_{i,n}$ in $O(j - i)$ time.

Proof. We take a similar approach to Suri's algorithm. We simply compute the visibility polygon starting from ℓ_i within polygon $P_{i,n}$, and find the largest ℓ_j that intersects the visibility polygon. As discussed by Suri: "Instead of spending $O(n)$ time for computing each $V(e_i)$ [the i'th visibility polygon] we spend only $O(n_i)$ time, where n_i is the total number of triangles of T [the triangulation] that are intersected by $V(e_i)$". Therefore, the runtime of computing a one link solution is $O(j - i)$. □

2.2.1 A decision algorithm when the bend locations are known

If we know the indices of where we want the bends, then there is a simple linear program that computes a feasible solution. Assume we are told that there are k links with the $k - 1$ bends at indices $i_1, i_2, \ldots, i_{k-1}$ where $i_0 = 1 < i_1 < \cdots < i_{k-1} < i_k = n$. This can be solved by finding a feasible solution to a linear program. Let $y_1, .., y_n$ be variables. We add variables $m_1, c_1, \ldots, m_k, c_k$ to capture the equations of the lines of the links. A precise 1.5D terrain is a feasible solution to the following linear program:

- $b_i \leq y_i \leq t_i$ for all $i \in [n]$
- $y_i = m_{i_j} x_i + c_{i_j}$ for all i s.t. $i_{j-1} \leq i \leq i_j$ and for all $j = 1, \ldots, k$

This can be helpful for deciding whether there is a solution with at most k links, assuming $k = O(1)$ or $k = n - O(1)$. We simply try all $\binom{n}{k-1}$ possible places for the bends, and solve the LP, and return any feasible solution.

Additionally, this linear program can be used to show that the problem lies in NP. The bend locations can be computed non-deterministically, then the y-values are computed using the linear program.

2.2.2 The polygon algorithm

We will now examine a linear time 2-approximation algorithm for this problem. It is a simple 3-step algorithm. First, construct the polygon $P_{1,n}$. Next, compute the *minimum link path* between the first vertical line segment and the last vertical line segment that is contained within this polygon. We will use the modified version of Suri's linear time algorithm (as discussed in Observation 2.2.1). There is no guarantee that the links obtained from this algorithm bend at the vertical segments, which means it does not necessarily translate to a 1.5D terrain.

The final step is to convert this path into a proper 1.5D terrain. From left to right, replace every bend that is strictly in-between two segments with two new bends, one at each intersection between the piecewise linear curve and the two vertical segments. If there already is a bend at either of those points, we do not need to add new bends, just use the ones already there. This ensures all bends occur at vertical intervals. We at worst double the number links on Suri's path. Since every 1.5D terrain is a feasible solution to Suri's problem, then the number of links in the optimal solution to Suri's problem is a lower bound to our optimal solution, and therefore, we clearly have a 2-approximation. This approach clearly takes $O(n)$ time.

2.2.3 The greedy algorithm

We will now give a second 2-approximation algorithm. This algorithm will inspire another algorithm used in Section 4.2.2.

This is a simple greedy algorithm. We choose the first link so that it covers as many segments as possible. Let us say we can cover the first i_1 segments $\ell_1, \ldots, \ell_{i_1}$ with one link. We compute the y values $y_1, \ldots, y_{i_1}$ within the first i_1 intervals $I_1, \ldots, I_{i_1}$ in linear time (see Observation 2.2.2).

We continue this process starting at segment ℓ_{i_1+1} to determine the remaining $n - i_1$ elevations. (Note that once y_{i_1+1} has been decided, we need p_{i_1+1} to connect to p_{i_1}, so we add a link connecting the two points. Potentially we can merge this short link with the next link, but that may not always be possible). From ℓ_{i_1+1}, we compute the farthest index i_2 we can reach with one link from segment $i_1 + 1$. We then start from segment $i_2 + 1$ to find the farthest index i_3 we can reach, and so on and so forth. We keep going until we reach an iteration k where $i_k \leq n - 1$.

If $i_k = n - 1$, we place a point p_n anywhere on ℓ_n and connect a link from p_{n-1} to p_n. This case results in at most $2k$ links. The other case is $i_k = n$, in which case we do not need to add an extra short link, so there is at most $2k - 1$ links.

Claim 2.2.1. Let $i_0 = 1$. Let $i_1, \ldots, i_{k-1}, i_k$ denote the indices from the description of the algorithm. Then the problem requires at least $k - 1$ bends (k links). (If $i_k \neq n$, then an additional bend is required).

Proof. Consider an arbitrary solution y. Let $j = 1, \ldots, k - 1$. Since it is not possible to cover line segments $\ell_{i_{j-1}}, \ldots, \ell_{i_j}$ with one link, then there must be at least 1 *bend* between $i_{j-1} + 1$ and i_j. Therefore, the input requires $k - 1$ *bends*, so the problem requires at least k links. If $i_k = n - 1$, we require an additional bend between $i_{k-1} + 1$ and i_k. □

Let OPT denote the optimal number of bends. If $i_k = n$, then there are at most $2k - 2$ bends, and the input has at least $k - 1$ bends by Claim 2.2.1. If $i_k = n - 1$, then there are at most $2k - 1$ bends, and the input has at least k bends by Claim 2.2.1. In both cases it is clearly a 2-approximation algorithm for minimizing the number of bends.

Runtime: The step that involves computing a one link solution between $\ell_{i_{j-1}}$ and ℓ_{i_j} can be done in $O(i_j - i_{j-1})$ time (see Observation 2.2.2). Therefore in total, this algorithm takes $O(n)$ time.

2.2.4 A brief comparison between the two algorithms

Both algorithms are guaranteed to return a 2-approximation of the optimal terrain. For some inputs, the polygon algorithm (from Section 2.2.2) returns a better solution than the greedy algorithm (from Section 2.2.3), see Figure 2.3 for an example input.

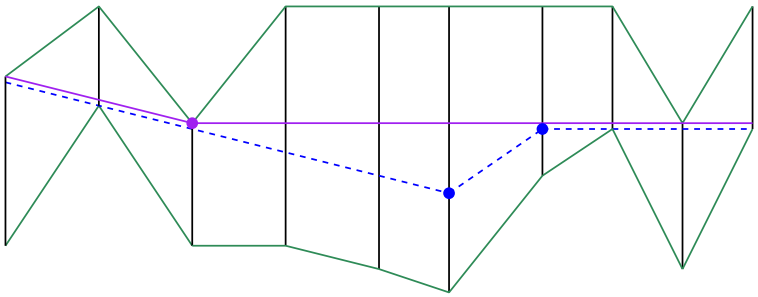


Figure 2.3: An example input where the polygon algorithm achieves a better solution than the greedy algorithm. The solid purple terrain is the solution returned by the polygon algorithm, which is the optimal solution. The dashed blue terrain is the solution returned by the greedy algorithm. The two terrains overlap at the start and end, but are drawn slightly apart for better visualization.

Meanwhile, there are inputs where the greedy algorithm achieves a better solution than the polygon algorithm. See Figure 2.4.

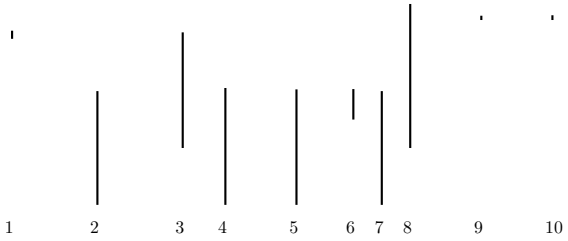
2.3 Minimizing the length of the terrain

A very natural objective function is to minimize the sum of the lengths of the segments of the 1.5D terrain.

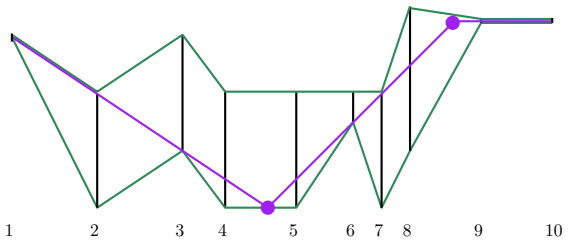
Objective #3: Minimizing the total length of the terrain.

Theorem 2.3.1. There is a linear time algorithm to compute a 1.5D terrain that minimizes the total length.

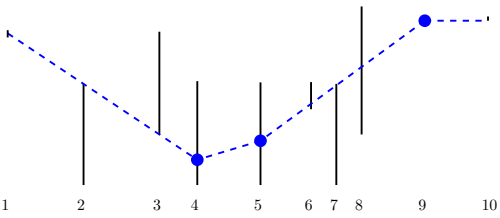
We reduce this problem to finding a shortest path between two edges in a polygon. Chiang and Tamassia [10] solve a more general problem of finding a shortest path between two convex polygons inside a bigger simple polygon. In fact, they give an algorithm for a



(a) A sample input. The first and last two segments are 0 length.



(b) The unique shortest link distance path in $P_{1,n}$. Therefore, the polygon algorithm will return a terrain with 4 bends, 5 links.



(c) The output of the greedy algorithm. For this input, this algorithm returns an optimal solution.

Figure 2.4: An example input where the greedy algorithm returns a better solution than the polygon algorithm.

query version of the problem but we only need the shortest path between two fixed edges. Given two convex polygons C_1, C_2 within a simple polygon P, their algorithm computes the shortest path from C_1 and C_2 within P in $O(n + \log h)$ time, where n is the number of vertices of P, and h is the number of vertices of C_1 and C_2.

Using this result, we will simply compute the shortest path π from ℓ_1 to ℓ_n within polygon $P_{1,n}$ (see Definition 2.0.1) in $O(n)$ time. For all $i = 1, \ldots n$, the value we return for $p_i = (x_i, y_i)$ is simply the unique point in the intersection of ℓ_i and π.

We observe that the shortest path π must be made up of $O(n)$ links, whose turns can only happen at vertices of the polygon, see Chapter 15 of [12] (Lemma 15.1). Therefore, the 1.5D terrain obtained from $p_1, \ldots, p_n$ is exactly π. Since π is optimal, then the computed 1.5D terrain is also optimal.

2.4 Optimizing steepness

Let the ***steepness*** of an edge of the terrain be the absolute value of its slope. We can solve the problem of *minimizing the maximum steepness*.

Objective #4a: (Lexicographically) minimizing the maximum steepness.

We will also explore the problem of *maximizing the minimum* steepness, a problem that has relevance for dispersion problems.

Objective #4b: (Lexicographically) maximizing the minimum steepness.

What does lexicographically mean in these contexts? For a given 1.5D terrain, define the steepness vector of the terrain to be the vector of $n-1$ steepness values obtained from the $n-1$ edges of the curve (note that there might be duplicate values).

For the min-max objective function, the vector will be sorted in *decreasing* order. For two different terrains T, T' with steepness vectors s, s', we say that T is *lexicographically less steep* than T' if $s \neq s'$ and for the first index i where s differs from s', $s_i < s'_i$. We want to return the terrain that is lexicographically the least steep.

For the max-min objective function, the vector will be sorted in *increasing* order. For two different terrains T, T' with steepness vectors s, s', we say that T is *lexicographically steeper* than T' if $s \neq s'$ and for the first index i where s differs from s', $s_i > s'_i$. We want to return the terrain that is lexicographically steepest.

The max-min objective is not a very natural measure for the niceness of a terrain, because the terrain will have sharp turns when switching from steep increasing segments to steep decreasing segments. However, this problem is relevant to Section 3, which asks to place a single point on each vertical segment in order to maximize the minimum pairwise distances.

Our results for this section are:

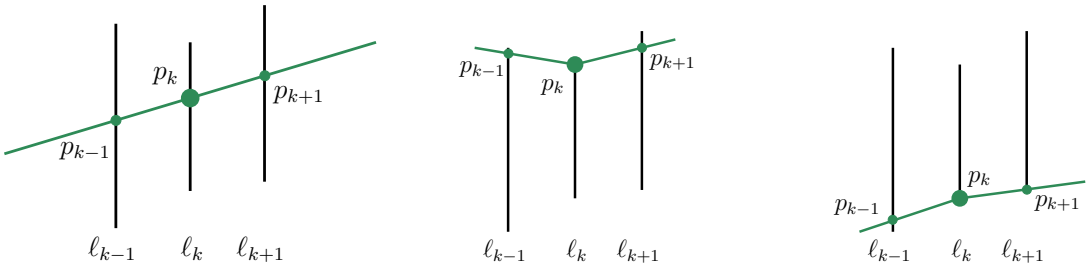
Theorem 2.4.1. There is an $O(n)$ time algorithm to lexicographically minimize the maximum steepness vector of an imprecise 1.5D terrain.

Theorem 2.4.2. There is a dynamic programming algorithm to lexicographically maximize the minimum steepness vector of an imprecise 1.5D terrain in $O(n^3)$ time.

Theorem 2.4.1 will be proven in Section 2.4.1, and Theorem 2.4.2 will be proven in Section 2.4.2.

2.4.1 Minimize the maximum steepness

We first prove Theorem 2.4.1. The algorithm is quite simple. We return the 1.5D terrain with the minimum total length. As discussed in Section 2.3, this can be done in linear time.



(a) The terrain goes straight through the segment ℓ_k.

(b) The terrain turns up at the top endpoint (x_k, t_k).

(c) The terrain turns down at the bottom endpoint (x_k, b_k).

Figure 2.5: The three cases for how a locally shortest path can cross a line segment.

Correctness. Let y be the output to this algorithm. As discussed in Section 2.3, the terrain with the minimum total length is obtained from the shortest path π from ℓ_1 to ℓ_n in polygon $P_{1,n}$.

Definition 2.4.1. A path is locally shortest if it cannot be made shorter by changing the path in a small neighbourhood.

Specifically for polygon $P_{1,n}$, this means that the path is piecewise linear and for each $k = 1, \ldots, n$, one of the following holds:

1. The terrain goes straight through the segment ℓ_k.
2. The terrain turns up at the top endpoint (x_k, t_k).
3. The terrain turns down at the bottom endpoint (x_k, b_k).

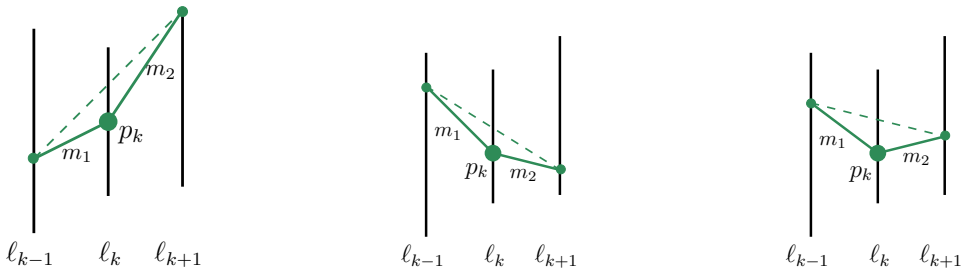
See Figure 2.5 for an illustration of these three cases.

It is commonly known that shortest between two points in polygon is unique. In our setting, we are looking at shortest paths between two segments. It is not hard to argue that the shortest path π in $P_{1,n}$ is unique and is locally shortest—with one exception when the shortest path is a horizontal line segment, which we might be able to shift up or down.

We will prove the following lemma in order to prove correctness.

Lemma 2.4.1. If a path π^* lexicographically minimizes the maximum steepness vector, then it is locally shortest path.

Correctness easily follows from this. If the shortest path is unique, then it is the *unique* locally shortest path, therefore, only the shortest path lexicographically minimizes the maximum steepness following Lemma 2.4.1. Otherwise, the shortest path is a horizontal segment so the steepness vectors is all 0, so the shortest path lexicographically minimizes the maximum steepness in this case as well.



(a) Case (a). (b) Case (b). (c) Case (c).

Figure 2.6: Moving point p_k up in all cases will improve the steepness vector. The dashed line represents the straight path from p_{k-1} to p_{k+1}.

Proof. We prove the contrapositive. Assume that π^* is not locally shortest. Consider a bend that turns up and that can be locally shortened. (The "turning down" case can be argued in a similar fashion). Let us say the bend is made up of points p_{k-1}, p_k, p_{k+1} on π^*. See Figure 2.6.

We can move p_k up by a small amount up to "straighten" the bend, which is a local adjustment that improves the length of the path. Also, we note that straightening this bend will cause a change in the slopes of the two edges forming this bend. We denote the two slopes m_1, m_2, see Figure 2.6. We break down into cases.

(a) If both slopes are positive, then the smaller slope value (m_1) will increase while the larger one (m_2) will decrease. See Figure 2.6a. By decreasing m_2 by a small amount (so long as we still have $m_1 \leq m_2$), we will improve the steepness vector, so π^* does not lexicographically minimize the maximum steepness.

(b) If both slopes are negative, then it is symmetric to case (a), so π^* does not lexicographically minimize the maximum steepness.

(c) If the first slope is negative and the second slope is positive, then the first slope will increase and the second one will decrease. See Figure 2.6c. Both slopes are less steep than before, so π^* does not lexicographically minimize the maximum steepness.

(d) It is not possible for the first slope to be positive and the second slope to be negative while "turning up".

In all cases, π^* does not lexicographically minimize the maximum steepness. □

2.4.2 Maximizing the minimum steepness

We will now prove Theorem 2.4.2 by giving a dynamic programming algorithm to lexicographically maximize the minimum steepness vector.

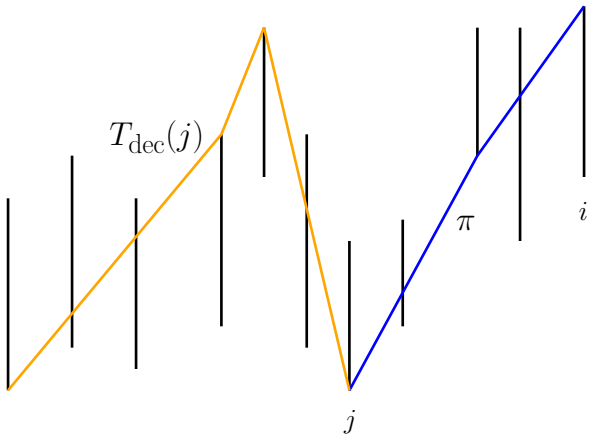


Figure 2.7: Illustrating how to compute a candidate solution $T^{(j)}$ for $T_{\text{inc}}(i)$. One example for j is given with this figure, but we must try all possible indices j (for where the last increasing subsequence starts).

Algorithm overview. This is a dynamic programming algorithm. Any terrain consists of alternating increasing (non-decreasing) and decreasing (non-increasing) sequences. When our goal is to maximize steepness, every increasing sequence, say from index j to k, should go from the bottom endpoint b_j to the top endpoint t_k (formalized as Observations 2.4.1, 2.4.2 below). This motivates solving two subproblems for each i—to compute the optimal terrain $T_{\text{inc}}(i)$ from 1 to i that ends with an increasing subsequence, and to compute the optimal terrain $T_{\text{dec}}(i)$ from 1 to i that ends with an decreasing subsequence.

Going forward, we will focus on how to compute $T_{\text{inc}}(i)$, the other case is symmetric. To solve each subproblem $T_{\text{inc}}(i)$, we consider all possibilities $j < i$ for the index where the increasing subsequence starts. Since the subsequence before this last subsequence is decreasing, we use the solution $T_{\text{dec}}(j)$ for the first j y-values. Given j, we compute an increasing set of y-values $y_j, \ldots, y_i$ from intervals I_j to I_i. To determine these last $i - j$ elevation values, we perform a shortest path computation from (x_j, b_j) to (x_i, t_i) and return the y-values where the path intersects the line segments. In Figure 2.7, we show an example for one value of j.

Algorithm details. Each $T_{\text{inc}}(i)$ will be a tuple. The first entry is $y^*_{\text{inc}}(i)$, which is a list of the i y-values representing the optimal solution for this subproblem. (We could just store the y-values from j to i for the optimal j, but this makes no difference to the asymptotic time or space complexity.) The second entry is $s_{\text{inc}}(i)$, which is the objective value for this solution, namely an increasing list of the $i-1$ steepness values for the terrain obtained from $y^*_{\text{inc}}(i)$. In a similar fashion, $T_{\text{dec}}(i)$ is the tuple $(y^*_{\text{dec}}(i), s_{\text{dec}}(i))$.

For the base case we set $y^*_{\text{inc}}(2) = (b_1, t_2)$. If $b_1 > t_2$, then $y^*_{\text{inc}}(2)$ should be set to null, as no solution exists with $y_2 \geq y_1$. $s_{\text{inc}}(2)$ is simply $\frac{|t_2 - b_1|}{x_2 - x_1}$. Similarly set $y^*_{\text{dec}}(2) = (t_1, b_2)$ and $s_{\text{dec}}(2) = \frac{|b_2 - t_1|}{x_2 - x_1}$ (or null if $t_1 < b_2$). Note that $T_{\text{inc}}(1)$ and $T_{\text{dec}}(1)$ are not considered because

steepness is not well defined when a terrain consists of only one point.

To compute $T_{\text{inc}}(i)$ for $i \geq 3$, we iterate over j as suggested in the algorithm overview, and for each j we generate one candidate solution $T^{(j)}_{\text{inc}}(i)$. For simplicity of notation, we will instead use the notation $T^{(j)} = (y^{(j)}, s^{(j)})$.

The first j y-values of $y^{(j)}$ are obtained using a lookup to $T_{\text{dec}}(j)$. To lexicographically maximize the minimum steepness for the last $i - j + 1$ points, we borrow an idea used by Biedl et al [3]. We determine the y-values by computing the shortest path from (x_j, b_j) to (x_i, t_i) within the polygon $P_{i,j}$ (see Definition 2.0.1), and return the y-values where the path intersects the line segments. See Figure 2.7 for an example. For this step, we will use Guibas' [25] linear time shortest path algorithm. If the y-values of the shortest path are not non-decreasing, we can conclude it is never possible to make the elevations for these points non-decreasing, so we do not return a solution $T^{(j)}$ from iteration j.

To compute $s^{(j)}$, we use the sorted list of steepness values from $s_{\text{dec}}(j)$ and obtain the remaining $i - j$ steepness values from the shortest path.

After we generate all candidate solutions $T^{(1)}, \ldots, T^{(j)}$, we return the one that is the best. That is, return the one that results in the lexicographically steepest terrain, by comparing the steepness vectors $s^{(1)}, \ldots, s^{(i-1)}$.

Correctness. We make the following observations about the optimal solution:

Observation 2.4.1. For all $i = 2, \ldots, n$, $T_{\text{inc}}(i)$ must have its i'th y-value equal to t_i.

Observation 2.4.2. For all $i = 2, \ldots, n$, for any index j where the terrain $T_{\text{inc}}(i)$ switches from decreasing to increasing, the j'th elevation value must be b_j.

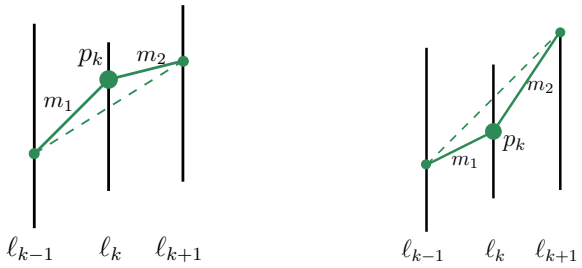
The proofs of these observations are straightforward. If either observation was not true, we could easily shift point i up or point j down and improve the steepness vector.

We next prove a lemma for computing the last $i - j + 1$ elevations for a given i and iteration $j < i$.

Lemma 2.4.2. Let $j < i$. Let π denote the (unique) shortest path from (x_j, b_j) to (x_i, t_i) in P_{ij}. Then the optimal 1.5D terrain for the subproblem $\ell_j, \ldots, \ell_i$ with the restriction $y_j \leq \cdots \leq y_i$ should have point p_k placed on the unique intersection between π and ℓ_k, for all $k = j, \ldots, i$.

Proof. The proof of this lemma is similar to that of Claim 2.1 from Biedl et al. [3]. We will give our own proof because this problem is in a different context than theirs (dispersing points along a real line).

Let π^* denote the path that is obtained from the optimal terrain for the subproblem $\ell_j, \ldots \ell_i$ with the restriction $y_j \leq \cdots \leq y_i$. We will prove $\pi^* = \pi$, which clearly shows the lemma holds. We will prove it in a similar fashion as Lemma 2.4.1.



(a) Case (a). (b) Case (b).

Figure 2.8: Pushing point p_k down (Case (a)) or up (Case (b)) will straighten the path. The smaller of the two slopes increases, so the steepness vector improves with the movement of p_k.

Assume for contradiction π^* is not locally shortest. As in the proof of Lemma 2.4.1, we consider a bend that turns up and that can be locally shortened. Let us say the bend is made up of points p_{k-1}, p_k, p_{k+1} on π^*, see Figure 2.8. Case (a) is the upward bend case, and Case (b) is the downward bend case.

We can move p_k slightly up or down to try to "straighten" the bend. We note straightening this bend will cause a change in the slopes of the two edges forming this bend. The slopes are denoted by m_1, m_2. By assumption, both slopes are positive, so then the smaller slope value will increase while the larger one will decrease. By increasing the smaller one by a small amount (so long as we maintain that the smaller slope is less than or equal to the larger slope), we will improve the resulting steepness vector, which contradicts π^* maximizing the minimum steepness.

Therefore, π^* must be locally shortest. Since π is the unique shortest path, then it is the unique locally shortest path, so $\pi^* = \pi$, proving the lemma. □

We will now prove $T_{\text{inc}}(i)$ and $T_{\text{dec}}(i)$ are computed correctly by induction.

Base case. The base cases $T_{\text{inc}}(2)$ and $T_{\text{dec}}(2)$ are both clearly correct.

Inductive step. Let $3 \leq i \leq n$. We will focus on computing $T_{\text{inc}}(i)$, computing $T_{\text{dec}}(i)$ can be proven in a symmetric fashion. Assume by induction $T_{\text{inc}}(2), T_{\text{dec}}(2), \ldots, T_{\text{inc}}(i-1), T_{\text{dec}}(i-1)$ have been computed correctly.

Note that there is some index j where the optimal terrain switches from decreasing to increasing. Chose the largest such j, unless the terrain is never decreasing, in which case we let $j = 1$. By Observation 2.4.2, the j'th y-value must be b_j.

With this observation, the first $j-1$ steepness values are independent of the last $i-j$ steepness values. Therefore, the $j-1$ steepness values should simply be the steepness values for the optimum solution $T_{\text{dec}}(j)$, which by induction has been computed correctly. Knowing

the terrain is non-decreasing from j to i, the last $i-j$ steepness values should be obtained from a shortest path from (x_j, b_j) to (x_i, t_i) by Lemma 2.4.2. This solution being described is exactly the solution $T^{(j)}$ (defined above, see the Algorithm details section).

Finally, since $T_{\text{inc}}(i)$ is simply computed to be the best of $T^{(1)}, \ldots, T^{(i-1)}$, then clearly $T_{\text{inc}}(i)$ has been computed correctly.

Implementation and runtime. We note that there are at most $2n$ subproblems to solve. We will show $T_{\text{inc}}(i)$ takes $O(i^2)$ time to compute, which results in the total runtime being $O(n^3)$ time. Recall that $T_{\text{inc}}(i)$ is set to be one of the candidates $T^{(1)}, \ldots, T^{(i-1)}$.

A straightforward solution is to compute each $T^{(j)} = (y^{(j)}, s^{(j)})$, $j = 1, \ldots, i-1$, separately. We look up the first j elevations from $y^*_{\text{dec}}(j)$ and we use Guibas' [25] linear time algorithm to compute the shortest path from (x_j, b_j) to (x_i, t_i) in order to obtain the last $i-j+1$ elevation values. The i elevation values give us the $i-1$ steepness values, which we can then sort in time $O(i \log i)$. Summing over all j gives a runtime of $O(i^2 \log i)$. We claim that this can be reduced to $O(i^2)$ as discussed in the next two paragraphs.

We want shortest paths from each (x_j, b_j) to (x_i, t_i). Since all of the shortest paths end at (x_i, t_i), we can simply compute the shortest path tree to all vertices *starting from* (x_i, t_i) in $O(i)$ time. Furthermore, we can compute the sorted steepness vectors of the paths as we compute the shortest path tree, as follows. Suppose an edge of steepness s is added to the shortest path tree. The edge passes *straight through* some number k of the vertical segments, and we create the steepness vector for the child in the shortest path tree by taking the sorted steepness vector for the parent and inserting $k+1$ copies of steepness value s for the new edge. This takes $O(i^2)$ time overall, since we are creating and storing a steepness vector of size $O(i)$ at each of the i nodes of the shortest path tree.

Finally, to compute $s^{(j)}$, we need the first $j-1$ steepness values (which we store sorted) and the last $i-j$ steepness values, which are sorted by the previous paragraph. We merge these two vectors in linear time to get $s^{(j)}$. This takes $O(i^2)$ time for all j.

Now that we have (sorted) steepness vectors $s^{(1)}, \ldots, s^{(i-1)}$, we need to determine the lexicographically steepest vector. For two sorted steepness vectors, comparing them takes $O(i)$ time. So, comparing all sorted steepness vectors to find the best solution takes $O(i^2)$ time.

Therefore, computing $T_{\text{inc}}(i)$ takes $O(i^2)$ time. In total, the runtime is $\sum_i O(i^2) = O(n^3)$. Each $T_{\text{inc}}(i)$ and $T_{\text{dec}}(i)$ takes $O(i)$ space. Therefore, the space complexity is $\sum_i O(i) = O(n^2)$.

Chapter 3

Distant Representatives for Segments in the Plane

In the previous chapter we considered problems of constructing "optimum" 1.5D terrains by placing one point in each input vertical line segment. In this chapter we consider another problem involving the imprecise point model, where one ***representative point*** is chosen within each input object. The problem we will be focusing on is the *distant representatives* problem first introduced by Fiala at al. [18]. The general problem statement asks: given objects in the plane, place one representative point per object in order to maximize the minimum distance between any pair of points.

We will examine the case where the input objects are line segments in the plane, which makes the problem quite similar to those of the previous chapter (Chapter 2). The most important difference with the distant representatives problem is that we want to have *all* points far apart from each other, whereas the problems for imprecise 1.5D terrains are concerned with properties of the edges of the terrain. Therefore, for this problem, we can allow the input line segments to overlap.

However, Objective #4b (maximizing the minimum steepness) for imprecise 1.5D terrains is somewhat similar to this problem (wanting points to be far apart), except there, only distances between adjacent segments were relevant.

This chapter will be structured differently from the previous chapter. The sections will involve adding small variations or assumptions to the input, in order to obtain different and interesting results. See the first column of Table 3.1 for the assumptions on the input. For instance, in Section 3.3, we will consider a very general case of inputs with horizontal segments as well as vertical segments, and in Section 3.2, we will restrict to unit parallel segments.

Overview. We have three results to present. Two of the results (the ones in Sections 3.2 and 3.3) were written for the paper "Distant Representatives for Rectangles in the Plane" by Biedl et al. [4]. As a major co-author of those results, we will present them in this thesis.

In Section 3.1, we will present an algorithm to solve the problem for parallel segments. The algorithm proves the problem belongs to the complexity class XP, where the parameter is the size of the strip that contains the segments. This approach only works in the L_1 and L_∞ norms, building off a lemma from Biedl et al. [4].

It would be nice to have an algorithm that proves the problem is in P and not just XP, however this is unlikely: in Section 3.2, we show this version of the problem is NP-hard. In fact, we can show this for the L_2 norm as well. Furthermore, the reduction uses *unit length* segments. This result appears in Section 4.1 of Biedl et al. [4], also see Appendix D.1 of the extended version of the paper [5]. An NP-hardness reduction has already been provided for the L_2 norm by Roeloffzen in his Master's thesis [41, Section 2.3], but we add details regarding the bit complexity of the segments constructed during the reduction that should have been addressed in his proof.

Finally, we will look at the more general case of orthogonal segments in Section 3.3. By allowing both horizontal and vertical segments, we can expand on the previous idea and get a gap-reduction that shows APX-hardness, that is, we can cannot even approximate within a constant factor unless P = NP. Our main result is that, assuming P $\neq$ NP, no polynomial time approximation algorithm achieves a factor better than 1.5 in L_1 and L_∞ and 1.4425 in L_2. This result appears in Section 4.2 of Biedl et al. [4], also see Appendix D.3 of the extended version of the paper [5].

The distant representative problem. The distant representatives problem is defined as follows:

Input: A set of horizontal and/or vertical line segments $\mathcal{L} = \{\ell_1, \ldots, \ell_n\}$.

Output: A set of points $p_1, \ldots, p_n$ where $p_i = (x_i, y_i)$ placed on line segment ℓ_i.

Objective function: Maximize the minimum pairwise L_∞, L_1 or L_2 distance between any pair of points, that is compute $\delta^*_\infty = \max_y \min_{i \neq j} d_\infty(p_i, p_j)$, $\delta^*_1 = \max_y \min_{i \neq j} d_1(p_i, p_j)$ or $\delta^*_2 = \max_y \min_{i \neq j} d_2(p_i, p_j)$.

When there is no need to distinguish between the optimum value for the three norms, we will simply use δ^* to denote the optimum value.

3.1 An XP algorithm for parallel line segments

In this section, we will be looking at inputs with only vertical segments. Building on a lemma from Biedl et al. [4], we can give an algorithm for the L_1 and L_∞ norms. The runtime will

Input Segments	Norm	Complexity Class
[§ 3.1] Parallel Segments	L_1, L_∞	XP
[§ 3.2] Unit Parallel Segments	L_1, L_2, L_∞	NP-hard
[§ 3.3] Orthogonal Segments	L_1, L_2, L_∞	APX-hard

Table 3.1: Results of Section 3.

be efficient in terms of n and a parameter h, so long as h is thought of as constant. We will now formally define our parameter h.

Assume that all input coordinates are integers, and not just rational as was the case for the segments in Section 2. Because the distant representatives problem is proportional to scaling and invariant to translation, we can simply take any rational input and transform to our desired integer coordinate system.

Assume the vertical line segments are sorted in non-decreasing x-order. Let us say $\ell_i = \{x_i\} \times I_i$, where $I_i = [b_i, t_i]$. Let h denote the difference in height between the smallest bottom endpoint and the largest top endpoint. In fact, without loss of generality we can assume that $\min(b_i) = 0$ and $h := \max(t_i)$. The runtime of our algorithm will depend on h. Since the problem is NP-hard (see Section [§ 3.2]), it is unlikely that there is an algorithm that is only polynomial in terms of n.

Theorem 3.1.1. There is a $n^{O(h^2)}$ algorithm for the distant representatives problem for vertical segments with vertical integer coordinates in $[0, h]$.

This form of runtime (i.e., $n^{O(\text{poly}(h))}$) is known as belonging to the complexity class XP [15]. The upshot is that for every fixed parameter h, there is a polynomial time algorithm. It is open if there is a Fixed Parameter Tractable (FPT) [15] or a pseudo-polynomial time algorithm.

Preliminaries.

First, we take note of an observation made by Biedl et al. [4]. We will restate it as a lemma.

Lemma 3.1.1. There is an $O(n^2 \log n)$ algorithm for the distant representatives problem when the segments are all on a single line.

They observe that the decision version of the problem can be solved using an algorithm from the scheduling literature [43]—each representative point is regarded as the centre-point of a unit length job.

Input assumptions.

Observation 3.1.1. There are at least two distinct x-coordinates, otherwise the problem can be solved in $O(n^2 \log n)$ by Lemma 3.1.1.

Lemma 3.1.2. There are at most h vertical line segments at each x-coordinate, otherwise the problem can be solved in polynomial time.

Proof of Lemma 3.1.2. Consider the case when there are more than h vertical line segments at a given x-coordinate. If there were at least $k \geq h+1$ segments at some coordinate, then looking strictly at those segments, an upper bound on δ^* is the minimum pairwise distance for those segments. Evenly dispersing the points would be the best we could hope for, which would give $\delta^* \leq \frac{h}{k-1} \leq 1$. We note that points with different x-coordinates have distance at least $1 \geq \delta^*$ apart, so the problem can be split up into independent subproblems by looking at all of the segments at a given x-coordinate. Each subproblem can be solved with the optimization algorithm for segments on a line. The optimal δ^* value is the minimum of the δ^* over all subproblems. This approach runs in $O(n^2 \log n)$ time by Lemma 3.1.1. □

We use this lemma to prove the following lemma needed for the algorithm:

Lemma 3.1.3. Let $f = h^2 + h$. Then every point on interval ℓ_i is distance at least δ^* away from intervals $\ell_1, \ldots, \ell_{i-f}$.

Proof. Let p denote an arbitrary set of representatives points. For any $j \leq i - f$, we will show that $d(p_i, p_j) \geq \delta^*$. This will work for both norms with one proof. First, we have $d(p_i, p_j) \geq |x_i - x_{i-f}|$. To lower bound $|x_i - x_{i-f}|$, we look at the set $\{x_{i-f}, x_{i-f+1}, \ldots, x_i\}$. There are $f + 1 = h^2 + h + 1$ indices listed here, but because elements may be repeated as many as h times, the size of this set is at least $\lceil \frac{h^2+h+1}{h} \rceil = h + 2$. Then clearly $|x_i - x_{i-f}| \geq ((h+2) - 1)X = (h+1)X$, where X is defined to be the minimum x distance between any two distinct x coordinates in the input. From this, we clearly see that $|x_i - x_{i-f}| \geq X(h+1) \geq Xh + X \geq h + X$.

Finally, we note that $h + X \geq \delta^*$, because $\delta^* \leq d(p_k, p_{k-1}) \leq |y_k - y_{k-1}| + |x_k - x_{k-1}| \leq h + X$, where k is defined to be the index where $|x_k - x_{k-1}| = X$. Putting the inequalities together, this proves that $d(p_i, p_j) \geq \delta^*$. Since p was arbitrary, then the lemma clearly holds. □

This lemma is very useful for designing a dynamic programming algorithm, as the point on ℓ_i only needs to be concerned with the points on the segments that are close. However, the challenge that remains is that infinitely many y-values from the interval I_i can be used. We can discretize the problem using results inspired by a technique used by Biedl et al. [4].

Output assumptions.

Claim 3.1.1. There exists a natural number $d \leq n$ and an optimal solution y^* for input $\mathcal{L}$ such that $y_i^* \in \mathbb{Z}/d$ for all $i \in [n]$.

The consequence of this claim is that we can simply add the constraint that $y_i \in \ell_i \cap \mathbb{Z}/d$ to our problem, without risk of missing the optimal δ^*. This discretizes the problem. This is assuming we know what d is, but trying all values for d and returning the best solution will suffice. The following lemmas help prove the claim:

Lemma 3.1.4. There is an optimal solution y so that for every $i \in [n]$, either $y_i = b_i$ or there exists $d_i \in [n-1], z_i \in \mathbb{Z}$ such that $y_i = d_i\delta^* + z_i$.

Lemma 3.1.5. δ^* is a rational number of the form k/d, where d is some natural number less than or equal to n.

Lemma 3.1.5 is similar to Lemma 11 from [4]. We will borrow some of the ideas from this proof directly, though it ends up being slightly easier to prove the lemma in the special case of vertical segments. Their lemma deals with the distant representatives problem in L_∞ for *rectangles* in the plane with rational coordinates, which is a more general case than what we are dealing with. Restricting ourselves to just vertical segments with integer coordinates lets us prove a result for the L_1 norm as well. This lemma will not hold in the L_2 norm, which prevents us from giving an algorithm that works for the L_2 norm.

Proof of Lemma 3.1.4. Consider any optimal solution y to the problem, and over all possible optimal solutions, take the one with minimum $\sum y_i$ value. Let $p_i = (x_i, y_i)$ for all $i = 1, \ldots, n$. Let i_j denote the index of the point with the j'th smallest y-value, for all $j = 1, \ldots, n$.

We know that y_{i_1} must be on the bottom endpoint b_{i_1}, otherwise we could move it down without decreasing the minimum pairwise distance, but this would result in a solution with a better $\sum y_i$ value. For the remaining points, we also should not be able to move them downwards, which means some point below it is exactly distance δ^* away from it, or the point is at the bottom endpoint. We break into cases by distance metric.

Case: L_∞. In L_∞, we have that for all $j = 1, \ldots, n$, either $y_{i_j} = b_{i_j}$ or $d_\infty(p_{i_j}, p_{i_k}) = \delta^*_\infty$ for some $k < j$. In fact, $d_\infty(p_{i_j}, p_{i_k}) = \max(|x_{i_j} - x_{i_k}|, |y_{i_j} - y_{i_k}|) = \delta^*_\infty$ can be simplified to $y_{i_j} - y_{i_k} = \delta^*_\infty$ (meaning the distance is tight in terms of the difference in y). This is because decreasing y_{i_j} causes the distance between p_{i_j} and p_{i_k} to become less than δ^*_∞, otherwise, $\sum_i y_i$ has not been minimized.

We can further generalize this second statement to the following: there are integers $0 < d_{i_j} < n$ and $k < j$ so that $y_{i_j} = b_{i_k} + d_{i_j}\delta^*_\infty$. This can be proven by induction on j. This proves Lemma 3.1.4 in the L_∞ case.

Case: L_1. In L_1, we have that for all $j = 1, \ldots, n$, $y_{i_j} = b_{i_j}$ or $|x_{i_j} - x_{i_k}| + y_{i_j} - y_{i_k} = \delta^*_1$ for some $k < j$. It is harder to generalize the second expression as we did in the L_∞ case, because there is a term involving the x_i values. However, we know that all of the x-coordinates are constant integers. So what we can say is: there are integers $0 < d_{i_j} < n, k < j$ so that $y_{i_j} = b_{i_k} + d_{i_j}\delta^*_1 + X_{i_j}$, where X_{i_j} is some integer value involving the sum and/or difference of x values. Setting $z_{i_j} = b_{i_k} + X_{i_j}$, this proves Lemma 3.1.4 in the L_1 case. □

We can now prove Lemma 3.1.5.

Proof of Lemma 3.1.5. First, we need to find some point p_j that is on its top endpoint and is distance δ^* above some other point p_i. From this, we apply Lemma 3.1.4 to p_i to prove Lemma 3.1.5. Assume for contradiction that no such j exists. Let $\varepsilon > 0$ be a sufficiently small number that is less or equal to $\min\{t_k - y_k : k \in [n], y_k \neq t_k\}$.

If all points are on their top endpoints, then ε is not well defined. But in this case, we get a contradiction. At least one point is not at a bottom endpoint, otherwise, all vertical segments are length 0, which is a completely trivial problem. So this bottom point is δ^* units above another point, otherwise it could be pushed down, contradicting that y minimizes $\sum_i y_i$.

Given ε, we will define a new solution as follows: increase every value $y_{i_j} \neq t_{i_j}$ up to $y_{i_j} + \frac{j}{n}\varepsilon$. Meanwhile, the y-values that cannot be increased will remain at their top endpoint. This is clearly a feasible solution. Note that the distance between the points that move *will increase*, so the distance is strictly greater than δ^*. Comparing a point that moves with a point that did not move (i.e., a point at its segment's top endpoint), we note that their distance will remain $> \delta^*$ if ε is sufficiently small. This is assuming the distance between the top endpoint and the moving point was not exactly δ^* before the move, but by assumption, this was the case. So we have just constructed a new solution where the minimum pairwise distance is greater than δ^*. This is a contradiction.

Therefore, there must be two points p_i, p_j that are distance δ^* apart with $y_j = t_j$. Applying Lemma 3.1.4 to y_i, we see $y_i = b_i$ or $y_i = d_i\delta^* + z_i$. In the L_∞ case, either $t_j = y_j = y_i + \delta^*_\infty = b_i + \delta^*_\infty$ or $t_j = y_j = y_i + \delta^*_\infty = (d_i + 1)\delta^*_\infty + z_i$, and rearranging for δ^*_∞ proves the claim. In the L_1 case, either $t_j = y_j = y_i + \delta^*_1 - |x_i - x_j| = b_i + \delta^*_1 - |x_i - x_j|$, or $t_j = y_j = y_i + \delta^*_1 - |x_i - x_j| = b_i + (d_i + 1)\delta^*_1 + z_i - |x_i - x_j|$, and rearranging for δ^*_1 proves the claim. □

Finally, we prove Claim 3.1.1.

Proof of Claim 3.1.1. Let $\delta^* = k/d$ where d is an integer less than or equal to n by Lemma 3.1.5. Take the optimal solution y provided by Lemma 3.1.4. For every $i \in [n]$, either $y_i = b_i \in \mathbb{Z}/d$, or $y_i = d_i\delta^* + z_i$. Since $d_i\delta^* \in \mathbb{Z}/d$ and $z_i \in \mathbb{Z}/d$, then so is y_i. □

The Dynamic Programming Algorithm

We will now prove Theorem 3.1.1. We give a dynamic programming algorithm for fixed h, and our current guess for the value of the denominator of δ^*, denoted d, following Lemma 3.1.5.

Let v be a vector of f elevation values, where each value is restricted to grid points $\mathbb{Z}/d$ between 0 and h (i.e. $v \in (\mathbb{Z}/d \cap [0, h])^f$). The idea for this dynamic programming algorithm is to consider all possible combinations of elevations v for the last $f := h^2 + h$ segments of the subproblem $\mathcal{L}_i = \{\ell_1, \ldots, \ell_i\}$. That is, for the current combination v, we set y_{i-f+1} to be the first element of v, and we set y_{i-f+2} to be the second element of v, and so on. For

every i and for every v, we store the optimum value for $\mathcal{L}_i$, assuming we are forced to use the elevations from v for the last f imprecise points. The notation we will use for this solution is $\delta^*(i, v)$.

Only f segments are needed because by Lemma 3.1.3, the only points that could be within distance δ^* of a point on ℓ_i are the points on $\ell_{i-f+1}, \ldots, \ell_{i-1}$. Also, by Claim 3.1.1, the output points can be restricted to the grid points $\mathbb{Z}/d$, so the number of possible combinations is finite. The table we store all of these values in is size $(n-f)\times(dh+1)$, since $|(\mathbb{Z}/d \cap [0,h])^f| = |\{0, \frac{1}{d}, \frac{2}{d}, \ldots, h\}| = dh+1$. Naturally, because some of the y-intervals may be a strict subset of $[0, h]$, then not all table entries can or should be filled.

In order to compute $\delta^*(i, v)$, we first compute the minimum pairwise distance for the last f points. For the remaining pairwise distances, we will look at the optimal values for the subproblems in the row $\delta^*(i-1, \cdot)$ of the table. To access an entry of this row, we need the f elevation values for $y_{i-f}, y_{i-f+1}, \ldots, y_{i-1}$. The vector v determines the elevation values for $y_{i-f+1}, \ldots, y_{i-1}$. As for y_{i-f}, we try all possible elevation values, and for each choice, we use it to index into row $\delta^*(i-1, \cdot)$. We return the best solution computed over all choices for y_{i-f}. With some additional notation, we can turn the description of the dynamic programming algorithm into an explicit formula for $\delta^*(i, v)$.

Let $\lambda(i, v)$ denote the distance between the closest pair of points amongst the last f points of $\mathcal{L}_i$, assuming the elevation values are determined by v.

The base case is:

$$\delta^*(f, v) = \lambda(f, v)$$

Let $\gamma(i, v)$ denote the distance between the point $p_i \in \ell_i$ and the closest point amongst $p_{i-f+1}, \ldots, p_{i-1}$, assuming the elevation values $y_{i-f+1}, \ldots, y_i$ are determined by v. For $a \in \mathbb{Z}/d \cap [0, h]$, let $\text{Shift}(v, a)$ denote the vector v but with the last element removed and the element a added to the front.

The recursive step is:

$$\delta^*(i, v) = \min\{\gamma(i, v), \max\{\delta^*(i-1, \text{Shift}(v, a)) : a \in I_{i-f} \cap \mathbb{Z}/d\}\}$$

After the table has been fully computed, we return the best solution is $\delta^* = \max\{\delta^*(n, v) : v \in (\mathbb{Z}/d \cap [0, h])^f\}$. In order to return an actual set of representative points, each entry $\delta^*(i, v)$ of the table should also store the best value a for the elevation of y_{i-f}.

This approach only makes sense when $n > f$. If $n \leq f$, dynamic programming is not used, instead we will brute force check for the optimal solution by applying Claim 3.1.1.

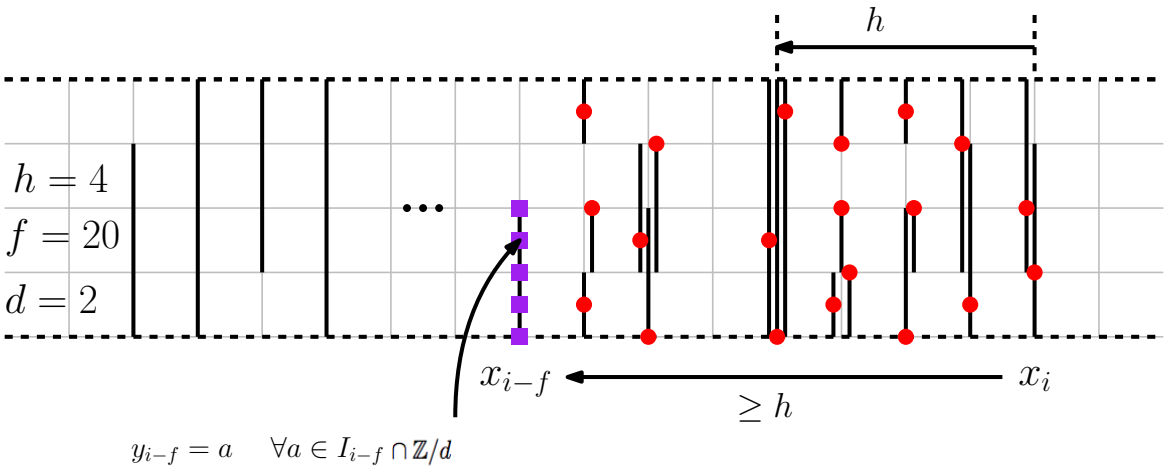


Figure 3.1: How to compute $\delta^*(i, v)$. The dots (red) are the points we place on segments $\ell_{i-f+1}, \ldots, \ell_i$ with elevations determined by v. We can look at $\delta^*(i-1, \cdot)$, where $y_{i-f+1}, \ldots, y_{i-1}$ are determined by v. Since we need to know the placement of points on the last f segments of $\mathcal{L}_{i-1}$ in order to access entries of $\delta^*(i-1, \cdot)$, we simply try all possible values $a \in I_{i-f} \cap \mathbb{Z}/d$ for y_{i-f}. The purple boxes represent all possible points we can choose $p_{i-f} = (x_{i-f}, y_{i-f})$ to be.

Correctness: The base case is clearly correct. We will show $\delta^*(i, v)$ has been computed correctly in the general case. By Lemma 3.1.3, the placement of $p_i = (x_i, y_i)$ is not affected by $p_1, \ldots, p_{i-f}$. Therefore, $y_1, \ldots, y_{i-f}$ should be chosen optimally only concerned about the choices y_{i-f} to y_{i-1} as determined by v. By induction, we should look at entries in $\delta^*(i-1, \cdot)$ where the last $f-1$ values are determined by v. Since we need to know the last f points of $\mathcal{L}_{i-1}$ to access $\delta^*(i-1, \cdot)$, we simply try all possible values for y_{i-f}, and take the best solution from $\delta^*(i-1, \cdot)$ that we find. In order to correctly compute $\delta^*(i, v)$, we also have to account for the closest pair involving the last point p_i, which is why we also compute $\gamma(i, v)$.

Runtime. We first consider when $n \geq f$. For the base case, it takes $O(f^2)$ time to compute the closest pair of points for a given v, so computing for all v takes at most $O(f^2(dh+1)^f)$ time. Since $d \leq n$ and $h < f \leq n$, then $dh < n^2$, so $dh + 1 \leq n^2$, therefore we can simplify to $O(f^2(dh+1)^f) = O(n^2(n^2)^f) = O(n^{2(h^2+h)+2}) = n^{O(h^2)}$.

What is the cost of computing $\delta^*(i, v)$ in the general case? Computing $\gamma(i, v)$ takes $O(f)$ time. Trying all values for y_{i-f} takes $O(dh+1)$ time, so in total, computing $\delta^*(i, z)$ from $\delta^*(i-1, \cdot)$ takes $O(f(dh+1))$ time. Computing for all v takes $O(f(dh+1)^{f+1})$ time, which is $n^{O(h^2)}$ time by a similar argument as above.

The time to fill in the table is $(n-f)n^{O(h^2)} = n^{O(h^2)}$. Computing the final answer from $\delta^*(n, \cdot)$ is simply $O(n(dh+1)^f) = n^{O(h^2)}$ time. Accounting for all possible values of d, we get at most $n * n^{O(h^2)} = n^{O(h^2)}$ time.

Finally, we address the case $n \leq f$. This was a corner case where dynamic programming

was not used, but we still need to show that the runtime is $n^{O(h^2)}$. For each d, the time complexity to brute force all solutions is $c * (dh+1)^n$ for some constant c. Since $d \leq n$, the time is at most $c * (nh+1)^n$. If $h \leq n \leq f$ then $c(nh+1)^n \leq c(n^2+1)^f = n^{O(h^2)}$. The case when $n < h$ is trickier. We will argue that if we swap the base and the exponent for the expression $(nh+1)^n$, we get an upper bound for the expression. Note that the function $x/\log(x)$ is an increasing function when $x \geq e = 2.7182\ldots$. Asymptotically, we can assume $n \geq e$. Since $n < nh+1$ then $n/\log n < (nh+1)/\log(nh+1) \implies (nh+1)^n < n^{nh+1}$. Therefore, $c(nh+1)^n < cn^{nh+1} < cn^{h^2+1} = n^{O(h^2)}$. Accounting for at most n values for d, the runtime is $n * n^{O(h^2)} = n^{O(h^2)}$.

3.2 NP-hardness for parallel segments

In this section we show that, even for the special case of unit parallel segments, the decision version of the problem is NP-complete for L_1 and L_∞ and NP-hard for L_2 (where bit complexity issues prevent us from placing the problem in NP). We assume that the segments are horizontal—this is at odds with the assumption of vertical segments for terrains in Chapter 2 and with Section 3.1, but it matches the earlier work [4]. This L_2 result was proved previously by Roeloffzen in his Master's thesis [41, Section 2.3] but we add details regarding bit complexity that were missing from his proof.

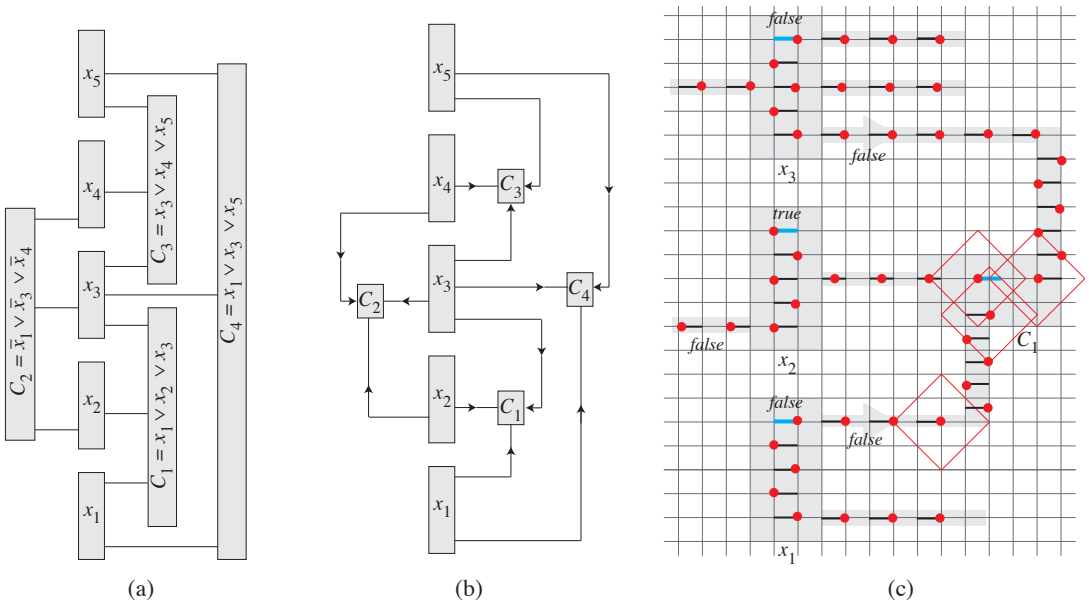
Our reductions are from the NP-complete problem Monotone Rectilinear Planar 3-SAT [11] in which each clause has either three positive literals or three negative literals, each variable is represented by a thin vertical rectangle at x-coordinate 0, each positive [negative] clause is represented by a thin vertical rectangle at a positive [negative, resp.] x-coordinate, and there is a horizontal line segment joining any variable to any clause that contains it. See Figure 3.2(a) for an example instance of the problem. For n variables and m clauses, the representation can be on an $O(m) \times O(n+m)$ grid.

Theorem 3.2.1. The decision version of the distant representatives problem for unit horizontal segments in the L_1, L_2 or L_∞ norm is NP-hard.

The decision problem lies in NP for the L_1, L_∞ norm [4]. Bit complexity issues prevent us from placing the decision problem in NP for the L_2 norm.

For our reduction from Monotone Rectilinear Planar 3-SAT we first modify the representation so that each clause rectangle has fixed height and is connected to its three literals via three "wires"—the middle one remains horizontal, the bottom one bends to enter the clause rectangle from the bottom, and the top one bends twice to enter the clause rectangle from the far side. See Figure 3.2(b). Each wire is directed from the variable to the clause, and represents a literal. The representation is still on an $O(m) \times O(n+m)$ grid.

To complete the reduction to the distant representatives problem we replace the rectangles with variable and clause gadgets constructed from unit horizontal intervals, and also implement the wires using such intervals.



(a) (b) (c)

Figure 3.2: (a) An instance of Monotone Rectilinear Planar 3-SAT. (b) The modified representation used for our NP-hardness proofs, with wires from variable to clause gadgets. (c) A detail of our NP-hardness construction for clause $C_1 = x_1 \vee x_2 \vee x_3$ in the L_1 norm showing how the truth-value setting $x_1 =$ False, $x_2 =$ True, $x_3 =$ False, permits representative points (shown as red dots) at distance at least $\delta_1 = 2$.

Our constructions will ensure the following properties.

P1 Each variable gadget has a *variable interval* whose representative point can only be at its left endpoint (representing the True value of the variable) or at its right endpoint (representing the False value of the variable). With either choice, there is a valid assignment of representative points to all intervals in the variable gadget.

P2 A wire is constructed as a sequence of intervals. There are two special valid assignments of representative points to the intervals of a wire which we call the "false setting" and the "true setting". Details will be given later on, but for now, we just note that along the horizontal portion of a wire, the false setting places the representative point of each interval at the interval's forward end (relative to the direction of the wire). See Figure 3.2(c).

The true/false settings for wires behave as follows. If a wire corresponds to a literal that is false (based on the left/right position of the representative point of the corresponding variable interval), then the false setting is the *only* valid assignment of representative points for the intervals in the wire. If a wire corresponds to a literal that is true, then the true setting of the wire is *a* valid assignment of representative points. Note the asymmetry here—the false setting is forced but the true setting is not.

P3 Each clause gadget consists of one *clause interval*. If all three wires coming in to a clause gadget have the false setting then there is no valid assignment of a representative point to the clause interval. If at least one of the wires has a true setting then there is a valid assignment of a representative point to the clause interval.

Lemma 3.2.1. Any construction with the above properties gives a correct reduction from Monotone Rectilinear Planar 3-SAT to the decision version of the distant representatives problem.

Proof. We must prove that the original instance, Φ, of Monotone Rectilinear 3-SAT is satisfiable if and only if the constructed instance, $\mathcal{I}$, of the distant representatives problem has a valid assignment of representative points, i.e., an assignment of representative points such that any two points are at least distance δ_ℓ apart.

First suppose that Φ is satisfiable. By Property P1 we can choose a a valid assignment of representative points to the intervals of the variable gadgets such that a variable being True/False corresponds to using the left/right endpoint (respectively) of the variable interval. We then choose the true/false settings of the wires according to Property P2—the false setting for wires of false literals and the true setting for wires of true literals. Since Φ is satisfiable, every clause contains a True literal. The corresponding incoming wire has been given a true setting. Then, by Property P3, the clause interval has a valid assignment of representative points. Thus $\mathcal{I}$ has a valid assignment of representative points.

For the other direction, suppose $\mathcal{I}$ has a valid assignment of representative points. By Property P1 this corresponds to a truth value assignment to the variables. By Property P2 the wires corresponding to false literals can only have the false setting (though we don't know about the wires corresponding to true literals). By Property P3 every clause has at least one incoming wire that does not have the false setting, and this wire must then correspond to a True literal. Thus, every clause is satisfied and Φ is satisfiable. □

In the following subsections we describe the variable gadgets, wires, and clause gadgets for each of the norms L_1, L_2, L_∞. In each case we prove that the above properties hold.

L_1 norm, $\delta_1 = 2$.

Variable gadget. We use a *ladder* consisting of unit intervals, called *rungs*, in a vertical pile, unit distance apart. See Figure 3.3(a). Number the rungs starting with rung 1 at the top. Observe that the L_1 distance between opposite endpoints of two consecutive rungs is $\delta_1 = 2$. Thus there are precisely two ways to assign representative points to a ladder of at least two rungs. For Property P1, let the variable interval be rung 1, and associate the value True [False] if rung 1 has its representative point on the left [right, resp.].

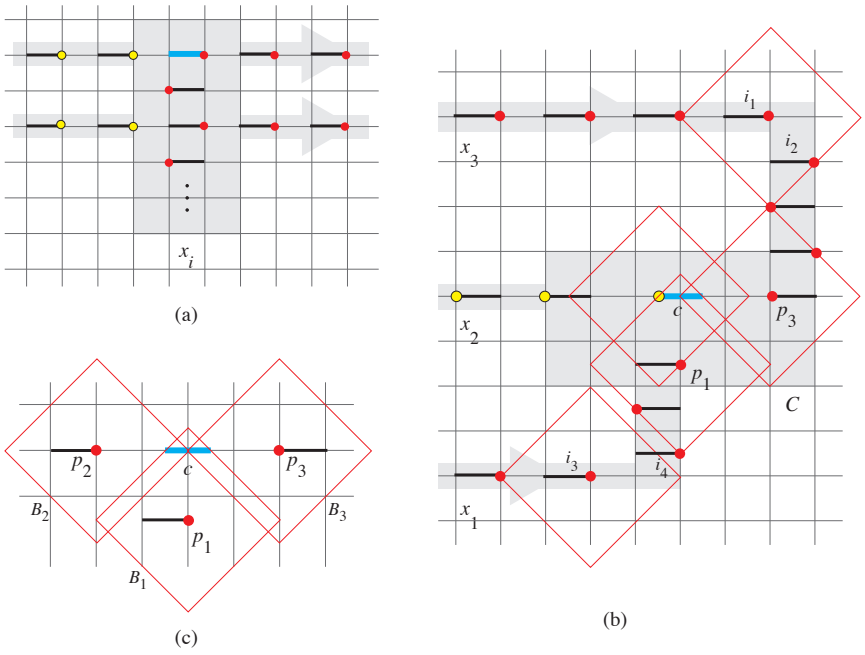


Figure 3.3: Construction for the L_1 norm. (a) Variable gadget. Intervals have length 1 and $\delta_1 = 2$. Variable x_i is shown with the False setting where the representative point (the red dot) on the variable interval (shown in cyan) is on the right end. The two wires heading right are forced to have the false setting (shown with red dots). The two wires heading left have the true setting (shown with yellow dots). (b) Wires entering the clause gadget for clause $C = x_1 \vee x_2 \vee x_3$. The clause interval c (shown in cyan) is displaced horizontally by 1/2. Valid representative points are shown for the truth-value setting $x_1 =$ False, $x_2 =$ True, $x_3 =$ False. (c) A close-up of the clause interval c showing the L_1 balls B_i of radius $\delta_1 = 2$ centred at p_i, $i = 1, 2, 3$.

Horizontal wire. For each horizontal portion of a wire, use a sequence of unit intervals separated by gaps of length 1. Attach the wires to the odd numbered rungs of a ladder in a variable gadget, with the rung acting as the first interval of the wire. The false setting has representative points at the forward end of each interval (relative to the direction of the wire). The true setting has representative points at the other end of each interval. For Property P2 (that the false setting is forced) observe that if a variable is False then its odd-numbered rungs have their representative points on the right, so any horizontal wire extending to the right is forced to use representative points on the right (the forward end) of every interval of the wire. On the other hand, if a variable is True then horizontal wires extending to the right *may* use the true setting. Analogous properties hold for the horizontal wires extending to the left.

Turning wires. We focus on the situation for a positive clause—the situation for a negative clause is symmetric. The top wire coming in to a clause gadget turns downward via a wire

ladder as shown in Figure 3.3(b). Note that the false setting of interval i_1 in the figure forces the false setting of interval i_2, which then forces the settings down the wire ladder. Note that the wire ladder can be as long as needed. Since wires emanate from odd-numbered rungs of variable ladders, the wire ladder has an even number of rungs and the bottom interval of the wire ladder, at the horizontal line of the middle wire coming in to the clause, has its false setting on the left (see point p_3 in the figure). One can verify that the true setting (with representative points at the opposite end of each interval) is valid.

The bottom wire coming in to a clause gadget turns upward as shown in Figure 3.4(b) via a wire ladder of intervals that are on the half-grid. This wire ladder has an odd number of rungs, and we can ensure at least 3 rungs. The false setting of interval i_4 is forced because of the false setting of interval i_3 together with the ladder above i_4. The topmost interval of the wire ladder has its false setting on the right. One can verify that the true setting (with representative points at the opposite end of each interval) is valid.

We have now established Property P2 for wires that turn.

Clause gadget. See Figure 3.3(c). The figure shows the clause interval c together with the last interval in each of the three wires that come in to the clause gadget, and the false settings of their representative points at p_1, p_2, p_3. The L_2 distance between p_1 and either endpoint of c is $\delta_1 = 2$. Let B_i be the L_1 ball of radius δ_1 centred at p_i, $i = 1, 2, 3$. We now verify Property P3. Observe that no point of the interval c is outside all three balls. Thus, if all three incoming wires have the false setting, there is no valid representative point for interval c. However, if at least one of the incoming wires has the true setting, we claim that there is a valid representative point on interval c: If p_1 is at the left of its interval, use the midpoint of c, and if either of p_2, p_3 is at the other endpoint of its interval, use the endpoint of c on that side. Thus Property P3 holds.

L_2 norm, $\delta_2 = \frac{13}{5}$

Consider two unit intervals one above the other, separated by vertical distance d_y. The L_2 distance between opposite endpoints of the intervals is $d = \sqrt{1 + d_y^2}$. In order to have rational values for d_y and d, we need scaled Pythagorean triples, natural numbers a, b, c with $a^2 + b^2 = c^2$. We base our construction on the Pythagorean triple $5, 12, 13$. (The triple $3, 4, 5$ causes some interference.) To avoid writing fractions everywhere, we describe the construction for intervals of length 5, with $\delta_2 = 13$. Scaling everything by $\frac{1}{5}$ gives us back unit intervals.

Our construction of variable gadgets and horizontal wires is like the L_1 case, just with different spacing.

Variable gadget. Use a ladder with rungs of length 5 spaced 12 vertical units apart. See Figure 3.4(a). The L_2 distance between opposite endpoints of two consecutive rungs is

$\delta_2 = 13$. Associate the value True [False] if rung 1 has its representative point on the left [right, resp.]. Property P1 holds.

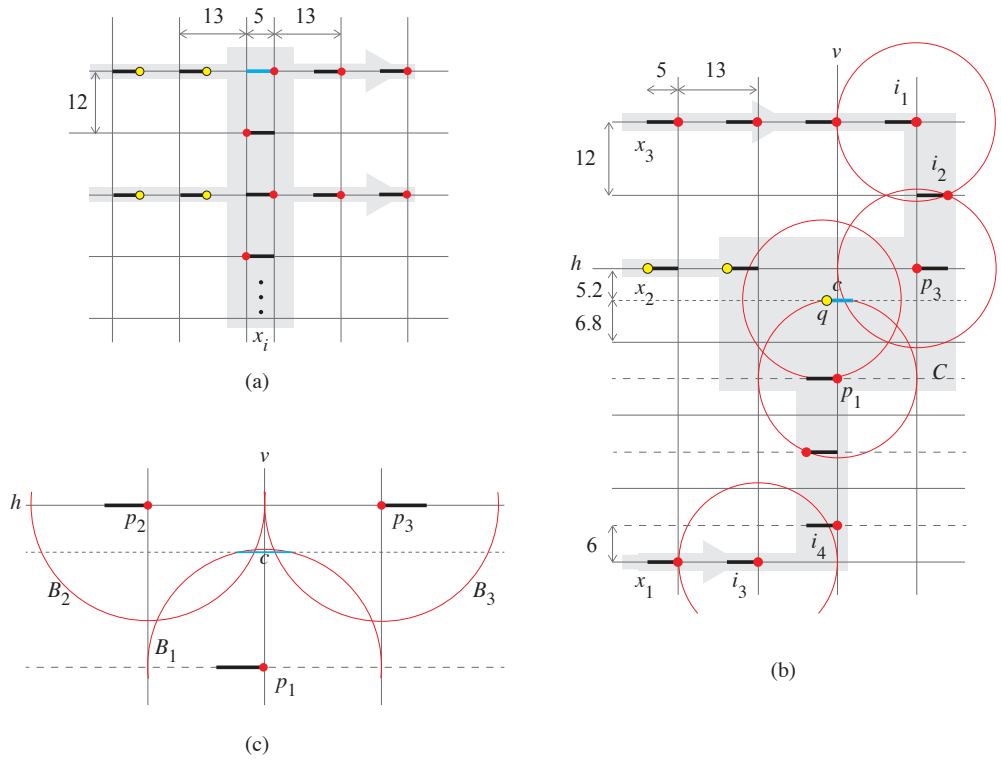


Figure 3.4: Construction for the L_2 norm. (a) Variable gadget. Intervals have length 5 and $\delta_2 = 13$. Variable x_i is shown with the False setting where the representative point (the red dot) is on the right end of the variable interval (shown in cyan). The two wires heading right are forced to have the false setting (shown with red dots). The two wires heading left have the true setting (yellow dots). (b) Wires entering the clause gadget for clause $C = x_1 \vee x_2 \vee x_3$. The clause interval c (shown in cyan) extends from $v - 2.5$ to $v + 2.5$ at y-coordinate $h - 5.2$, where v and h are the grid coordinates as shown. Valid representative points are shown for the truth-value setting $x_1 =$ False, $x_2 =$ True, $x_3 =$ False. (c) A close-up of the clause interval c showing the balls B_i of radius $\delta_2 = 13$ centred at p_i, $i = 1, 2, 3$.

Horizontal wire. For each horizontal portion of a wire, use intervals of length 5 separated by gaps of length 8 (so the right endpoints of two consecutive intervals are distance 13 apart). Attach the wires to the odd-numbered rungs of the ladder of the variable gadget. The false setting has representative points at the forward end of each interval, and is forced if the corresponding literal is False. The true setting has representative points at the other end of each interval, and is valid if the corresponding literal is True. So Property P2 holds.

Turning wires. As for L_1, we focus on the situation for a positive clause—the situation for a negative clause is symmetric. The top wire coming in to a clause gadget turns downward as shown in Figure 3.4(b). As for L_1, the false setting of interval i_1 forces the false setting of interval i_2, which then forces the bottom interval of the wire ladder (coming in to the clause) to have its false setting on the left (see point p_3 in the figure). One can verify that the true setting (with representative points at the opposite end of each interval) is valid.

The bottom wire coming in to a clause gadget turns upward as shown in Figure 3.4(b) via a wire ladder of intervals that are on the half-grid, i.e., i_4 in the figure is 6 units above i_3. This wire ladder has an odd number of rungs, and we can ensure at least 3 rungs. The false setting of interval i_4 is forced because of the false setting of interval i_3 together with the ladder above i_4. The topmost interval of the wire ladder has its false setting on the right. One can verify that the true setting (with representative points at the opposite end of each interval) is valid.

We have now established Property P2 for wires that turn.

Clause gadget. See Figure 3.4(c). The figure shows the clause interval c together with the last interval in each of the three wires that come in to the clause gadget, and the false settings of their representative points at p_1, p_2, p_3. The clause interval c extends from $v-2.5$ to $v+2.5$ at y-coordinate $h-5.2$, where v and h are the grid coordinates as shown. Then the L_2 distance between p_1 and either endpoint of c is $\sqrt{2.5^2+12.8^2} \approx 13.04$ which is greater than $\delta_2 = 13$. Let B_i be the ball of radius δ_2 centred at p_i, $i = 1, 2, 3$. The endpoints of c lie just outside B_1. We now verify Property P3. Observe that no point of the interval c is outside all three balls. Thus, if all three incoming wires have the false setting, there is no valid representative point for interval c. We now consider what happens if at least one incoming wire has the true setting, i.e., if p_1, p_2, or p_3 were at the other end of its interval. If p_2 were at the other endpoint of its interval, then the left endpoint of c would be a valid representative point. Similarly, if p_3 were at the other endpoint of its interval, then the right endpoint of c would be a valid representative point. Finally, if p_1 were at the other endpoint of its interval, then the midpoint of c would be a valid representative point. Thus Property P3 holds.

L_∞ norm, $\delta_\infty = \frac{1}{2}$

In this case ladders still work, but it is difficult to attach wires to ladders, so we use a more complicated variable construction. A further difficulty for the L_∞ case is that we were unable to construct a clause gadget of unit intervals based on choosing representative points only on the left/right endpoints of intervals. (Although this is easy if the clause interval can have length 2.) Instead, our construction will place representative points at the endpoints or at the middle of each interval, which is why we set $\delta_\infty = \frac{1}{2}$. For this norm, we describe horizontal wires first.

Horizontal wire. We use a double row of unit intervals spaced 1/6 apart vertically. Specifically, along one horizontal line, we place a sequence of unit intervals with endpoints at each integer coordinate, and along the horizontal line 1/6 below, we place a sequence of unit intervals with endpoints at each half integer coordinate. See Figures 3.5 and 3.6. For Property P2, note that if two consecutive intervals have their representative points at their right endpoints, then all intervals further to the right on the wire must also have their representative points at their right endpoints. Along a horizontal wire, the true setting places representative points at the midpoints of the intervals.

Variable gadget. This gadget has eight intervals as shown in Figure 3.5. The three intervals i_1 are coincident (or almost so), as are the three intervals i_2. These force the representative point on interval i_3 to the middle of the interval. Then the representative point on the variable interval (coloured cyan in the figure) must be either the left endpoint (representing a True value) or the right endpoint (representing a False value).

This eight-interval configuration is expanded to a "double ladder" with intervals spaced 1/6 apart vertically as shown in Figure 3.5. Down the ladder on the false side (which is the right side in the figure), the representative point for each interval is forced by those on the two intervals above it. Down the ladder on the true side (the left side in the figure) we may use the assignment of representative points as shown in the figure. Wires extend to the right and left of the double ladder as shown in the figure. Any wire corresponding to a false literal is forced to the false setting. Any wire corresponding to a true literal may have the true setting. Properties P1 and P2 hold.

Turning wires. The top wire coming in to a positive clause gadget turns downward as shown in Figure 3.6. The false setting along the wire (see (a) in the figure) forces the bottom interval of the ladder to have its false setting on the left (see point p_3 in the figure). The ladder can be extended to the appropriate length by adding multiples of three intervals. The true setting is shown in Figure 3.6(b), and allows the right endpoint of the clause interval to be used as a representative point.

The bottom wire coming in to a clause gadget turns upward as shown in Figure 3.6. The false setting along the wire (see (a) in the figure) forces the top interval of the ladder to have its false setting on the right (see point p_1 in the figure). The ladder can be extended to the appropriate length by adding multiples of three intervals. The true setting is shown in Figure 3.6(b), and allows the midpoint of the clause interval to be used as a representative point. We have now established Property P2 for wires that turn.

Clause gadget. See Figure 3.6. The figure shows the clause interval c together with the three wires coming in to the clause gadget. Let p_1, p_2, p_3 be the representative points of the last intervals in the wires entering the clause gadget. Consider the false positions of p_1, p_2, p_3. (Figure 3.6(a) shows the false positions of p_1 and p_3.) The L_∞ distance between p_1 and either

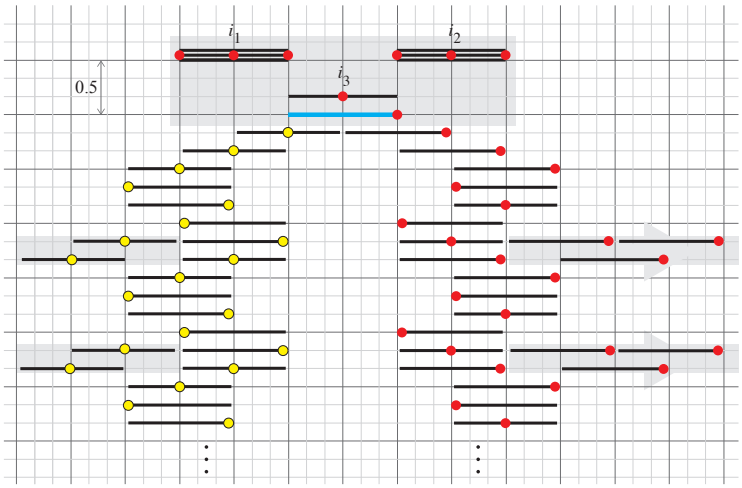


Figure 3.5: The variable gadget and emanating wires for L_∞. Grid spacing is 1/6. The shaded rectangle at the top shows the variable gadget. The variable interval (shown in cyan) has a choice of representative point at the left endpoint (True) or the right endpoint (False). The False choice is shown here. The placement of representative points down the right hand ladder (shown as red dots) is then forced, and then the wires emanating to the right are forced to the false setting. The placement of points down the left hand ladder (shown as yellow dots) is allowed, and the wires emanating to the left are allowed to be in the true setting.

endpoint of c is $\delta_\infty = \frac{1}{2}$. Let B_i be the L_∞ ball of radius δ_∞ centred at p_i, $i = 1, 2, 3$. We now verify Property P3. Observe that no point of the interval c is outside all three balls. Thus, if all three incoming wires have the false setting, there is no valid representative point for interval c. However, if at least one of the incoming wires has the true setting, we claim that there is a valid representative point on interval c. Refer to Figure 3.6(b). If p_1 is at the middle of its interval, use the midpoint of c, and if either of p_2, p_3 is at the middle of its interval, use the endpoint of c on that side. Thus Property P3 holds.

3.3 APX-hardness for orthogonal segments

In this section, we prove hardness-of-approximation results for the distant representatives problem on horizontal and vertical segments in the plane. Specifically, we prove lower bounds on the approximation factors that can be achieved in polynomial time, assuming P $\neq$ NP.

Theorem 3.3.1. For $\ell = 1, 2, \infty$, let g_ℓ be the constant shown in Table 3.2. Suppose P $\neq$ NP. Then, for the L_ℓ norm, there is no polynomial time algorithm with approximation factor less than g_ℓ for the distant representatives problem for horizontal and vertical segments.

We prove Theorem 3.3.1 using a *gap reduction*. This standard approach is based on the fact that if there were polynomial time approximation algorithms with approximation factors

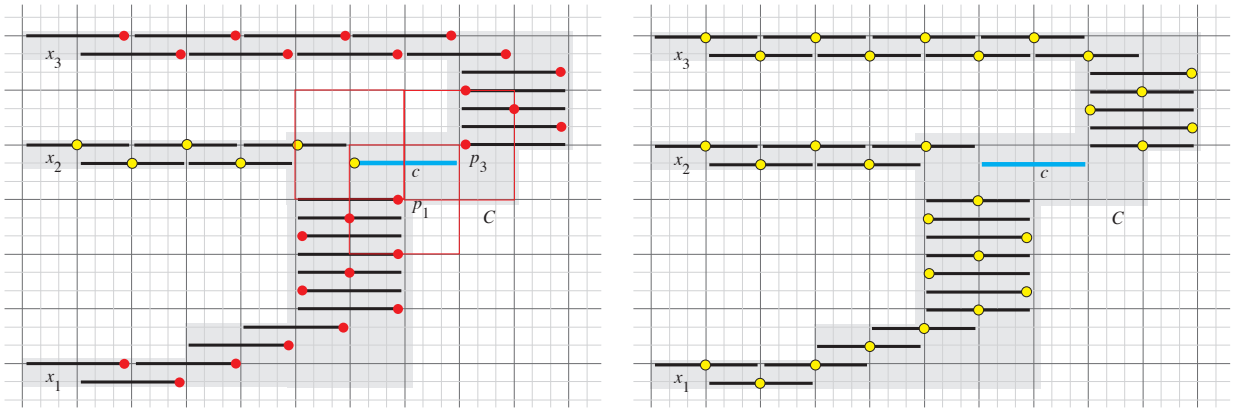


Figure 3.6: The clause gadget and its entering wires for L_∞ for clause $C = x_1 \vee x_2 \vee x_3$. The clause interval c is shown in cyan. (a) Valid representative points for the truth-value setting x_1 = False, x_2 = True, x_3 = False. (b) The true settings on the incoming wires. (These points are not forced.)

	L_1	L_2	L_∞
lower bound	$g_1 = 1.5$	$g_2 = 1.4425$	$g_\infty = 1.5$

Table 3.2: Best approximation ratios that can be achieved unless P=NP.

better than g_ℓ then the *gap versions* of the problem (as stated below) would be solvable in polynomial time. Thus, proving that the gap versions are NP-hard implies that there are no polynomial time g_ℓ-approximation algorithms unless P=NP.

Recall that δ^*_ℓ is the max over all assignments of representative points, of the min distance between two points.

Gap Distant Representatives Problem.
Input: A set I of horizontal and vertical segments in the plane.
Output:

- YES if $\delta^*_\ell(I) \geq 1$;
- NO if $\delta^*_\ell(I) \leq 1/g_\ell$;
- and it does not matter what the output is for other inputs.

To prove Theorem 3.3.1 it therefore suffices to prove:

Theorem 3.3.2. The Gap Distant Representatives problem is NP-hard.

This is proved via a reduction from Monotone Rectilinear Planar 3-SAT, much like in the previous section. The gadgets are simpler because we can use vertical segments, but we

must prove stronger properties. Given an instance Φ of Monotone Rectilinear Planar 3-SAT we construct in polynomial time a set of horizontal and vertical segments I such that:

Claim 3.3.1. If Φ is satisfiable then $\delta_\ell^*(I) = 1$.

Claim 3.3.2. If Φ is not satisfiable then $\delta_\ell^*(I) \leq 1/g_\ell$.

Thus a polynomial time algorithm for the Gap Distant Representatives problem yields a polynomial time algorithm for Monotone Rectilinear Planar 3-SAT. We give some of the reduction details, but defer the proofs of the claims to the full version.

Reduction details We reduce directly from Monotone Rectilinear Planar 3-SAT.

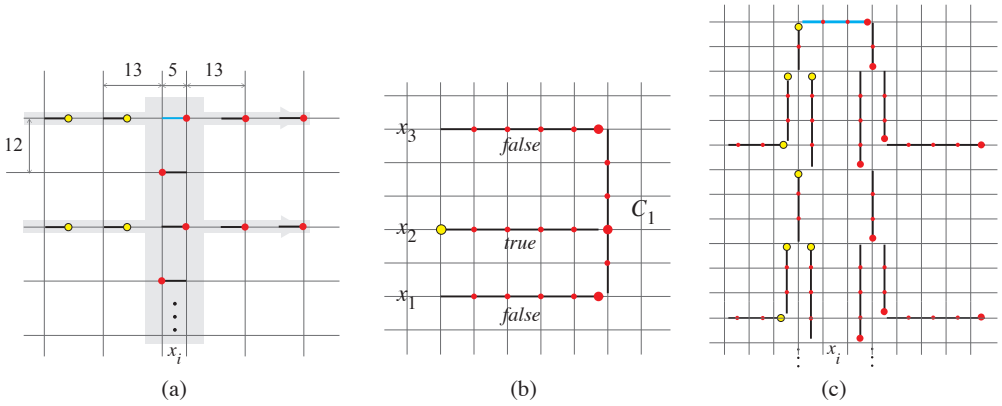


Figure 3.7: (a) A variable gadget for NP-hardness for L_2, based on Pythagorean triple $5, 12, 13$. To achieve $\delta = 13$ the representative point for the variable interval (in cyan) is forced to the left (true) or the right (false) in which case intervals on the right are also forced. (b) A clause gadget for the APX-hardness reduction, with three horizontal wires attached. The small dots represent the 0-length line segments. For clarity, the long segments are not drawn all the way to their endpoints. Wires x_1 and x_2 are in the false setting and wire x_2 is in the true setting, which allows the representative point for C_1 to be placed where the x_2 wire meets it, while keeping representative points at least distance 1 apart. (c) The basic splitter gadget for APX-hardness for L_∞ placed on the half grid and showing two wires extending left and two right. The variable segment (in thick cyan) for the variable x_i has its representative point (the large red dot) at the right, which is the false setting. The representative points shown by large red/yellow dots are distance at least 1 apart in L_∞.

Wire. A wire is a long horizontal segment with 0-length segments at unit distances along it, except at its left and right endpoints. See Figure 3.7(b). The representative point for a 0-length segment must be the single point in the segment (shown as small red dots in the figure). As before, a wire is directed from the variable gadget to the clause gadget. We distinguish a "false setting" where the wire has its representative point within distance

1 of its forward end (at the clause gadget) and a "true setting" where the wire has its representative point within distance 1 of its tail end (at the variable gadget).

Clause Gadget. A clause gadget is a vertical segment. Three wires corresponding to the three literals in the clause meet the vertical segment as shown in Figure 3.7(b). There are 0-length segments at unit distances along the clause interval except where the three wires meet it.

Variable Gadget. A variable segment has length 3, with two 0-length segments placed 1 and 2 units from the endpoints. A representative point in the right half corresponds to a false value for the variable, and a representative point in the left half corresponds to a true value. In order to transmit the variable's value to all the connecting horizontal wires we build a "splitter" gadget. The basic splitter gadget for L_∞ is shown in Figure 3.7(c). The same splitter gadget works for the other norms but we can improve the lower bounds using modified splitter gadgets as described in the full version.

To obtain our claimed lower bounds, the basic splitter gadget of Figure 3.7(c) must be modified for the L_2 and L_1 norms. For L_2 we modify the spacing slightly. In particular (see Figure 3.9(c)), we overlap successive vertical segments by a small amount t. In order to keep the endpoint of each segment at distance 1 from the nearest 0-length segments, we must have $t \leq t^* = 1 - \sqrt{3}/2 \approx .13397$ as indicated by the large radius 1 circle in the figure. To get back on the grid, we increase the gap between the next two 0-length segment to $1 + t$. In order to have rational coordinates, we use $t = 2/15 = .133\dot{3}$. (Closer continued fraction approximations to t^* give marginal improvements in g_2.)

For L_1 we construct an alternate splitter gadget with crossing segments as shown in Figure 3.8. As in the L_2 case, the gap between a 0-length segment and a segment endpoint or another 0-length segment is increased in some cases, specifically from 1 to 4/3.

Reduction correctness

Proof of Claim 3.3.1. Suppose the formula Φ is satisfiable. We show that there is an assignment of representative points for the intervals in I such that the distance between any two points is at least 1 in the L_ℓ norm. We cannot do better than distance 1 because there are 0-length segments at distance exactly 1. Thus $\delta^*_\ell(I) = 1$.

For each variable segment, we place its representative point at the right endpoint if the variable is True in Φ, and at the left endpoint otherwise. Representative points for the other segments in the splitter gadget are placed at the endpoints of the segments as shown in Figure 3.7. See also Figure 3.9(c) for details of the L_2 case. For each wire, we place its representative point at the forward end (at the clause gadget) if the corresponding literal is false, and at the tail end (in the splitter gadget) if the corresponding literal is true. Each clause has a True literal—choose one and place the representative point for the clause segment at the place where the wire for this True literal meets it. See Figure 3.7 which shows that in all cases, the distance between any two representative points is at least 1. □

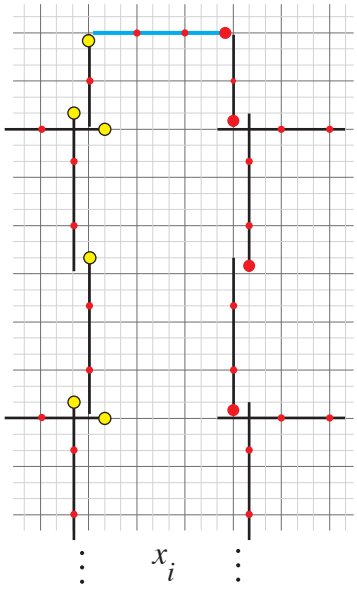


Figure 3.8: The splitter gadget for L_1 on a grid subdivided into thirds with two wires extending left and two right. The variable segment (in thick cyan) for the variable x_i has its representative point (the large red dot) at the right, which is the false setting. The representative points shown by large red/yellow dots are distance at least 1 apart in L_1.

Proof of Claim 3.3.2. Suppose the formula Φ is not satisfiable. Consider any assignment of representative points to the intervals of I. We will show that there are two representative points within distance $1/g_\ell$. Note that $1/g_1 = 1/g_\infty = 2/3$ and $1/g_2 = .69324....$ Observe that finding two points within distance $2/3$ suffices for all norms.

The representative points determine a truth value assignment $\mathcal{V}$ to the variables as follows: if a variable segment has its representative point in the right half, assign it True, otherwise assign it False. Since Φ is not satisfiable, there must be some clause C whose three literals are all false under the assignment $\mathcal{V}$. Suppose that C contains three positive literals (the case of three negative literals is symmetric).

Observe that if the representative point on a segment is placed in the unit gap between two 0-length segments, then there are two points within distance $1/2$, which is less than $1/g_\ell$. In the L_2 and L_1 splitters, we created longer gaps between successive 0-length segments. In the L_2 splitter, there are gaps of length $1+t$; a point in such a gap would cause two points to be within distance $(1+t)/2 = .566\dot{6}$ which is less than $1/g_2$. In the L_1 splitter there are gaps of length $4/3$; a point in such a gap would cause two points to be within distance $2/3 = 1/g_1$.

Thus we may suppose that every wire and every segment in a splitter gadget has its representative point "near" (i.e., within distance 1 of) one of its two endpoints, and that every clause segment has its representative point near an incoming wire. For the clause C (the one whose three literals are all false), suppose its representative point is near incoming wire w, and suppose w is associated with variable x. We now separate into two cases depending

whether the wire w has its representative point near the forward end or the tail end.

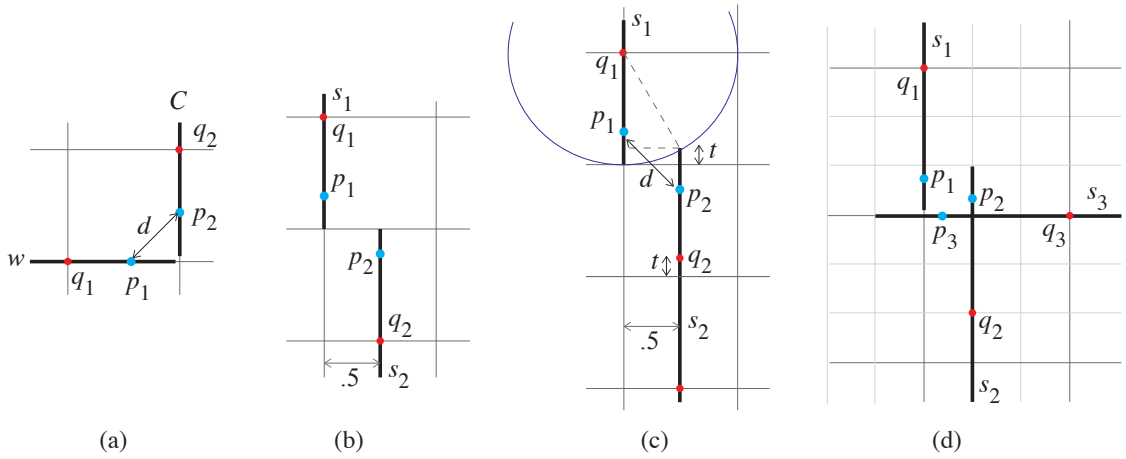


Figure 3.9: (a) A representative point p_2 on clause segment C near a representative point p_1 on wire w. (b) Representative points p_1 and p_2 on successive vertical segments of the L_∞ splitter. (c) Representative points p_1 and p_2 on successive segments of the L_2 splitter. In this splitter successive vertical segments overlap by amount t. The top endpoint of s_2 is not inside the unit circle centred at q_1, so long as $t \le t^* = 1 - \sqrt{3}/2$. (d) Representative points p_1, p_2, p_3 on successive segments of the L_1 splitter.

Case 1. The wire w has its representative point near the forward (clause) end. The situation is as shown in Figure 3.9(a), with representative point p_1 on the wire w and representative point p_2 on the clause segment. If p_i is within distance 2/3 from its nearest 0-length segment q_i, we are done. Otherwise consider $d_\ell(p_1, p_2)$, marked d in the figure. We have $d_\ell(p_1, p_2) \le d_1(p_1, p_2) \le 2/3$.

Case 2. The wire w has its representative point near its tail end. Recall that wire w is associated with variable x which is set False, i.e., the variable segment for x has its representative point near the right end.

In the splitter gadget there is a sequence of segments from the variable segment for x to the wire w. Somewhere along the sequence there must be two consecutive segments s_1 and s_2 with representative points p_1 and p_2 where p_1 is near the end of s_1 and p_2 is near the start of s_2.

If the endpoints of s_1 and s_2 meet at a right angle, then the analysis in Case 1 shows that there are two points within distance 2/3.

We separate the remaining cases by the norm. For the L_∞ norm the segments s_1 and s_2 must be vertical, as shown in Figure 3.9(b). If either point p_i is within distance 2/3 of its nearest 0-length segment q_i, we are done. Otherwise p_1 and p_2 lie in a rectangle of size $\frac{1}{2} \times \frac{2}{3}$ so their L_∞ distance is at most 2/3.

Next we consider the L_1 norm. See Figure 3.9(d), which shows segment s_1 and two possible following segments s_2 and s_3, together with possible representative points p_i on s_i

and the nearest 0-length segment q_i on s_i, $i = 1, 2, 3$. If p_i is within distance $2/3$ of q_i we are done. Otherwise p_1 and p_2 lie in a square of side-length $1/3$ so their L_1 distance is at most $2/3$. The same is true for p_1 and p_3.

Finally we consider the L_2 norm, see points p_1 and p_2 in Figure 3.9(c). This is the one case were we do not guarantee two points within distance $2/3$ but only within the higher bound $1/g_2 = .69324....$ There are three distances involved: from p_1 to the closest 0-length segment q_1 on s_1, from p_1 to p_2, and from p_2 to the closest 0-length segment q_2 on s_2. To maximize the minimum, the three distances should all have the same value d.

Then $d = \sqrt{(\frac{1}{2})^2 + (2 - t - 2d)^2}$, so $3d^2 - 4(2 - t)d + (2 - t)^2 + .25 = 0$. With $t = 2/15$ this solves to $d \approx .693237$ which is $\leq 1/g_2 = .69324....$

This completes the proof of the claim. □

Chapter 4

Imprecise 2.5D Terrains

This section will explore 2.5D terrains that have imprecise points. The input points are given imprecisely, as they were in Section 2, but now the points reside in 3-dimensional space. In particular, the x- and y-coordinates are precise, but the z-coordinates are only known within an interval range. The formal definition of the ***imprecise 2.5 terrain model***, first given by Gray et al. [21] is as follows:

Input: A set of n points $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ and a triangulation $\mathcal{T}$ of the points. In addition, a set of intervals $I_i = [b_i, t_i]$ associated with each (x_i, y_i). The points and intervals determine vertical line segments $\ell_i = \{x_i\} \times \{y_i\} \times [b_i, t_i]$. This input is called an ***imprecise 2.5D terrain***.

Output: A set of coordinates $z_1 \in I_1, \ldots, z_n \in I_n$, determining points $(x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)$. By connecting points based on the triangulation $\mathcal{T}$, one obtains a triangulated polyhedral surface, which we call a ***2.5D terrain***.

Objective function: Return a 2.5D terrain that optimizes some objective function.

The goal of this section is to examine our objectives in the imprecise 2.5D terrain model: Objectives #1, #2a, #2b, #3 and #4a. The algorithms used for 1.5D terrains from Section 2 will not easily extend to 2.5D terrains. In some cases we give new algorithms, or alternatively, we prove that the problem becomes NP-hard. We will also consider restricted settings where the triangulation is a path of triangles (i.e., the planar dual graph is a path) or a strip of triangles. The setting where the triangulation is a tree of triangles (i.e., we have a triangulated polygon) may also be worth exploring in the future. A summary of the results of this section can be found in Table 4.1.

In Section 4.1, we examine Objective #1, the problem of minimizing the number of extrema. We look at triangulations restricted to a path of triangles, and give a polynomial

time algorithm. In Section 4.2, we look at the objective of minimizing the number of bends in the triangulation (Objective #2b). We also define a notion of a patch (Objective #2a), which is a generalization of a link in 1.5D. We give an algorithm for triangulations that consist of a path of triangles, and give an NP-hardness proof for the general case, which both work for the two objective functions.

In Section 4.3, we look at Objective #3, the problem of minimizing the total surface area. And in Section 4.4, we look at Objective #4a, the problem of a minimizing the maximum steepness. For these last two problems, we reduce to Second Order Cone Programming.

Definition 4.0.1. A ***Second Order Cone Program*** (a SOCP) is a optimization problem of the following form:

Input:

- $m, n, n_1, \ldots, n_m, k \in \mathbb{N}$
- $A_i \in \mathbb{R}^{n_i \times n}$, $b_i \in \mathbb{R}^{n_i}, c_i \in \mathbb{R}^n, d_i \in \mathbb{R}$ for all $i = 1, \ldots, m$.
- $E \in \mathbb{R}^{k \times n}, f \in \mathbb{R}^k, w \in \mathbb{R}^n$

Output: $x \in \mathbb{R}^n$

Objective:

$$\begin{array}{ll} \text{minimize } w^\top x & \\ \text{subject to} & \|A_i x + b_i\| \leq c_i^\top x + d_i \quad i = 1, \ldots, m \\ & Ex = f \end{array}$$

where $\|\cdot\|$ represents the Euclidean (L_2) norm of the given vector.

Second Order Cone Programming lies between Convex Quadratically Constrained Quadratic Programming (Convex QCQP) [6] and Semidefinite Programming (SDP) [31]. Second Order Cone Programs can be solved with additive error ε in time polynomial in the size of the input and $\log(\frac{1}{\varepsilon})$ using interior point methods [39].

Notation and Definitions.

- Let n denote the number of points, and N denote the number of triangles.
- The ***path restricted*** version of the imprecise 2.5D terrain model is when the triangulation consists of a simple path of triangles. Formally, this means the dual graph of the triangulation graph (without a vertex for the outer face) is a path.

Objective function	Path of triangles	General case
#1 minimize number of extrema	$\boldsymbol{O(n^4)}$ [§4.1]	NP-hard [22]
#2 optimizing coplanar features	**5-approx*** [§4.2.2]	**NP-complete** [§4.2.3]
#3 minimize surface area	**SOCP**	**SOCP** [§4.3]
#4a minimize maximum steepness	**SOCP**	**SOCP** [§4.4]

Table 4.1: Algorithm or hardness results for the imprecise 2.5D terrain model. Bold results are original to this thesis.
*The algorithm requires the input to lie within a strip, see Section 4.2 for more details.

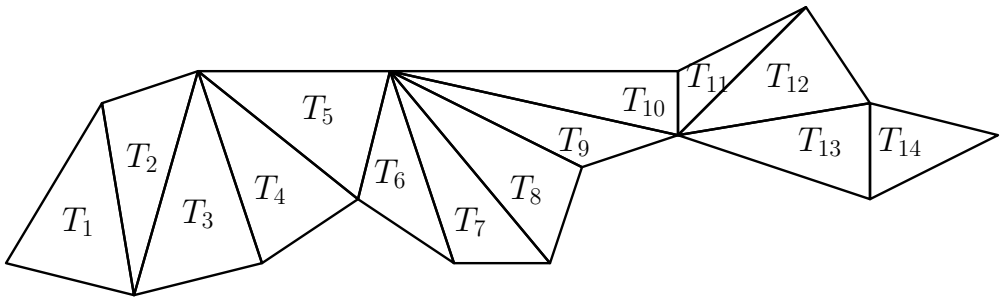


Figure 4.1: A sample input consisting of a path of triangles.

- The ***strip restricted*** version of the model is similar to the path restricted version, except all points lie on one of two planes. Specifically, all of the input y-coordinates are one of two fixed values. See Sections 4.2 and 4.2.2 for more details.
- For the path/strip restricted version of the model, we enumerate the triangles along the path $T_1, \ldots, T_N$. See Figure 4.1 for an example. In the general case, we will not enumerate the triangles, instead we use the notation $T \in \mathcal{T}$.
- For the path/strip restricted version of the model, let $\mathcal{L}_i$ be the subproblem consisting of the first i triangles.

4.1 Minimizing the number of local extrema

For our first problem, we will look at minimizing the number of local extrema along the 2.5D terrain. Recall in Section 2.1, we examined this problem for 1.5D terrains. We will use the definition given by Gray et al. in [22], which is:

Definition 4.1.1. In a precise 2.5D terrain, a ***plateau*** is a maximal collection of points C such that the points of C are connected together by the edges of the triangulation and all have the same elevation value.

Definition 4.1.2. In a precise 2.5D terrain, a ***local minimum*** (symmetric for ***local maximum***) is a plateau where any point connected by an edge to a point of the plateau must have a higher elevation than the plateau points.

Objective #1: Minimizing the number of local extrema.

Gray et al. [22] show that the problem of minimizing the number of local extrema is NP-hard for 2.5D terrains. Therefore, the focus of this section will be solving the problem for path restricted inputs.

Theorem 4.1.1. There is an $O(n^4)$ dynamic programming algorithm that solves the extrema problem for imprecise 2.5D terrains for path restricted inputs.

Preliminaries. We first discretize our problem by only allowing the elevation values to be chosen from a discrete set E. This is justified with the following claim:

Claim 4.1.1. Let $E = \{t_1, \ldots, t_n, b_1, \ldots, b_n\}$ denote the set of endpoints of the input intervals. Then there exists an optimal solution z^* so that $z_i^* \in E$ for all $i = 1, \ldots, n$.

The result of this claim is that we can discretize each interval I_i to be $I_i \cap E$ without worrying about missing the optimal solution.

Proof. Consider any optimal solution z^*. Suppose there are $k > 0$ elevation values not in E (multiple points can share the same elevation value). Let v be the *largest* elevation value that is not in the set E. We will show that we can modify the solution to shift all the points with elevation v to match another elevation value from E (thus reducing k) while keeping the solution optimal and not changing any other elevation values. Therefore, we can apply induction on k in order to obtain an optimal solution with all elevations in E.

Let w denote the *smallest* elevation value from E that is greater than v. This exists since $v \notin E$. Let D denote the set of points with this elevation (it may be empty). We will move the points of C up to have elevation w. Since w is the smallest value, then this new solution is feasible. We have clearly decreased k, and we claim the number of extrema will not increase with this move.

To show this, we will prove that any extremum in the new terrain resulted from at least one extremum from the original precise terrain. The extrema that are not at elevation w in the new terrain are clearly extrema in the original terrain as well, so we will focus on the extrema at elevation w. Let $M \subseteq C \cup D$ be such an extremum in the new terrain. We will focus on the case that M is a local maximum, M as a local minimum can be argued similarly.

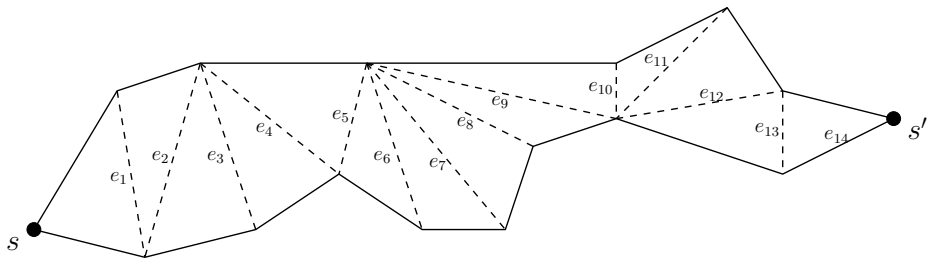


Figure 4.2: Labeling edges of the input triangulation, and defining s, s'.

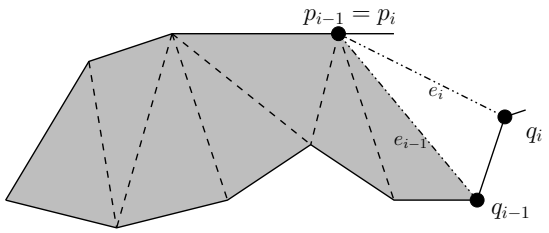


Figure 4.3: Computing a solution for the first i triangles from solutions for the subproblem in gray, the first $i-1$ triangles.

In the new terrain, every point that is adjacent to M has an elevation value less than or equal to v. If M only consists of points of C, then M is clearly a local maximum present in the original precise terrain as well. Otherwise, some points from $D \cap M$ form at least one local maximum in the original terrain.

Therefore, for every extrema M in the new terrain, there exists a subset of M representing an extrema in the original terrain. Therefore, the new terrain has at most as many extrema as the original one. □

As discussed in the notation and definitions section, we label the triangles $T_1, \ldots, T_N$ along the path. Additionally, for all $i = 1, \ldots, N-1$ define e_i to be the edge connecting triangles T_i and T_{i+1}. We also define e_N to be an arbitrary edge of T_N that is not e_{N-1}. See Figure 4.2 for an example. In a path triangulation, triangles T_1 and T_N each have a vertex that is not shared by any other triangles. Let us call those vertices s and s' respectively. The boundary of the triangulation splits into two chains between s and s'. Every edge e_i has one endpoint p_i on one chain and one endpoint q_i on the other chain. Triangles T_i and T_{i-1} will share one vertex of e_i: either $p_i = p_{i-1}$ or $q_i = q_{i-1}$. See Figure 4.3 for an example.

Recall that by Claim 4.1.1, we know we only need to look at a finite number of feasible solutions. A brute force algorithm would take $O((2n)^n)$ time, but using dynamic programming, we can improve this to polynomial runtime.

Algorithm. For each i, we will solve a set of subproblems where we choose possible elevations in E for each endpoint of edge e_i, and also choose whether each endpoint is a local min, local max, or neither. The solution to a subproblem records the number of extrema.

To capture this more precisely, we introduce some notation. Let $X = \{\uparrow, \downarrow, -\}$, where $\uparrow$ indicates a local maximum, $\downarrow$ indicates a local minimum, and $-$ indicates neither. Our subproblems have the form $S_i(v_1, m_1, v_2, m_2)$ where $v_1 \in E$ is the elevation of p_i, $v_2 \in E$ is the elevation of q_i, $m_1 \in X$ indicates the status of p_i and $m_2 \in X$ indicates the status of q_i. Thus the number of subproblems is $n \cdot |E|^2 \cdot 3^2 \in O(n^3)$. Note that not all subproblems are well defined. For instance, we cannot have $m_1 = \uparrow$ but $v_1 < v_2$ (i.e. cannot have p_i be part of a local maximum but be below q_i).

We claim that we can solve each of the $9|E|^2$ subproblems for e_i in $O(|E|)$ time using the solutions to subproblems for e_{i-1}.

Base case. The base case consists of one triangle T_1. To compute every $S_1(v_1, m_1, v_2, m_2)$, we simply try all possible values for the elevation of point s to generate at most $2n$ candidate solutions. For each candidate, we determine the extremum type of points p_1, q_1 and count the number of extrema, and save the solution in the appropriate $S_1(v_1, m_1, v_2, m_2)$.

Inductive step. We will look at how to compute $S_i(v_1, m_1, v_2, m_2)$ recursively. The triangle T_i consists of vertices p_i and q_i, and the third being either p_{i-1} (if $q_i = q_{i-1}$) or q_{i-1} (if $p_i = p_{i-1}$). Figure 4.3 shows this triangle in white. In this figure, q_{i-1} is the third vertex of T_i. We generate candidate solutions by having elevations determined by entries from S_{i-1}. To access these entries, we need the elevation values and extremum types for points p_{i-1}, q_{i-1}. We have the elevation value for p_{i-1} (since $p_{i-1} = p_i$ which has elevation v_1). Meanwhile, because q_i is adjacent to p_{i-1} and q_i is not a part of the first i triangles, then we cannot assume p_{i-1} has the same extremum type for $\mathcal{L}_{i-1}$ (the first $i-1$ triangles) as it does for $\mathcal{L}_i$ (the first i triangles). We will therefore try different possibilities for p_{i-1}'s extremum type. We also try different possible extremum types m and elevation values v for q_{i-1}. We will not have to look at *all* extremum types and elevation values for p_{i-1}, q_{i-1}, because not all cases are well defined for the given v_1, m_1, v_2, m_2 values.

We combine this *recursive partial solution* with the elevation value for the point that is not part of the first $i-1$ triangles (in Figure 4.3, this would be point q_i). This gives us a *candidate* solution.

For every candidate, we need to compute the number of extremum, which we want to do recursively in constant time. To do this, we take the number of extrema for the *recursive partial solution* and add at most 1 to it. What we add depends on whether q_i: (1) is a new extremum, (2) is joining an existing one, (3) changes the type of extrema that p_{i-1}, q_{i-1} belong to, or (4) is not part of an extremum. Determining whether to add 0 or 1 can be done in constant time. It is difficult to succinctly describe all of these cases in detail, but the basic idea is that q_i is only adjacent to $p_{i-1} = p_i$ and q_{i-1} (at least when only the first i triangles are considered), and so the elevation values and extremum types of p_i, q_i, q_{i-1} are the only ones that are relevant.

For condition (3), the extrema for p_{i-1}, q_{i-1} can only change from a maximum (or a minimum) to a neither type. The exception is when the total number extremum without

q_i is one, then p_{i-1} and q_{i-1} are part of one large plateau that is counted as one extremum (that is considered both a minimum and a maximum). However, this can easily be checked by seeing if $S_{i-1}(v_1, \uparrow, v, \uparrow) = S_{i-1}(v_1, \downarrow, v, \downarrow) = 1$. In this case, determining whether to add plus one or not depends on whether $v_2 \neq v_1$ or not.

We generate all candidates for $S_i(v_1, m_1, v_2, m_2)$, and keep the one with the fewest extrema. We also store elevation and extremum types for p_{i-1}, q_{i-1} in order to easily construct an optimal terrain after the table has been filled in.

Runtime. As stated earlier, there are $O(n^3)$ subproblems to compute. For each subproblem $S_i(v_1, m_1, v_2, m_2)$, the number of candidates generated is at most $3^2 \cdot 2n = O(n)$. This is because one of the endpoints of e_{i-1} has a known elevation value, so we only generate elevation values for only one of $z(p_{i-1})$ or $z(q_{i-1})$. For example, in Figure 4.3, the point $p_{i-1} = p_i$, so we already know what its elevation should be. Therefore, for each i the runtime is $O(n)$, which gives a total runtime of $O(n^4)$.

Correctness.

Lemma 4.1.1. For all $i = 1, \ldots, N$, each entry in S_i is correctly computed.

Proof. This will be proven by induction. The bases cases are obviously correct. For each subproblem $S_i(v_1, m_1, v_2, m_2)$ to compute, consider the actual optimal solution z^* for this subproblem. Consider taking off the point that is not in e_{i-1}. Using Figure 4.3 as an example, then this point is q_i. This gives a solution for the first $i-1$ triangles. We know the endpoints of e_{i-1} are at some elevations v_3, v_4 from E, and they are some type of extrema m_3, m_4 from X (the extrema types are with respect to the first $i-1$ triangles). Instead of using the elevation values from z^*, we should use $S_{i-1}(v_3, m_3, v_4, m_4)$ instead. This results in a solution with no more extrema than z^* if we ignore $z(q_i) = v_2$. Let us denote z^* without q_i as $\hat{z}$. We still need to account for q_i in the count of the number of extrema.

First, note that p_{i-1} has the same type of extrema and elevation in $\hat{z}$ as in $S_{i-1}(v_3, m_3, v_4, m_4)$. The same is true for q_{i-1}.

The addition of q_i changes the number of extrema depending on the following conditions for q_i: (1) it is a new extremum, (2) it is joining an existing one, (3) it changes the type of extrema that p_{i-1}, q_{i-1} belong to, or (4) it is not part of an extremum. Conditions (1),(2),(4) depend on the elevation value v_2 for q_i, and whether it is above or below points p_{i-1} and q_{i-1}, strictly in-between them, or at the same elevation as one of them. For condition (3), the extremum types of p_{i-1}, q_{i-1} are relevant. Since the two solutions we are comparing have the same elevation and extrema values for their p_{i-1}'s (and their q_{i-1}'s), then whichever conditions are true will be true for both solutions.

Therefore, the addition of q_i changes the number of extrema in both $S_{i-1}(v_3, m_3, v_4, m_4)$ and $\hat{z}$ by the same additive amount. Therefore, we have found a solution that is no worse than z^*. This solution is clearly a candidate that is generated by the algorithm, because it computes many candidates recursively from S_{i-1}, therefore, the algorithm is correct. □

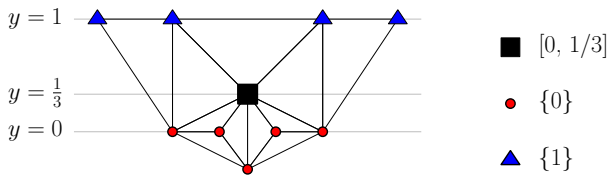


Figure 4.4: Counterexample to prove Claim 4.2.1 (from a bird's eye view). The central point (drawn with a big black square) is the only one with a non-trivial interval, viz. $[0, 1/3]$.

4.2 Minimizing the number of patches/bends

In this section, we generalize the problem of minimizing the number of links (Objective #2a) to 2.5D terrains. We define a ***patch*** to be a collection of triangles that are connected in the dual graph (i.e. share edges) and all coplanar. This is similar to the 1.5D link problem, because the links of the 1.5D terrain consists of multiple segments that are be connected together and are all *collinear*.

Objective function #2a: Minimizing the number of patches.

In 1.5D, minimizing the number of links is the same as Objective #2b, minimizing the number of bends. In 1.5D, bends are points where the *turn* angle between the two incident edges is not equal to 0. The generalization to 2.5D is to consider the *dihedral* angles between two triangles that share an edge. We say that there is a ***bend*** at this edge when the dihedral angle is not equal to 180°.

Objective function #2b: Minimizing the number of bends.

These two objective functions do not have the same optimum solutions in general. But they both have a similar goal of returning a terrain that have many adjacent coplanar triangles.

Claim 4.2.1. If the triangulation is outerplanar i.e., all vertices are on the outer face of the triangulation, then minimizing the number of patches is an equivalent objective to minimizing the number of bends. In the general case, the two objectives are not necessarily equivalent.

Proof. To prove the first part of the claim, for a precise 2.5D terrain with an outerplanar triangulation, let B denote the number of bends and let P denote the number of patches. We can prove by induction that $B + 1 = P$, and so the claim clearly follows from this. The base case is when $B = 0$, and the whole terrain is one patch, i.e. $P = 1$. Assume $B \geq 1$, and consider an arbitrary bend. We split the terrain along this edge, resulting in two 2.5D (outerplanar) terrains. The number of bends in the terrain is $B_1 + B_1 + 1$, where B_1, B_2 are the number of bends in the two smaller terrains. The plus one comes from the bend we

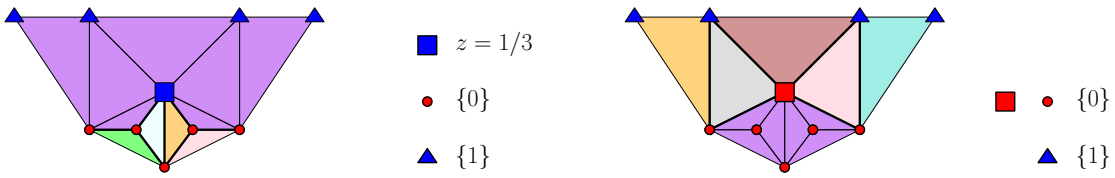


Figure 4.5: Solution (A) : If we use the top end of the black interval, we get 5 patches (optimal) and 7 bends. Solution (B) : If we use the bottom end of the black interval, we get 6 patches and 6 bends (optimal).

split at. Similarly, $P = P_1 + P_2$. By induction, $B_1 + 1 = P_1$ and $B_2 + 1 = P_2$, therefore, $B + 1 = P$, which proves the inductive step.

For the second part of the claim, see Figure 4.4 for a counterexample. There is only one non-trivial interval in the input. In Figure 4.5, the solution on the left minimizes the number of patches, and the right minimizes the number of bends.

The proof that these two solutions are optimal for their respective objective functions is quite simple. Consider any other solution where the one imprecise z value is something in $(0, \frac{1}{3})$. For any two triangles on the bottom portion (that is, below the imprecise point), three out of the four vertices forming these two triangles are non-collinear points with elevation 0, so the triangles can only be coplanar if the imprecise point is also set to 0. Similarly, for the top portion, two triangles that share an edge have three fixed non-collinear vertices in common, so there is a unique plane that could contain those triangles, and that is when $z = 1/3$. Hence, when $z \in (0, 1/3)$, we see that there must be 11 patches and 13 bends, i.e. it is not optimal. □

Results. In Section 4.2.2, we show that there is a 5-approximation algorithm that works for both objective functions for inputs restricted to a strip (where all input y-values are one of two fixed values). See Figure 4.7 for an example input within a strip. Before we do this, we warm up by looking at a special case where there is only a single imprecise point on one side of the strip, see Section 4.2.1. In Section 4.2.3, we give an NP-hardness reduction in the general setting. The same reduction works for the decision version of both objective functions.

4.2.1 An algorithm for fan triangulations

We will first consider a special case where only one point (called the *apex* point) is on one side of the strip. For such an input triangulation, we call it a ***fan***. See Figure 4.6 for an example input. We label the apex a. Every triangle has an edge not incident to a that we call the ***base*** of the triangle. The base edges all have the same y-coordinate so they line in a plane $y = c$. This plane is drawn as a thin red box in Figure 4.6.

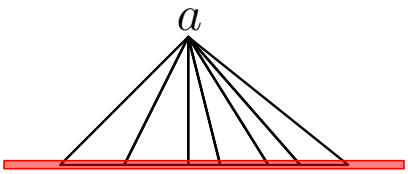


Figure 4.6: A bird's eyeview of an example of a 2.5D terrain with a triangulation consisting of a fan of triangles.

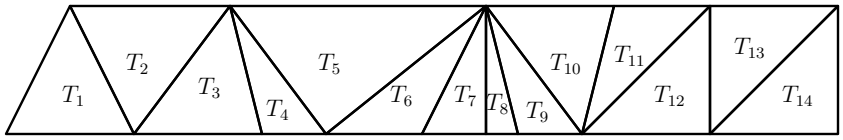


Figure 4.7: A path triangulation within a *strip*, which means the y values take on one of two values.

Theorem 4.2.1. There is a 2-approximation algorithm for the problem of minimizing the number of bends/patches when the input is restricted to a fan of triangles in a strip.

The algorithm will build upon the algorithm used in Section 2.2.3. Consider the base edges of all of the triangles. They form an imprecise 1.5D terrain. In essence, we can solve a problem in 1.5D by looking at the vertical line segments lying in the plane $y = c$. As far as the apex is concerned, we note the following observation and its corollary.

Observation 4.2.1. Fix a precise 2.5D terrain for the given fan input. Let T, T' be two adjacent triangles. Let s, s' be the base segments of T, T' (respectively). Then T, T' are coplanar if and only if s and s' are collinear.

From this observation we clearly see that:

Corollary 4.2.1. The choice of elevation for the interval at a is irrelevant. That is, whether two adjacent triangles are coplanar is not affected by the placement at a.

Proof of Theorem 4.2.1. If we want to 2-approximate the number of bends for this input, we should 2-approximate the 1.5D bends problem that lies in the plane $y = c$. We do this using the algorithm from Section 2.2.3. □

4.2.2 An algorithm for path triangulations within a strip

In this section we give a 5-approximation when the input consists of a path of triangles within a strip. See Figure 4.7 for a sample triangulation. Unfortunately, we do not see a way to solve the problem with the same generality as we did in Section 4.1 (there, the triangulation was not necessarily in a strip). We note that with this strip restriction, minimizing the

number of patches and minimizing the number of bends are equivalent objective functions by Claim 4.2.1.

Theorem 4.2.2. There is a 5-approximation algorithm for the problem of minimizing the number of bends/patches when the input is restricted to a strip.

Any 5-approximation algorithm for the problem of minimizing the number of bends will also give a 5-approximation for the optimal number of patches. Therefore, we will focus approximating the optimal number of bends going forward.

The algorithm will be somewhat similar to the algorithm used in Section 2.2.3 to approximate the minimum number of bends in an imprecise 1.5D terrain. In particular, we will greedily fix elevations to create a maximal sequence of coplanar triangles, then skip ahead to the next triangle whose elevations are not fixed, and repeat. Unlike in 1.5D, we may need to skip over more than a constant number of triangles, so we need an extra step to choose the elevations of the triangles that we skip—as it turns out, those triangles will form a fan, so we can use the above 2-approximation, but we need some additional steps to account for vertices that are involved in many triangles.

The algorithm is iterative and each iteration involves two steps.

We start the first iteration from the first triangle. To compute the patch containing T_1, we greedily compute it so that it covers as many triangles as possible. This is illustrated as a striped blue patch in Figure 4.8. We denote i_1 as the largest index of a triangle in this patch. To compute this patch, we return a feasible solution to a linear program. Let A, B, C be variables representing the plane $z = Ax + By + C$. Then the linear program is:

- $b_i \leq z_i \leq t_i$ for all vertices i of triangles $T_1, \ldots, T_{i_1}$
- $z_i = Ax_i + By_i + C$ for all vertices i of triangles $T_1, \ldots, T_{i_1}$

We find the largest value for i_1 using a binary search by running this linear program multiples times, decreasing i_1 whenever the linear program is infeasible and increasing i_1 whenever the linear program is feasible. This completes step 1 of the first iteration.

Let T_{j_1} be the last triangle that shares a vertex with T_{i_1}. Observe that $j_1 > i_1 + 1$ since T_{i_1+1} shares two vertices with T_{i_1}. In iteration 2, we will repeat step 1 on the triangles starting from T_{j_1+1}, since that triangle has none of its vertex elevations fixed. It remains to choose the elevations for the triangles $T_{i_1+1}, \ldots, T_{j_1}$. This is done in step 2 of the first iteration, which we now describe.

Observe that triangles $T_{i_1+2}, \ldots, T_{j_1-2}$ form a (possibly empty) fan f_1 whose apex is a vertex of T_{i_1}, and whose other vertices are on the opposite chain and are disjoint from T_{i_1} (whose vertices have already been fixed) and T_{j_1+1} (whose vertices will be fixed in the next iteration of step 1). See Figure 4.8 for an example, f_1 is illustrated in dark orange. Use

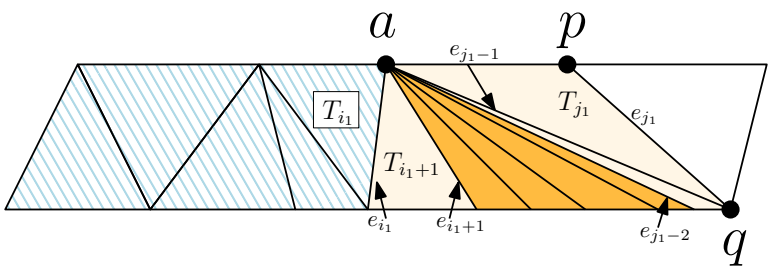


Figure 4.8: The first iteration of the algorithm.

Theorem 4.2.1 to choose the elevations of the non-apex vertices for f_1, which 2-approximates the minimum number of bends inside the fan.

In iteration 2, we compute index i_2, which represents the last triangle in the largest patch starting from T_{j_1+1}. We then look at the fan f_2 containing T_{i_2+2} up to T_{j_2-2}, and in a similar fashion as before, we do step 2 for this fan. We continue iterating to get additional index values $i_1, j_1, i_2, j_2, \ldots, i_m, j_m$ and fans $f_1, \ldots, f_m$. Note that after iteration m there may be an additional partial iteration where only step 1 occurs.

Correctness. We will prove that this algorithm 5-approximates the optimal number of bends.

Let B^* denote the optimal number of bends for the input. Let B denote the number of bends returned by the algorithm.

For each fan f_k, let s_k denote the number of bends in the fan f_k using the elevations computed by the 2-approximation algorithm from Section 4.2.1. Let s_k^* denote the optimal number of bends that can be achieved in the fan f_k. Then $s_k \leq 2s_k^*$ by Theorem 4.2.1. We will establish a upper bound on B using the s_k values, and a lower bound on B^* using the s_k^* values.

Lemma 4.2.1. $B \leq 5m + \sum_{k=1}^{m} s_k$

Proof. We can formally prove this by induction, but we will simply focus on the first iteration. Figure 4.8 illustrates the first iteration of the algorithm. There are s_1 bends strictly within the fan f_1. Outside the fan, there are at most 5 bends obtained from the first j_1 triangles (at edges $e_{i_1}, e_{i_1+1}, e_{j_1-2}, e_{j_1-1}, e_{j_1}$). Applying a simple induction argument over all iterations gives us this total. □

Lemma 4.2.2. $B^* \geq m + \sum_{k=1}^{m} s_k^*$.

Proof. For all $k = 1, \ldots, m$, there are at least s_k^* bends between triangles in each fan subproblem f_k that we solve. Additionally, for each patch $T_{j_{k-1}+1}, \ldots, T_{i_k}$, because i_k is maximal,

we know there must be (at least) one bend somewhere between triangles $T_{j_{k-1}+1}, \ldots, T_{i_k+1}$. Summing over all $k = 1, \ldots, m$ proves the lemma. □

From this, we conclude that:

$$B \;\leq\; 5m + \sum_{k=1}^{m} s_k \;\leq\; 5m + \sum_{k=1}^{m} 2s_k^* \;\leq\; 5m + 5\sum_{k=1}^{m} s_k^* \;\leq\; 5B^*$$

This completes the proof that we have a 5-approximation.

4.2.3 NP-hardness for the general setting

We will show that the objectives of minimizing the number of patches and minimizing the number of bends is NP-hard for the case of a general triangulation, using a reduction from Monotone Rectilinear Planar 3-SAT. Recall that these are not equivalent objective functions in general. However, it will not be hard to see that for the imprecise terrains generated by the polynomial reduction, NP-hardness for either objective follows through with the reduction.

Theorem 4.2.3. Minimizing the number of bends is NP-complete in the general setting. Using the same reduction, minimizing the number of patches is NP-complete in the general setting.

Both decision problems are in NP. For minimizing the number of bends, a non-deterministic algorithm for the decision version of the problem is as follows: Let k be the maximum number of bends we ask for. First, non-deterministically determine which of the k edges shall have a bend. For each *remaining* edge e, define linear programming variables a_e, b_e, c_e, with the intended meaning that the triangles on both sides of e lie on the same plane $a_e x + b_e y + z + c_e = 0$. Also define linear programming variables z_i for all imprecise points i. A solution in the feasible region for the following linear program will be a 2.5D terrain with k bends.

- $b_i \leq z_i \leq t_i$ for all i
- for all edges e where we *do not want a bend*, let i, j, k, l be the resulting indices of the imprecise points that form the two triangles sharing this edge. Then we want the points to be coplanar, so:
  - $a_e x_i + b_e y_i + z_i + c_e = 0$
  - $a_e x_j + b_e y_j + z_j + c_e = 0$
  - $a_e x_k + b_e y_k + z_k + c_e = 0$
  - $a_e x_l + b_e y_l + z_l + c_e = 0$

Since linear programming can be solved in polynomial time, then we clearly see that this is a non-deterministic polynomial time algorithm. For the patches problem, we can similarly non-deterministically choose the disjoint patches of triangles, and then solve for the z-values using linear programming.

Reduction details. The reduction will be from the NP-complete problem Monotone Rectilinear Planar 3-SAT [11], which is the same variant of 3-SAT used in the hardness reductions in Section 3. In Monotone Rectilinear Planar 3-SAT [11], each clause has either three positive literals or three negative literals, each variable v is represented by a thin vertical rectangle centered at the point $(0, y_v)$, each positive [negative] clause is represented by a thin vertical rectangle at a positive [negative, resp.] x-coordinate, and there is a horizontal line segment joining any variable to any clause that contains it. For n variables and m clauses, the representation can be on an $O(m) \times O(n+m)$ grid.

As was done for the reductions in Section 3.2, we first modify the representation of Monotone Rectilinear Planar 3-SAT so that each clause rectangle has fixed height and is connected to its three literals via three "wires"—the middle one remains horizontal, the bottom one bends to enter the clause rectangle from the bottom, and the top one bends twice to enter the clause rectangle from the far side. See Figure 4.9. Each wire is directed from the variable to the clause, and represents a literal. The representation is still on an $O(m) \times O(n+m)$ grid.

Given an instance of Monotone Rectilinear Planar 3-SAT Φ, we will construct an imprecise terrain, by constructing a set of imprecise points in 3D, and a triangulation of these points.

Variable gadget and variable component. The variable gadget for a variable v consists of four triangles. See Figure 4.10a for a bird's eye view. We call the two triangles in white the ***selector*** triangles. We call the green striped triangle the *true* triangle and the checkered triangle the *false* triangle. These four triangles share one vertex, which has x-coordinate 0 and y-coordinate y_v. Therefore, the variable gadget for variable v will be placed inside the rectangle corresponding to variable v in the input 3-SAT representation.

In the *true* and *false* triangles, the 5 vertices of these triangles all have 0 length z-intervals. Specifically, the intervals are chosen so that every *false* triangle must lie on the plane $z = 0$, and, every *true* triangle must lie on the plane $x - z = 0$. See Figure 4.10a for the exact x, y-coordinates and z-intervals. Note that the two selector triangles are forced to be coplanar to each other, regardless of what the z-coordinate of the leftmost vertex is chosen to be. (This is because the three points with x-coordinate 0 are collinear). As it will become clear later on, these two triangles should either both be coplanar to the *true* triangle, or both be coplanar to the *false* triangle, and these two choices amount to us choosing this variable to be true or false.

From one of the two leftmost triangles, create chains of triangles going off into horizontal and vertical directions. Figure 4.10b shows this, and shows how we can split off into multiple

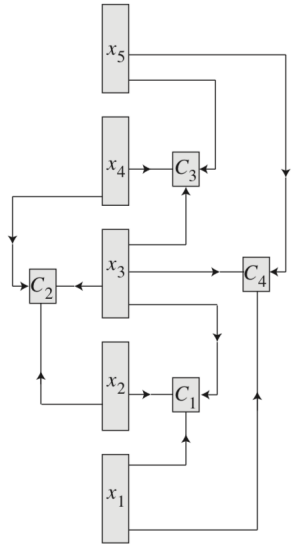


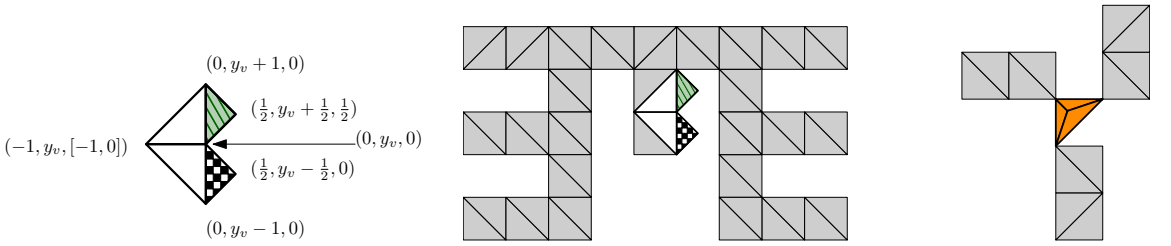
Figure 4.9: An instance of Monotone Rectilinear Planar 3-SAT, modified so the clauses have fixed height.

paths (allowing for multiple edges out of the Planar 3SAT vertex corresponding to this variable gadget). We call these solid grey triangles ***path*** triangles. Note all x, y coordinates are integers. We need to choose the intervals for the imprecise points of the path triangles. For each point (x, y), choose the z-interval to be $[x, 0]$ if $x < 0$ and $[0, x]$ if $x \geq 0$. In particular, all path triangles can be chosen to be coplanar to the *true* triangle: since the plane containing the *true* triangle is $x - z = 0$, then the $z-$coordinates of each point would need to be set to x. They also all can be chosen to be coplanar to the *false* triangle, by setting the z-coordinates of all points to 0. Choosing these triangles to be coplanar to the *false* triangle is like choosing the variable to be false ("false assignment"), and coplanar to the *true* triangle is like choosing the variable to be true ("true assignment"). After describing the whole instance, it will become clear that these two assignments for these triangles are the only ones that can be used in an optimal solution.

Definition 4.2.1. A *variable component* consists of a variable gadget and its corresponding path triangles.

Definition 4.2.2. The *false* assignment for a variable component sets the elevations of every imprecise point in the component to be the endpoint $z = 0$. The *true* assignment for a variable component sets the elevations of every interval in the component to be the opposite endpoint from the one that is used in the *false* assignment (for non-trivial intervals, the opposite is the non-zero endpoint).

Clause gadget. Consider a clause c containing three positive literals. (The case of three negative literals will be symmetric.) As part of the variable components, we have built



(a) Variable gadget (b) Variable component (c) Clause gadget

Figure 4.10: Reduction gadgets

chains of triangles following the edges of the Monotone Rectilinear Planar 3SAT instance, and three of those chains meet at a clause gadget, as illustrated in the Figure 4.10c. A clause gadget consists of three triangles (coloured in orange in the figure). Since the clause contains positive literals, it is to the right of the variable gadgets, and so the intervals assigned to the three incoming path triangles have intervals with non-negative values.

The vertical line segment in the centre of the clause will have its bottom endpoint be much higher than the surrounding segments. Let us say the top left corner of this clause gadget is at coordinate (x, y). The line segment in the center of the clause is set to be: $(x + 1/3, y - 1/3, [\frac{1}{3}x, x + 1/3])$. We will claim that if we use the *true* assignments for (at least) one of the three variable components of this clause, then the point on the centre line segment can be chosen so that the triangles of the clause gadget are all coplanar. However, for the one truth-value assignment when all three literals are false, every possible placement on the centre segment will result in the three triangles of the clause gadget not being coplanar to each other. See Lemma 4.2.5 in the correctness section for more details.

We design the clause gadget for three negative literals in a similar fashion. The centre of the clause gadget will instead use the line segment $(x - 1/3, y - 1/3, [2/3x - 1/3, 0])$, where (x, y) are the coordinates of the top right corner of the gadget.

Completing the triangulation. We combine all of the variable components and clause gadgets together. The final result is not yet a triangulation, as there may be some polygonal faces (i.e., "holes"). To fix this, place a point at every integer x, y coordinate inside a face, and set the corresponding z-intervals to be $[0, 0]$. We will then add the half-integer points $(i + 0.5, j + 0.5)$ for $i, j \in \mathbb{Z}$ that are within a polygonal face. For these points, we assign them a very low z-interval. Specifically, let $Z > 0$ be the largest absolute value of an interval endpoint created so far. Set the corresponding intervals of the half-integer points to be $[-4Z, -4Z]$. There are polynomially many new points, because they have only been added in the closed faces which have polynomial size. Triangulate in an X-pattern, see Figure 4.11. These newly added triangles are shaded blue in the figure and have dotted edges.

However, we triangulate in a slightly different way near the variable and clause gadgets, see Figure 4.11. For those points inside an integer grid square, they are chosen to have a fixed z-values that is very low, in a similar fashion to the half-integer points.

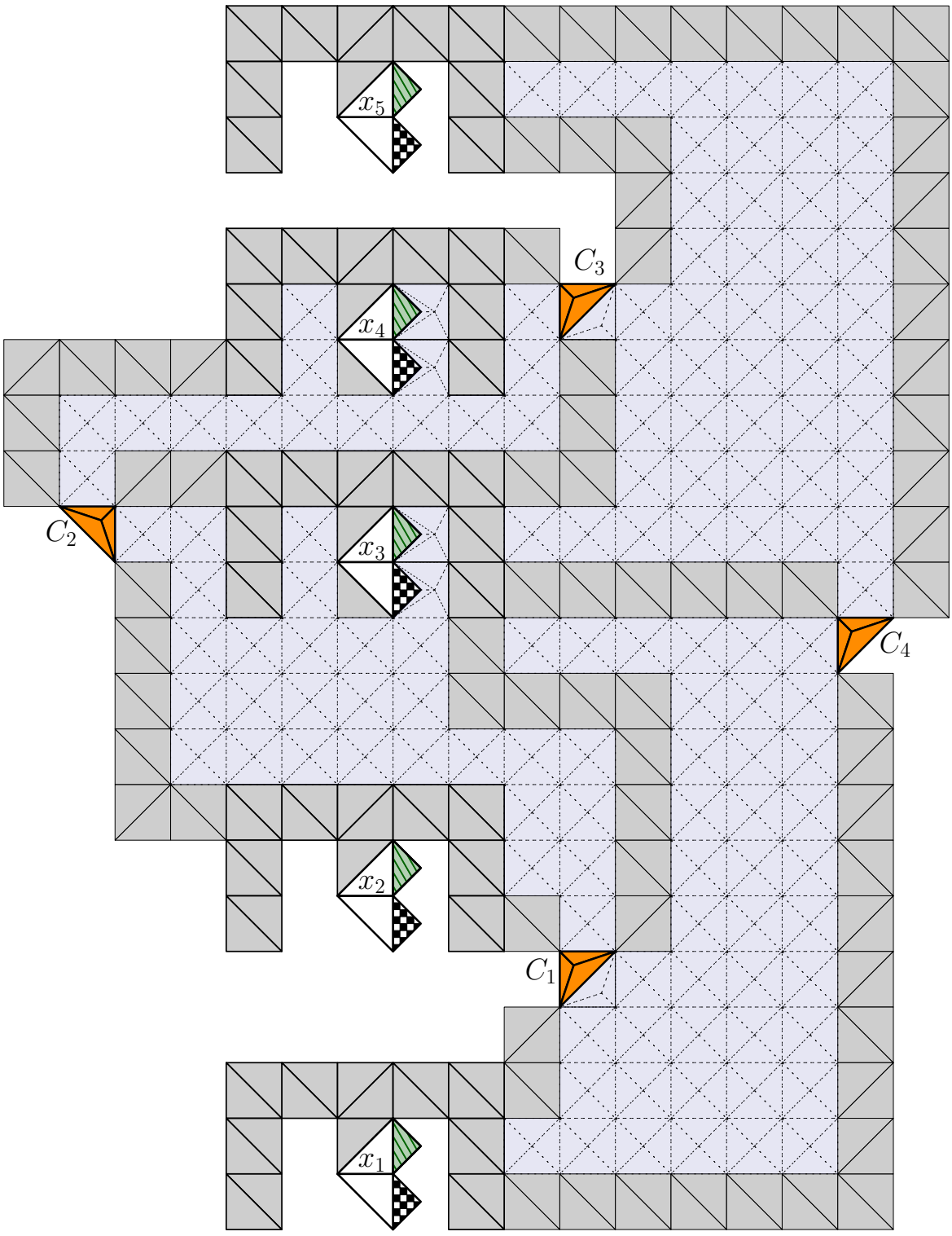
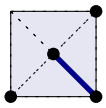
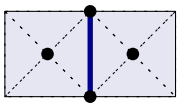


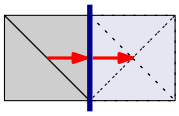
Figure 4.11: The resulting imprecise 2.5D terrain constructed from the instance of 3-SAT in Figure 4.9.



(a) Case 1



(b) Case 2



(c) Case 3

Figure 4.12: Proving Lemma 4.2.3. The general idea is to show the triangles sharing the thick dark edge cannot be coplanar. In Cases 1 and 2, we prove this by showing the four points labelled in Figure 4.12a and 4.12b cannot be made coplanar. In Case 3, we prove this by showing the two red vectors cannot be made collinear.

We call these blue triangles with dotted edges *spike* triangles. While many of the spike triangles have precise vertices, the ones that share an edge with a path triangle have imprecise vertices. Observe that for any precise terrain, all the points have a z-value between $-Z$ and Z, except the points we added at half-grid points, which are forced to have z value $-4Z$. We call the spike triangles that are not right angled *special*.

Lemma 4.2.3. A spike triangle cannot be coplanar with an adjacent triangle.

Proof. We consider all cases, depending on the type of the triangle that the spike triangle is adjacent to. A few of the cases are illustrated in Figure 4.12. Case 4 deals with the special spike triangles, while the remaining cases deal with the non-special (right-angled) spike triangles.

Case 1. First, consider two neighbouring *spike* triangles within the same "square" (that is, the shared edge is one of the edges that make up the X in the X-pattern). Without loss of generality, we will compare the bottom right edge between the bottom triangle and the right triangle. See Figure 4.12a. Suppose the 2 triangles are coplanar (so the 4 points illustrated in Figure 4.12a are coplanar). Then the middle point would lie on the line segment (in 3D) from the bottom left point to the upper right point, which is impossible since the middle point is the only global minimum at elevation $-4Z$.

Case 2. We also can compare *spike* triangles that are part of different X's, as illustrated in Figure 4.12b. We can easily argue the four points in Figure 4.12b are not coplanar. The argument for this case is quite similar to the argument for Case 1, so it is omitted.

Case 3. We compare an edge between a *spike* triangle and a non-*spike* triangle. Consider the example Figure in 4.12c, which shows an edge between a path triangles and a spike triangle. (The other triangle need not be a path triangle for this argument to work). To verify these triangles are not coplanar, we will compare the slopes of the two red horizontal vectors lying on the two triangles. If they are not collinear, then we have proven the triangles are not coplanar.

The change in x and y between the two endpoints of the vector are the same for both vectors (taking the value of the right endpoint minus the left endpoint, we get $\Delta x = 0.5, \Delta y = 0$). We will look at the change in z for both segments. For the one on the path triangle, recall the intervals of the triangle are contained in $[-Z, Z]$, and in particular, that means any point any path triangle will have an elevation between $-Z$ and Z. Therefore, the change in z for the first vector is at least $-Z - (Z) = -2Z$.

For the second vector, its right endpoint has z-coordinate exactly $-4Z$. Since the left endpoint is also on the path triangle, it must have elevation at least $-Z$ (since $-Z$ is the lowest possible elevation for path triangles). So the change in z is at most $-3Z$. Therefore, the change in z for the two vectors cannot be equal, so the two vectors are not collinear. Therefore, the triangles are not coplanar.

Case 4. The cases above handle the majority of spike triangles, which all have the same shape. A similar argument can be made for the special spike triangles. Proving this lemma for these triangles is not really any more difficult than the previous cases, but it would be repetitive, so the proof is omitted. □

Useful lemmas for correctness. Before we state the lemmas we need, we introduce some preliminary notation and definitions. Recall that there are n variables and m clauses. Recall that a variable component is the variable gadget plus its corresponding path gadgets.

For each *variable component* v, let b_v [p_v] be the number of bends [patches] inside the variable component for v. For each clause c, let b_c [p_c] be the number of bends [patches] inside the clause gadget for c. Note that boundary edges of variable components are always bends by Lemma 4.2.3, so b_v only counts interior edges of variable components. Note that the values b_v, p_v, b_c, p_c depend on the placement of elevations z.

Observe that every triangle in a component only shares edges with other triangles in the same component, and potentially some spike triangles. Thus by Lemma 4.2.3, all spike triangles have a bend at each of its edges and all spike triangles are in their own patch. This results in the following claim.

Claim 4.2.2. Let B [P] denote the number of bends [patches] for the given precise terrain. Then $B = \sum_v b_v + \sum_c b_c + S$ and $P = \sum_v p_v + \sum_c p_c + s$, where S denotes the number of edges where at least one incident triangle is a spike triangle, and s denotes the number of spike triangles.

We will look at each individual gadget independently in order to determine how many bends and patches there are in total.

Lemma 4.2.4. The following two statements are both equivalent and true.

1. Every variable component needs at least two patches i.e., $p_v \geq 2$, and the *true* and *false* variable component assignments are the only two assignments where this is achievable.

2. Every variable component needs at least one bend i.e., $b_v \geq 1$, and the *true* and *false* variable component assignments are the only two assignments where this is achievable.

Proof. First, we note that the two claims are exactly equivalent, because $p_v = b_v + 1$ (this is a result of the variable component being outerplanar and by Claim 4.2.1). Therefore, two patches means there is exactly one bend, and three or more patches means there are two or more bends.

We will prove the first version of the claim. The true triangle and false triangle cannot be coplanar to each other, and as a result, there will be at least two patches. It is obvious that using the *true* or *false* assignment for this variable component will gives us exactly two patches.

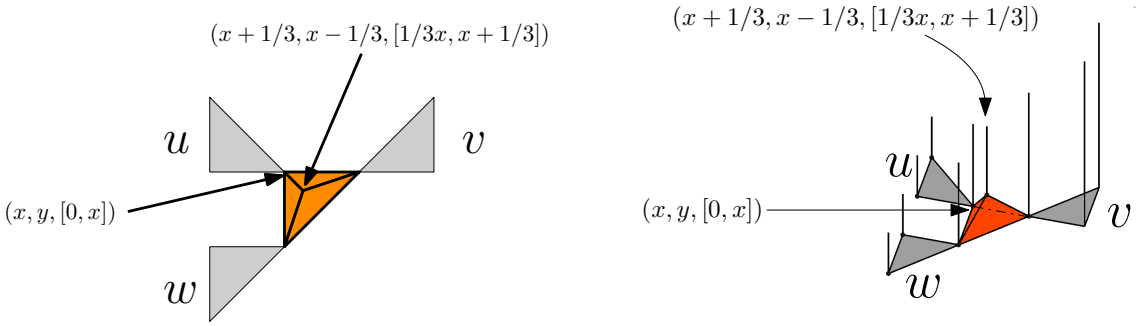
For any other assignment, we will show it must have at least three patches. We can break into two cases. If the entire variable component without the *true* and *false* triangles are all one patch, then this patch will not be coplanar to the *true* triangle or to the *false* triangle, because that would be the true or false variable assignments. Therefore, there will be three patches in this case. On the other hand, if the entire variable component without the *true* and *false* triangles is not one patch, then consider one of these patches that does not have either of the selector triangles in them. This must exist: in particular, the two selector triangles that are part of the variable gadget are always in the same patch together, so there must be a different patch without these two triangles, and so this patch is not connected to the *true* and *false* triangles. This patch, plus the patch containing the *true* and *false* triangles gives us three distinct patches, therefore, there are at least three patches in this case as well. □

Next, we consider an individual clause gadget. For its three corresponding variable components, we consider the 8 combinations of true/false assignments for the three variables, and then consider the choice for where to place the centre point of the clause gadget.

Lemma 4.2.5. Let b_c [p_c] denote the number of bends [patches] used *inside* the clause gadget for clause c, assuming c has positive literals.

1. Assume the variable components that are connected to the clause gadget of c all use the *false* variable assignment. Then the three triangles of the clause gadget will never be coplanar to each other, therefore $b_c = 3$ [$p_c = 3$] for all placements of the centre point.

2. Assume at least one of the variable components that are connected to the clause gadget of c uses the *true* variable assignment. Then the three triangles of the clause gadget can be chosen to coplanar to each other, i.e. $b_c = 0$ [$p_c = 1$].



(a) A birds eyeview of a clause gadget.

(b) The same clause gadget in 3D.

Figure 4.13: Visualizing the vertical line segments of near the clause gadget. Variable components u, v, w meet up at the three triangles forming the clause gadget.

We can make a similar claim for a clause with negative literals: for every clause c with negative literals, $b_c = 3$ $[p_c = 3]$ when all of the incoming variable components use the true assignment (i.e. all literals are false). Otherwise, we can choose the elevation of the point on the center segment so that the three triangles are coplanar to each other, i.e. $b_c = 0$ $[p_c = 1]$.

Proof. A clause gadget can be visualized in 3D, see Figure 4.13. All of the points in this figure have a z-interval with bottom endpoint 0 and top endpoint being the point's x-coordinate, except the point in the centre of the clause gadget, whose bottom endpoint is slightly higher than 0.

When all three of the variable components u, v, w use the *false* variable assignment, then the z-values for all of the vertices of these components are to equal 0. The point we place on the centre line segment must have an elevation greater than 0, so the three triangles forming the clause gadget cannot be made coplanar to each other. See the top left figure in Figure 4.14.

Meanwhile, when at least one of the variable component uses the *true* variable assignment, then the z-values for the vertices of this component are set to the top endpoint value. There are seven cases to consider where at least one variable component uses the *true* variable assignment. These are visualized with the remaining figures of Figure 4.14. As the figures illustrate, the three triangles making up the clause gadget can be made coplanar to each other, if the point on the centre line segment is chosen properly. □

Correctness. We will now prove correctness of the reduction. The two lemmas that follow prove correctness for the two objectives.

Lemma 4.2.6. Let $k = n + S$. Then there is a satisifiable assignment τ for Φ if and only if there is a selection of elevations z where $B \leq k$.

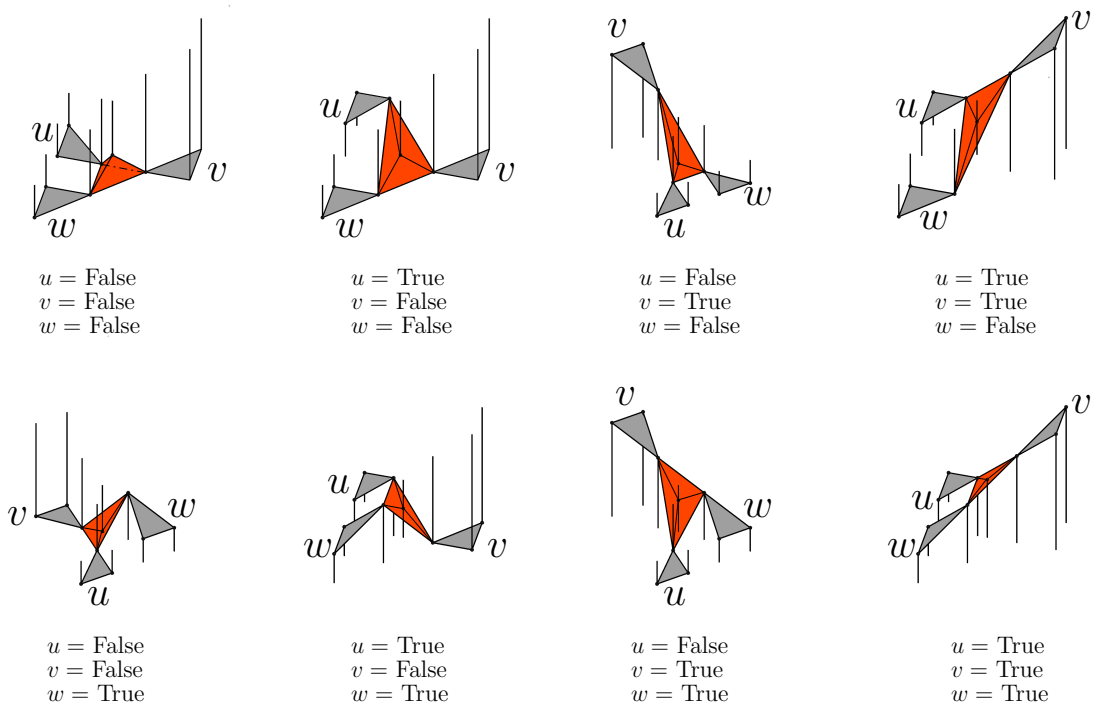


Figure 4.14: Eight cases for the clause gadgets. In all but the top-left figure, the three triangles of the clause gadget (in orange) are coplanar to each other. Therefore, choosing the three incoming variable gadget to all use the *false* assignment causes extra patches/bends.

Proof. If there is a satisfiable assignment for Φ, then for each true [false] variable, use the *true* [*false*] assignment for its corresponding variable component, which gives $\sum_v(1) = n$ bends by Lemma 4.2.4. By Lemma 4.2.5, we can choose the centre point of each clause gadget so that the clause gadget uses zero bends. The remaining edges must be part of at least one *spike* triangle. By Lemma 4.2.3, all of these edges are bends, so we get S additional bends from these edges. Therefore, we have a solution with $n + S = k$ bends.

Assume that there is a solution with $B \leq k$. First note that $B \geq k$ for *any* choice of precise points, because by using the lower bounds for b_v obtained from Lemma 4.2.4, $B = \sum_v b_v + \sum_c b_c + S \geq \sum_v 1 + S = n + S = k$. Therefore the solution with $B \leq k$ must have $b_v = 1$ and $b_c = 0$ for all variables v and clauses c. By Lemma 4.2.4, we must use one of two variable assignments for each variable component, which we can translate into a variable assignment τ for Φ. And since $b_c = 0$, by Lemma 4.2.5, this shows each clause is satisfied under this variable assignment. □

Lemma 4.2.7. Let $k = 2n + m + s$. Then there is a satisifiable assignment τ for Φ if and only if there is a selection of elevations z where $P \leq k$.

Proof. If there is a satisfiable assignment for Φ, then for each true [false] variable, use the *true* [*false*] assignment for its corresponding variable component, which gives $\sum_v(2) = 2n$ patches by Lemma 4.2.4. By Lemma 4.2.5, we can choose the centre point of each clause gadget so that the clause gadget uses one patch (for a total of m patches). By Lemma 4.2.3, spike triangles are always in their own patch, so we get s patches from these triangles. Therefore, we have a solution with $2n + m + s = k$ patches.

Assume that there is a solution with $P \leq k$. First note $P \geq k$ for *any* choice of precise points, because by using the lower bounds for p_v obtained from Lemma 4.2.4, and since clearly all $p_c \geq 1$, we get $P = \sum_v p_v + \sum_c p_c + s \geq \sum_v 2 + \sum_c 1 + s = 2n + m + s = k$. Therefore the solution with $P \leq k$ must have $p_v = 2$ and $p_c = 1$ for all variables v and clauses c. By Lemma 4.2.4, we must use one of two variable assignments for each variable component, which we can translate into a variable assignment τ for Φ. And since $p_c = 1$, by Lemma 4.2.5, this shows each clause is satisfied under this variable assignment. □

4.3 Minimizing the surface area

The objective function for this section is very intuitive:

Objective #3: Minimizing the surface area.

We will show this problem can be solved using Second Order Cone Programming (see Definition 4.0.1).

Theorem 4.3.1. There is a Second Order Cone Program (a SOCP) that solves the problem of minimizing the total surface area.

The z values will be variables used in the SOCP. We add the linear constraints that $b_i \leq z_i \leq t_i$ for all $i = 1, \ldots, n$ to ensure that each elevation value is within its interval.

For each triangle T, we introduce a variable s_T which represents the area of T. The constraint $\text{area}(T) \leq s_T$ ensures s_T is an upper bound to the area. This is the only constraint for s_T. The objective function is to minimize $\sum_{T \in \mathcal{T}} s_T$, so in the optimal solution to this SOCP, clearly the s_T will be chosen to be equal to $\text{area}(T)$. Therefore, the optimal solution to this SOCP will return an optimal value z for the problem of minimizing the total surface area.

Constants: $x_1, y_1, b_1, t_1, \ldots, x_n, y_n, b_n, t_n$

Variables: $z_1, \ldots, z_n, \{s_T : T \in \mathcal{T}\}$

$$\begin{aligned} &\text{minimize} && \sum_{T \in \mathcal{T}} s_T \\ &\text{subject to} && b_i \leq z_i \leq t_i, \quad i = 1, \ldots, n \\ & && \text{area}(T) \leq s_T \quad T \in \mathcal{T} \end{aligned}$$

We will now prove the constraints $\text{area}(T) \leq s_T$ are valid SOCP constraints. Let us say T has imprecise vertices $p_{i_1}, p_{i_2}, p_{i_3}$. Then the area of this triangle is $\frac{1}{2}\|(p_{i_2} - p_{i_1}) \times (p_{i_3} - p_{i_1})\|$, where $\times$ denotes the cross product between two vectors, and $\|\cdot\|$ represents the Euclidean (L_2) norm of the given vector.

We claim that $\|(p_{i_2} - p_{i_1}) \times (p_{i_3} - p_{i_1})\| = \left\| M_T \begin{bmatrix} z_{i_1} \\ z_{i_2} \\ z_{i_3} \end{bmatrix} + E_T \right\|$, where

$$M_T := \begin{bmatrix} y_{i_2} - y_{i_3} & y_{i_3} - y_{i_1} & y_{i_1} - y_{i_2} \\ x_{i_2} - x_{i_3} & x_{i_3} - x_{i_1} & x_{i_1} - x_{i_2} \\ 0 & 0 & 0 \end{bmatrix}$$

and

$$E_T := \begin{bmatrix} 0 \\ 0 \\ (x_{i_2} - x_{i_1})(y_{i_3} - y_{i_1}) - (y_{i_2} - y_{i_1})(x_{i_3} - x_{i_1}) \end{bmatrix}$$

This easily follows from the definition of cross product. According to the definition of SOCP, $\text{area}(T) \leq s_T$ is clearly a SOCP constraint.

4.4 Minimizing the maximum steepness

The goal of this section will be to generalize the objective of minimizing the maximum steepness for 1.5D terrains. A commonly used notion of steepness of a plane or triangle in 3D is the magnitude of its gradient.

Definition 4.4.1. The *steepness* of triangle T lying on the plane $z = f(x, y) = A_T x + B_T y + C_T$ is the L_2 norm of the gradient of f, i.e., $\sqrt{A_T^2 + B_T^2}$.

Note that in 1.5D, this sort of definition amounts to taking the absolute value of the slope of the line that the edge of the terrain lies on, which is exactly the definition we used in Section 2.4.

Objective function #4a: Minimizing the maximum steepness over all triangles.

Theorem 4.4.1. There is a Second Order Cone Program (a SOCP) that solves the problem of minimizing the maximum steepness.

We introduce variables A_T, B_T, C_T for each triangle T, with the goal of having the triangle T lie on the plane $z = A_T x + B_T y + C_T$. In particular, the three vertices of the triangle uniquely define the plane, so we will simply add the constraint that each vertex $i \in T$ lies on this plane.

Constants: $x_1, y_1, b_1, t_1, \ldots, x_n, y_n, b_n, t_n$

Variables: $z_1, \ldots, z_n, F, \{A_T, B_T, C_T : T \in \mathcal{T}\}$

$$
\begin{array}{lll}
\text{minimize } F & & \\
\text{subject to} & b_i \le z_i \le t_i, & i = 1, \ldots, n \\
& z_i = A_T x_i + B_T y_i + C_T & T \in \mathcal{T}, i \in T \\
& \sqrt{A_T^2 + B_T^2} \le F & T \in \mathcal{T}
\end{array}
$$

This shows that this problem can be solved using a SOCP.

Chapter 5

Conclusion

5.1 Concluding remarks

We started by discussing four objectives for the imprecise terrain model and how they exhibit realistic and nice properties. We explored these objectives in the imprecise 1.5D terrain model and obtained positive results: polynomial time algorithms for all objectives. For Objectives #2a and #2b, we settled for a 2-approximation, however this algorithm is very efficient (linear time).

Thinking about line segments in the plane proved to be quite interesting, so we explored the distant representatives problem for segments in the plane, and obtained new results.

For the imprecise 2.5D terrain model, we showed that each problem has some positive algorithmic result, even if it is only in a restricted setting. We also gave an NP-hardness reduction that works for both Objectives #2a, #2b in the general case.

5.2 Future work

There are a couple of interesting questions left from this thesis. First, we could not give an exact algorithm for Objectives #2a and #2b. Is there an exact algorithm for minimizing the number of links/bends in 1.5D, or is the problem NP-hard? How about minimizing the number of patches/bends for path triangulations in 2.5D? Can the algorithm for a strip be extended to a more general path triangulation? For the extrema and patches/bends problems, what can be said about outerplanar triangulations, which are slightly more general than path triangulations? Or in the general case, are there approximation algorithms? What approximation ratio can be achieved? For the problem of maximizing the minimum steepness in 2.5D, is there an algorithm, or is the problem NP-hard?

This thesis explored problems where the elevations of points are imprecise, but there is a fixed triangulation. We now suggest a few alternatives. First, the points are precise,

but the triangulation is not given. De Kok et al. [13] explored the Objective #1 for this model, for which they showed this problem is NP-hard. Are any of the other objectives NP-hard in this model? This model becomes significantly easier when the boundary of the terrain is known and no point lies in the interior of the polygon. We conjecture that dynamic programming algorithms would be quite easy for all of the objectives in this restricted model. These algorithms would take a similar approach to the dynamic programming algorithm for triangulating a simple polygon with minimum weight [20]. Are there parameterized results for any of the objectives explored in this thesis? A recent result by Hoffman and Okamoto [26] counted the number of interior points to the input polygon as a parameter, and gave a parameterized result for their problem.

Second, consider the model where the elevations of points are imprecise, *and* the triangulation is not given. Without a fixed triangulation forced upon us, this model allows us to ask more advanced questions about the input points. For instance, what if the input points are a large sample, but too many points were collected to feasibly save? Instead, we could triangulate a subset of points, and then we return a precise terrain where all the points we excluded still lie on the 2.5D terrain. This suggests that we have chosen a good sample to represent the points. Formally, the objective is to pick a subset and a 2.5D terrain of these points that ensures all the points lie on the terrain, and to choose the subset as small as possible (or, choose the subset and the triangulation so that the terrain has as few triangles as possible). This is an interesting problem in this model that might be worth exploring.

References

[1] Helmut Alt and Leonidas J. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. In *Handbook of Computational Geometry*, pages 121–153. Elsevier, 2000.

[2] Christoph Baur and Sándor P. Fekete. Approximation of geometric dispersion problems. *Algorithmica*, 30(3):451–470, 2001.

[3] Therese Biedl, Anna Lubiw, Anurag Murty Naredla, Peter Dominik Ralbovsky, and Graeme Stroud. Dispersion for intervals: A geometric approach. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, January 11-12, 2021*, pages 37–44. SIAM, 2021.

[4] Therese Biedl, Anna Lubiw, Anurag Murty Naredla, Peter Dominik Ralbovsky, and Graeme Stroud. Distant representatives for rectangles in the plane. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[5] Therese Biedl, Anna Lubiw, Anurag Murty Naredla, Peter Dominik Ralbovsky, and Graeme Stroud. Distant representatives for rectangles in the plane. *arXiv preprint arXiv:2108.07751*, 2021.

[6] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[7] Kevin Buchin, Maarten Löffler, Aleksandr Popov, and Marcel Roeloffzen. Uncertain Curve Simplification. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:22, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[8] Sergio Cabello. Approximation algorithms for spreading points. *Journal of Algorithms*, 62(2):49–73, 2007.

[9] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(3):485–524, 1991.

[10] Yi-Jen Chiang and Roberto Tamassia. Optimal shortest path and minimum-link path queries between two convex polygons inside a simple polygonal obstacle. *International Journal of Computational Geometry & Applications*, 7:85–121, 1997.

[11] Mark de Berg and Amirali Khosravi. Optimal binary space partitions in the plane. In My T. Thai and Sartaj Sahni, editors, *Computing and Combinatorics*, volume 6196 of *Lecture Notes in Computer Science*, pages 216–225, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[12] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer Science & Business Media, 2000.

[13] Thierry De Kok, Marc Van Kreveld, and Maarten Löffler. Generating realistic terrains with higher-order delaunay triangulations. *Computational Geometry*, 36(1):52–65, 2007.

[14] Reza Dorrigiv, Robert Fraser, Meng He, Shahin Kamali, Akitoshi Kawamura, Alejandro López-Ortiz, and Diego Seco. On minimum-and maximum-weight minimum spanning trees with neighborhoods. *Theory of Computing Systems*, 56(1):220–250, 2015.

[15] Rodney G. Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

[16] Anne Driemel, Herman Haverkort, Maarten Löffler, and Rodrigo Silveira. Flow computations on imprecise terrains. *Journal of Computational Geometry*, 4(1):38–78, 2013.

[17] Adrian Dumitrescu and Minghui Jiang. Dispersion in disks. *Theory of Computing Systems*, 51(2):125–142, 2012.

[18] Jiří Fiala, Jan Kratochvíl, and Andrzej Proskurowski. Geometric systems of disjoint representatives. In *International Symposium on Graph Drawing*, volume 2528 of *Lecture Notes in Computer Science*, pages 110–117. Springer, 2002.

[19] Jiří Fiala, Jan Kratochvíl, and Andrzej Proskurowski. Systems of distant representatives. *Discrete Applied Mathematics*, 145(2):306 – 316, 2005.

[20] Peter D Gilbert. New results on planar triangulations. Technical report, Illinois Univ. at Urbana-Champaign Applied Computation Theory Group, 1979.

[21] C. Gray and W. Evans. Optimistic shortest paths on uncertain terrains. In *Proceedings of the 16th Canadian Conference on Computational Geometry (CCCG'04)*, pages 68–71, Montréal, Canada, 2004.

[22] Chris Gray, Frank Kammer, Maarten Löffler, and Rodrigo I. Silveira. Removing local extrema from imprecise terrains. *Computational Geometry*, 45(7):334–349, 2012.

[23] Chris Gray, Maarten Löffler, and Rodrigo I Silveira. Minimizing slope change in imprecise 1.5D terrains. In *Proceedings of the 21st Canadian Conference on Computational Geometry (CCCG'2009)*, pages 55–58, Vancouver, Canada, 2009.

[24] Chris Gray, Maarten Löffler, and Rodrigo I Silveira. Smoothing imprecise 1.5D terrains. *International Journal of Computational Geometry & Applications*, 20(04):381–414, 2010.

[25] Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1-4):209–233, 1987.

[26] Michael Hoffmann and Yoshio Okamoto. The minimum weight triangulation problem with few inner points. *Computational Geometry*, 34(3):149–158, 2006.

[27] Hiroshi Imai and Masao Iri. Polygonal approximations of a curve — formulations and algorithms. In Godfried T. Toussaint, editor, *Computational Morphology*, volume 6 of *Machine Intelligence and Pattern Recognition*, pages 71–86. North-Holland, 1988.

[28] Vahideh Keikha, Maarten Löffler, Ali Mohades, and Zahed Rahmati. Width and bounding box of imprecise points. In *Proceedings of the 30th Canadian Conference on Computational Geometry (CCCG'18)*, pages 142–148, Winnipeg, Canada, 2018.

[29] J. Mark Keil and Tzvetalin S. Vassilev. Algorithms for optimal area triangulations of a convex polygon. *Computational Geometry*, 35(3):173–187, 2006.

[30] Yury Kholondyrev. *Optimistic and Pessimistic Shortest Paths on Uncertain Terrains*. PhD thesis, University of British Columbia, 2007.

[31] Miguel Sousa Lobo, Lieven Vandenberghe, Stephen Boyd, and Hervé Lebret. Applications of second-order cone programming. *Linear Algebra and its Applications*, 284(1):193–228, 1998.

[32] Maarten Löffler and Jack Snoeyink. Delaunay triangulation of imprecise points in linear time after preprocessing. *Computational Geometry*, 43(3):234–242, 2010.

[33] Maarten Löffler and Marc van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235–269, 2010.

[34] Maarten Löffler and Marc van Kreveld. Largest bounding box, smallest diameter, and related problems on imprecise points. *Computational Geometry*, 43(4):419 – 433, 2010.

[35] Hanan Luss. On equitable resource allocation problems: A lexicographic minimax approach. *Operations Research*, 47(3):361–378, 1999.

[36] Wolfgang Mulzer and Günter Rote. Minimum-weight triangulation is NP-hard. *J. ACM*, 55(2), May 2008.

[37] Włodzimierz Ogryczak. On the lexicographic minimax approach to location problems. *European Journal of Operational Research*, 100(3):566–585, 1997.

[38] Joseph O'Rourke. Polyhedra of minimal area as 3D object models. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence-Volume 2*, pages 664–666, 1981.

[39] Florian A. Potra and Stephen J. Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1):281–302, 2000. Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations.

[40] Jean-Marc Robert and Godfried Toussaint. Computational geometry and facility location. In *Proc. International Conference on Operations Research and Management Science*, pages 11–15, 1990.

[41] M.J.M. Roeloffzen. Finding structures on imprecise points. Master's thesis, TU Eindhoven, 2009.

[42] David Salesin, Jorge Stolfi, and Leonidas Guibas. Epsilon geometry: building robust algorithms from imprecise computations. In *Proceedings of the Fifth Annual Symposium on Computational Geometry*, pages 208–217, 1989.

[43] Barbara Simons. A fast algorithm for single processor scheduling. In *19th Annual Symposium on Foundations of Computer Science*, pages 246–252. IEEE, 1978.

[44] Subhash Suri. A linear time algorithm for minimum link paths inside a simple polygon. *Computer Vision, Graphics, and Image Processing*, 35(1):99–110, 1986.

[45] Csaba D. Toth, Joseph O'Rourke, and Jacob E Goodman. *Handbook of Discrete and Computational Geometry*. CRC press, 2017.

[46] Marc van Kreveld, Maarten Löffler, and Lionov Wiratma. On Optimal Polyline Simplification Using the Hausdorff and Fréchet Distance. In Bettina Speckmann and Csaba D. Tóth, editors, *34th International Symposium on Computational Geometry (SoCG 2018)*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 56:1–56:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.