Approximation Algorithms for Guarding 1.5 Dimensional Terrains

by

James Alexander King

B.Math., University of Waterloo, 2003

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Computer Science)

We accept this thesis as conforming to the required standard

. . . .

. . . .

THE UNIVERSITY OF BRITISH COLUMBIA

August, 2005

Abstract

The 1.5-dimensional terrain guarding problem gives an x-monotone chain (the terrain) and asks for the minimum set of vertices of the terrain (guards) such that every vertex of the terrain is seen by at least one guard. This terrain guarding problem is a restriction of the Set Cover problem, which is known to be both NP-complete and inapproximable within a sub-logarithmic factor [19]. Fortunately, it is known that the 1.5-dimensional terrain guarding problem is approximable to within a constant factor [5, 11], though no attempt has been made to minimize the approximation factor.

We give a 4-approximation algorithm for the general 1.5D terrain guarding problem, as well as a 2-approximation algorithm for the case with upward-looking guards, *i.e.* when guards cannot see vertices that are below themselves.

Contents

Abstract v

Contents vii

List of Figures ix

Acknowledgements xi

1 Introduction 1
- 1.1 Problem Statement 2
- 1.2 Motivation 4
  - 1.2.1 Applications 4
  - 1.2.2 Minimizing the Approximation Factor 4
  - 1.2.3 Failure of Trivial Methods 4
- 1.3 Related Work 6
  - 1.3.1 The Art Gallery Problem 6
  - 1.3.2 1.5-Dimensional Terrain Guarding 6
  - 1.3.3 2.5-Dimensional Terrain Guarding 8
  - 1.3.4 Watchtower Problems 8
- 1.4 Organization 9

2 Preliminaries 11
- 2.1 Terminology and Notation 12
- 2.2 Elementary Lemmas 12

3 Upward Looking Guards 15
- 3.1 Introduction to the Upward Looking Case 16
- 3.2 The TG-Up Algorithm 16
- 3.3 Modifications for TG-TT-Up 18
- 3.4 Time Complexity 18

4 The General Case 19
- 4.1 Introduction to the General Case 20
- 4.2 Preliminaries 20
- 4.3 Finding a Good Left Vertex 23
- 4.4 The Terminal Case 24
- 4.5 The Recursive Case 24

4.6 Modifications for TG-TT 25
4.7 Time Complexity 25

5 Domination of Directed Acyclic Graphs 29
5.1 Motivation 30
5.2 NP-Hardness and Inapproximability 30

6 Future Work 33
6.1 NP-Completeness 34
6.2 Characterization of Terrain Graphs 34
6.3 Reductions Between Restricted Visibility Problems 35

Bibliography 37

List of Figures

1.1 An example of a 1.5D terrain. d can see b, c, and e but not a or f. 2
1.2 A terrain for which the greedy method achieves a logarithmic approximation factor [4]. 5

2.1 The shaded areas are terrain free and their union contains $\overline{ad}$. . . 13

4.1 The order of some vertices related to an open vertex v. 21
4.2 $L'(v)$ and $R(v)$ must be in the shaded region. 22
4.3 The shaded region is terrain free, so every left vertex in $[L(v), R'(v)]$ must see v. 23
4.4 The nested interval $[L(y), R(L(y))]$ can be handled independently with the help of a dominant outside vertex. 25

Acknowledgements

I would like to thank my supervisor, Will Evans, for his guidance throughout my Master's degree and in particular for this project. I would also like to thank my second reader, David Kirkpatrick, for his many helpful comments. I am very grateful to Boaz Ben-Moshe for taking the time to assist me with this problem. I would like to thank all of my colleagues at UBC, especially the students and faculty of the BETA lab. Finally, I would like to thank Melinda for all of her love and support.

Chapter 1

Introduction

1.1 Problem Statement

In the 1.5-dimensional terrain guarding problem we are given as input a terrain T that is an x-monotone polygonal chain. An x-monotone polygonal chain is a polygonal chain that intersects any vertical line at most once. It can be thought of as an array of n vertices in 2-dimensional space sorted in ascending order of x-coordinate, where edges 'connect the dots' from left to right. Note that the x-monotonicity requires x-coordinates to be distinct.

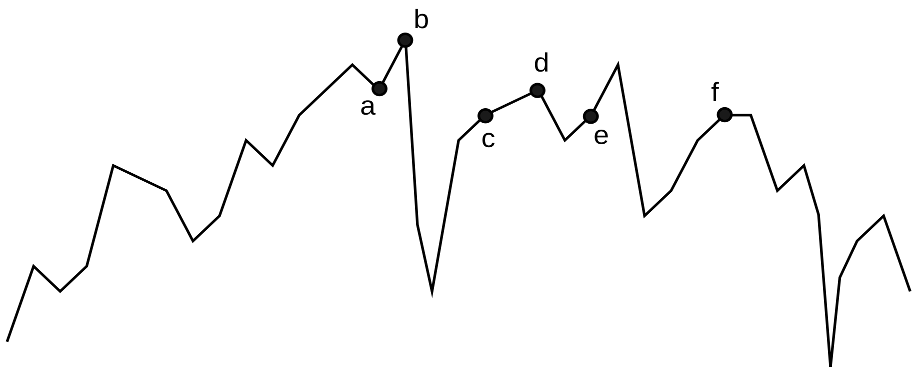

Figure 1.1: An example of a 1.5D terrain. d can see b, c, and e but not a or f.

We say that a point on the terrain sees another point on the terrain if there is a line of sight between them, *i.e.* the line segment connecting them is never strictly below T. A guard is simply a point on the terrain that we add to a 'guarding set'. Given a terrain T, we are asked for the smallest possible guarding set, *i.e.* the smallest set G of points on T such that every point on T is seen by some point in G.

Along with the terrain T, we are given two (possibly infinite) subsets of the points on T: S_G, the set of points on which we are allowed to place guards (*i.e.* the set of points that are allowed to be in our guarding set), and S_T, the target set of points that our guarding set must see. With these sets in mind, we can be slightly more formal and say that the problem is to find the smallest subset G of S_G such that every point in S_T is seen by some $g \in G$.

For a terrain T, we use $V(T)$ to denote the vertex set of T. There are two different versions of this terrain guarding problem that are natural to consider:

1. TG-VV: $S_G = V(T)$ and $S_T = V(T)$.
2. TG-TT: $S_G = T$ and $S_T = T$.

In this paper our main focus is the problem TG-VV in which guards must be placed on vertices and only vertices need to be guarded; this can be thought of as the discrete version of the problem. We will often refer to this problem simply as TG.

Ben-Moshe *et al.* [5] point out a reduction from TG-TT to TG-VV (see Section 1.3.2 for details). The algorithms we present actually work for both versions of the problem, though minor modifications are required to apply them to TG-TT.

1.2 Motivation

1.2.1 Applications

Naturally, the motivation for 1.5D terrain guarding comes from guarding or covering terrain. The 1.5D case appears, for example, when guarding or covering a road, perhaps with security cameras or street lights.

The 2.5D case has more powerful applications, most notably for providing a wireless communication network that covers a given region. Its proven intractibility and inapproximability, however, motivate us to look towards the 1.5D case for insight. It is also applicable, for example, if we only need to cover the path between two points on a polyhedral terrain. It has been pointed out [5] that the 1.5D terrain guarding problem can be utilized in heuristic methods for the 2.5D case.

1.2.2 Minimizing the Approximation Factor

The recent results of Ben-Moshe *et al.* [5] and Clarkson and Varadarajan [11] showed that constant-factor approximation algorithms exist for TG. Efforts to design an exact polynomial-time algorithm for TG have been unsuccessful and it is very possible that no such algorithm exists. If TG is NP-hard, then the best algorithm running in polynomial time will be the approximation algorithm with the lowest approximation factor. For this reason there is significant motivation to minimize the approximation factor.

1.2.3 Failure of Trivial Methods

The greedy algorithm for Set Cover, which achieves the optimal approximation factor of $O(\log n)$, repeatedly picks the set that contains the most uncovered elements. Similarly, the natural greedy algorithm for terrain guarding repeatedly picks the guard that sees the most unguarded vertices. It is not at all obvious that this method would not achieve a constant approximation factor. However, Ben-Moshe [4] provides the following example on which the greedy algorithm achieves a logarithmic approximation factor.

Observe the terrain in Figure 1.2. There are $O(\log n)$ 'bowls'. Each bowl contains 3 times as many vertices as the bowl to its right. Half of the vertices of each bowl are seen by a and the other half are seen by b. The minimum guarding set for the terrain is $\{a, b\}$, but the greedy algorithm will pick the circled vertices from left to right to achieve an approximation factor of $O(\log n)$.

There are other natural greedy-like algorithms that one might consider. For example, one could repeatedly pick the guard that maximizes the leftmost unguarded vertex or the lowest unguarded vertex. Terrains exist for these algorithms that prove they do not achieve constant approximation factors. The

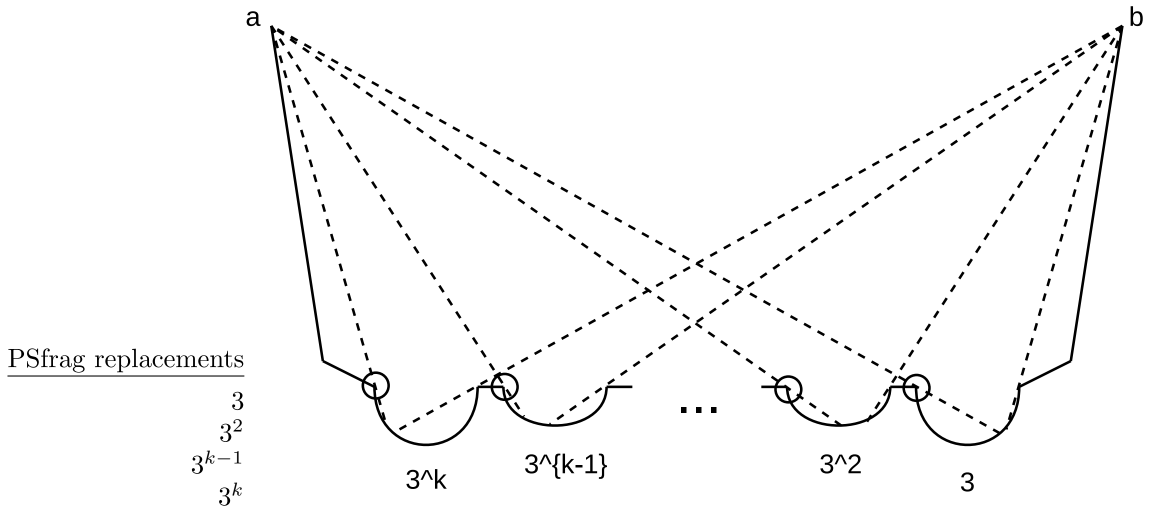

Figure 1.2: A terrain for which the greedy method achieves a logarithmic approximation factor [4].

apparent absence of simple algorithms that achieve constant approximation factors motivates us to consider more sophisticated techniques.

1.3 Related Work

1.3.1 The Art Gallery Problem

The 1.5D terrain guarding problem is similar to a well-studied polygon guarding problem known as the *art gallery problem.* This problem, posed by Victor Klee in 1973, asks for the minimum number of security cameras (that can rotate to obtain a full field of vision) required to guard a given art gallery. The art gallery is represented by a simple polygon and cameras are points inside the polygon. A camera guards a point if the line segment between them is contained within the polygon. The size n of a problem instance is the number of vertices of the polygon.

Chvátal's *Art Gallery Theorem* [10] states that $\lfloor \frac{n}{3} \rfloor$ cameras are always sufficient and sometimes necessary for guarding a polygon. Kooshesh and Moret [23] give a linear-time algorithm for guarding a polygon with $\lfloor \frac{n}{3} \rfloor$ cameras that is based on 3-colouring the triangulated polygon. Finding the minimum number of cameras required to guard a given polygon is NP-hard, as proven by Aggarwal [2].

The variation of the art gallery problem in which cameras must be placed on the perimeter of the polygon is more closely related to 1.5D terrain guarding. $\lfloor \frac{n}{3} \rfloor$ cameras are still sufficient and sometimes necessary, and Kooshesh and Moret's algorithm works in this variation since it actually places cameras on vertices. This variation is still NP-complete as proven by Lee and Lin [24].

Eidenbenz *et al.* [17] prove that the art gallery problem is APX-hard, regardless of restrictions on guard placement. This means that there is some positive constant δ such that no polynomial-time algorithm with an approximation factor less than $1 + \delta$ exists. In other words, the problem does not admit a polynomial-time approximation scheme unless P=NP. They provide an even stronger result in the case of polygons with holes: no polynomial-time algorithm can have a sub-logarithmic approximation factor.

1.3.2 1.5-Dimensional Terrain Guarding

When guarding a 1.5D terrain it is easy to see that $\lfloor \frac{n}{2} \rfloor$ guards are sufficient (place a guard at every other vertex) and sometimes necessary (*e.g.* if every vertex is on the upper hull of the terrain). If only the vertices of the terrain need to be guarded this bound drops to $\lceil \frac{n}{3} \rceil$.

Every instance of TG is an instance of Set Cover, but we know that Set Cover is NP-complete (see, *e.g.*, [20]) and no sub-logarithmic approximation factor can be obtained unless NP $\subseteq$ DTIME($n^{\log \log n}$) [19]. In general it is not particularly difficult to modify a TG-VV algorithm to solve instances of TG-TT, though this often involves some polynomial increase in time complexity.

It is unknown whether or not TG is NP-hard. In 1995 Chen *et al.* [9] proposed an NP-hardness proof obtainable via a modification of Lee and Lin's proof [24]. However, the proof, whose details were omitted, was never completed successfully. Since then, attempts to find a polynomial-time algorithm for TG and attempts to prove that it is NP-hard have both been unsuccessful.

The first constant-factor approximation algorithm for the 1.5D terrain guarding problem was given by Ben-Moshe *et al.* [5]. Their algorithm works by first placing guards to divide the terrain into independent subterrains. Each subterrain has the property that it does not require internal guards, *i.e.* every unguarded vertex can be seen from outside the subterrain. For each such subterrain that is not completely guarded they then proceed with steps that either reduce the subterrain or split it into multiple independent subterrains. They made no attempt to minimize their algorithm's approximation factor; as such it is very large (at least 48). It could be brought down possibly as low as 6 with some minor modifications and careful accounting, but due to the inevitable cost incurred by repeatedly dividing the terrain it does not seem that it could be brought any lower than 6.

Our approach differs from that of Ben-Moshe *et al.* in that ours is completely iterative. By avoiding a divide and conquer approach we are able to get the approximation factor as low as 4. We need to work differently than they do their iterative steps since we cannot depend on the desirable properties that their division offers.

Ben-Moshe *et al.* also give a reduction from TG-TT to TG-VV. They show that a quadratic number of extra vertices can be added in the middle of any terrain's edges to create a terrain in which any set of vertices that guards every vertex must guard the entire terrain. They then show that any edge guard can be replaced by two vertex guards, essentially saying that a c-approximation algorithm for TG-VV can be applied to one of these augmented terrains to give a $2c$-approximation algorithm for TG-TT.

Another constant-factor approximation algorithm is given by Clarkson and Varadarajan [11]. Consider a partition of a 1.5D terrain into maximal intervals such that, for any two points p and p' in a given interval, the leftmost point that sees p and the leftmost point that sees p' are the same. If we label each interval with the leftmost point that sees it and read the labels from leftmost interval to rightmost interval, Clarkson and Varadarajan note that we end up with an $(n, 2)$ Davenport-Schinzel sequence [25]. Such a sequence must have length at most $2n$. This characterization of the lack of complexity in 1.5D terrains allows them to efficiently find appropriate ϵ-nets [21] for instances of TG. They then apply the Set Cover method of Brönnimann and Goodrich [8] to solve the problem using these ϵ-nets. The end result is a constant-factor approximation algorithm that runs in polynomial time.

The 1.5D terrain guarding problem becomes easy if, instead of being placed on the terrain, all guards 'float' above the terrain at a fixed altitude that is above the highest vertex. Eidenbenz [14] gives a linear-time algorithm for finding an optimal set of guards in this case. The problem also becomes easy if guards can only look rightwards. Chen *et al.* [9] give a linear-time algorithm for this case.

1.3.3 2.5-Dimensional Terrain Guarding

A 2.5D terrain is a polyhedral surface that intersects every vertical line at most once and whose projection onto the x, y-plane is a simple polygon with no holes. The 2.5D Terrain Guarding Problem is therefore a natural extension of the 1.5D problem to the next dimension.

Bose *et al.* [7] prove that $\lfloor \frac{n}{2} \rfloor$ vertex guards are sufficient and sometimes necessary. If guards can be placed on edges then $\lfloor \frac{n}{3} \rfloor$ guards are sufficient [18] and $\lfloor \frac{4n-4}{13} \rfloor$ are sometimes necessary. Efficient algorithms for achieving these bounds for vertex guards and edge guards are given by Bose *et al.* [6].

Finding a minimum number of guards for a 2.5D terrain is NP-complete and Eidenbenz shows that it cannot be approximated within a sub-logarithmic factor unless NP $\subseteq$ DTIME($n^{\log \log n}$) [15]. Eidenbenz *et al.* show that the problem is also NP-complete and equally inapproximable when guards 'float' at a given altitude that is higher than the highest point in the terrain [16] (recall that this can be solved in linear time for 1.5D terrains).

1.3.4 Watchtower Problems

A k-watchtower problem provides a terrain and an integer k and asks for the minimum height h such that k guards can be placed at height h above the terrain (not above "sea level") to guard the terrain. The discrete version of the problem is that in which the guards must be above vertices of the terrain.

Most of the best results for these problems when $k = 2$ are due to Agarwal *et al.* [1]. For 1.5D terrains they present polynomial-time algorithms for both the discrete and continuous versions of the 2-watchtower problem. For 2.5D terrains they present a polynomial-time algorithm for the discrete 2-watchtower problem. Zhu gives an $O(n \log n)$ algorithm for the 1-watchtower problem on 2.5D terrains.

1.4 Organization

The rest of the paper is organized as follows. In Chapter 2 we introduce notation and some small but fundamental lemmas. In Chapter 3 we give our 2-approximation algorithm for the upward-looking case. In Chapter 4 we give our 4-approximation algorithm for the general case. In Chapter 5 we prove NP-hardness and inapproximability of DOMINATING SET on directed acyclic graphs to show that our 2-approximation algorithm for the upward-looking case is significant. In Chapter 6 we discuss open problems and suggest directions for future work regarding 1.5D terrain guarding.

Chapter 2

Preliminaries

2.1 Terminology and Notation

An instance of the 1.5D terrain guarding problem is simply an x-monotone chain T. This chain is a set of n vertices, $\{v_1, \ldots, v_n\}$, such that v_i is to the right of v_j if and only if $i < j$. We compare vertices using their indices; this makes it much easier to describe relationships between them. For example, $x < y$ means that x is left of y, and for a set S of vertices, $\max(S)$ is the rightmost vertex in S. When referring to points on the terrain that are not vertices we use the comparators in a similar manner.

For a vertex x we use $L(x)$ to denote the leftmost point where we can place a guard that sees x and $R(x)$ to denote the rightmost point where we can place a guard that sees x. It is not difficult to see that $L(x)$ and $R(x)$ will always be vertices in the general case, whether x is a vertex or not (this is not necessarily true in the upward-looking case, as we explain in Section 3.3). We use $T_L(x)$ to denote the terrain restricted to the interval $[v_1, x]$ and use $T_R(x)$ to denote the terrain restricted to $[x, v_n]$. We use $CH_L(x)$ to denote the convex hull of $T_L(x)$ and use $CH_R(x)$ for that of $T_R(x)$.

If a vertex x sees every unguarded vertex that another vertex y sees we say that x *dominates* y. We can also say that a set S dominates a vertex x if every unguarded vertex seen by x is also seen by some vertex in S. We say that x dominates y with respect to a certain region of T if x sees every unguarded vertex in that region that y sees.

We consider a minimum guarding set G_{OPT} for the terrain T. We assume there is some mapping g of vertices of T to guards in G_{OPT} such that, for a vertex v, $g(v)$ is a guard in G_{OPT} that sees v. We say that $g(v)$ is the guard *responsible for* g. g is surjective but never injective (since $|G_{OPT}| < n$); we use it to simplify the explanation of our accounting scheme. Our algorithms are iterative; in each iteration they find some appropriate unguarded vertex v. They then place a constant number of guards that together dominate $g(v)$ and charge them to $g(v)$. Repeatedly doing this gives us constant-factor approximation algorithms.

2.2 Elementary Lemmas

Here we state and prove several small but fundamental lemmas that we will use in the rest of the paper. They apply to both the general case and the upward-looking case. These lemmas and corollaries can be used with left and right interchanged; this is stated explicitly for Corollary 1 as an example but is not stated for the others.

Lemma 1. (Order Claim) [5, 9] *For vertices a, b, c, d such that $a \leq b < c \leq d$, if a sees c and b sees d then a sees d.*

Proof. This becomes quite clear with the help of a diagram (see Figure 2.1). It is trivially true if $a = b$ or $c = d$; otherwise we know that $a < b < c < d$. In this case b cannot be above $\overline{ac}$ and c cannot be above $\overline{bd}$ (otherwise the fact that a sees c and b sees d would be violated). This means that the two line segments must cross; we call their intersection point p. Considering the triangle formed by a, p, and d, we note that no point on the terrain can be above the lower hull and $\overline{ad}$ is the upper hull. Therefore no point on the terrain can be above $\overline{ad}$. For the upward-looking case it is not difficult to see that if a sees c and b sees d then a must be below d so the lemma holds. $\square$

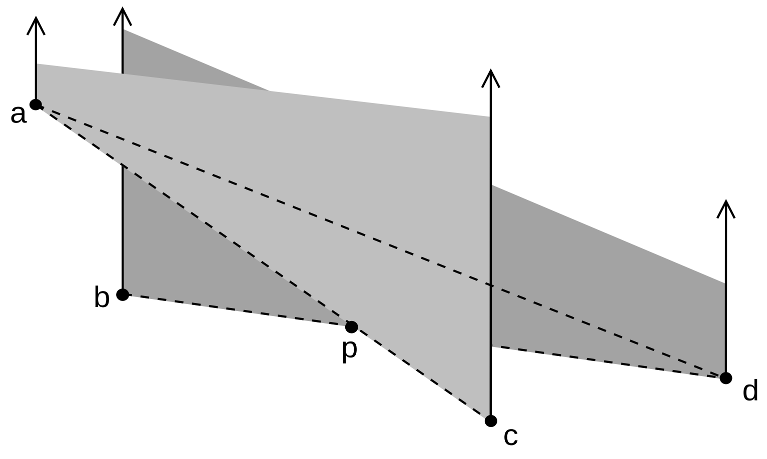

Figure 2.1: The shaded areas are terrain free and their union contains $\overline{ad}$.

Corollary 1. *For vertices v, u, x with $v \leq u < x$, if v and u can both be seen from $T_R(x)$ then $R(u) \leq R(v)$.*

Corollary 1. (Symmetric Version) *For vertices v, u, x with $x < u \leq v$, if v and u can both be seen from $T_L(x)$ then $L(v) \leq L(u)$.*

Lemma 2. *For an interval $[a, b]$ where a sees b, any guard in (a, b) is dominated with regard to $T_R(b)$ by a.*

Proof. Let v be a guard in (a, b) and let u be some vertex in $T_R(b)$ seen by u. If $u = b$ we know that a sees u. Otherwise the order claim, applied to a, v, b, u, states that a sees u. $\square$

Corollary 2. *For vertices x and y such that $x < y < R(x)$, we know that $R(y) \leq R(x)$.*

Chapter 3

Upward Looking Guards

3.1 Introduction to the Upward Looking Case

We consider a restricted version of the 1.5-dimensional terrain guarding problem in which guards cannot see below themselves. That is, a guard at x can see a point y if and only if the line segment $\overline{xy}$ does not pass under the terrain and y is not below x. With this restriction we refer to the problem as the *upward looking 1.5-dimensional terrain guarding problem.* We can define TG-VV-UP = TG-UP and TG-TT-UP, whose definitions should be obvious from Section 1.1. In Section 3.2 we give a 2-approximation algorithm for TG-UP. In Section 3.3 we show how the algorithm can, with minor modifications, be applied to TG-TT-UP to obtain the same approximation factor. In Section 3.4 we show that our algorithm runs in quadratic time.

The restriction that a guard cannot see points below it has several effects on the visibility graph of the terrain. First of all, since visibility between two vertices is no longer symmetric, the visibility graph is a directed graph. Secondly, if we assume that the vertices are in general position (specifically that no two are at the same height), the visibility graph is acyclic since a vertex on the terrain can only see vertices above it and can only be seen by vertices below it. With such an assumption the problem TG-UP is therefore equivalent to DOMINATING SET (see, *e.g.*, [20]) on a restricted class of directed acyclic graphs.

We refer to the DOMINATING SET problem on general DAGs as DOM-DAG. In Chapter 5 we show via a simple reduction from SET COVER that DOM-DAG is NP-hard. We also show that any polytime approximation algorithm for DOM-DAG must have at least a logarithmic approximation factor unless NP $\subseteq$ DTIME($n^{\log\log n}$). Since, with the assumption of general position, TG-UP is a restricted case of DOM-DAG, this inapproximability increases the significance of a constant factor approximation algorithm for TG-UP. We should note that our algorithm does not require vertices to be in general position.

3.2 The TG-UP Algorithm

Our algorithm starts with an empty set G of guards and repeatedly considers the lowest vertex p on the terrain that is not seen by G. We will place up to 3 guards to dominate $g(p)$ and charge them to $g(p)$. Our algorithm is based on the following lemma:

Lemma 3. *If p is the lowest unguarded vertex then $\{L(p), p, R(p)\}$ dominates any vertex that sees p.*

Proof. Any vertex that sees p must be in $[L(p), R(p)]$. Using Lemma 2 we can see that $\{L(p), p\}$ dominates any vertex in $[L(p), p]$ with regard to $T_R(p)$ and $\{p, R(p)\}$ dominates any vertex in $[p, R(p)]$ with regard to $T_L(p)$. p must be the highest point in $[L(p), R(p)]$, so there are no unguarded points in $[L(p), R(p)]$ other than p. What remains to be shown is that $\{L(p), p, R(p)\}$ dominates any vertex in $(L(p), p)$ with regard to $T_L(L(p))$ and dominates any vertex in $(p, R(p))$ with regard to $T_R(R(p))$.

Let v be a vertex in $(L(p), p)$. If there is an unguarded vertex u to the left of $L(p)$ that is seen by v then the order claim tells us the line segment between p and u is uninterrupted by the terrain. This means that if u does not see p then p must see u. u cannot see p because this would contradict the definition of $L(p)$, so p must see u. This proves our claim for the left side; the right side can be proven symmetrically. □

Our algorithm is very simple. Once we have found the lowest unguarded vertex p, we simply place guards at $L(p)$, p, and $R(p)$ and charge them to $g(p)$. Lemma 3 tells us that any vertex that sees p must be dominated by $\{L(p), p, R(p)\}$. These vertices may or may not be distinct so in actual fact we have placed 1, 2, or 3 guards.

At first it seems that this algorithm can have an approximation factor as high as 3, but with a bit more analysis we can prove an upper bound less than 2. Let P be the set containing the lowest unguarded point at each iteration of our algorithm (*i.e.* the points p for which we placed guards at $L(p)$, p, and $R(p)$). $|P| \leq |G_{OPT}|$. Let p be some vertex in P. We know that $L(p) \leq p \leq R(p)$, but we can be more specific. We consider three cases based on the vertices related to p:

1. $L(p) = p = R(p)$
2. $L(p) = p < R(p)$ or $L(p) < p = R(p)$
3. $L(p) < p < R(p)$.

Clearly our algorithm places 1 guard in the first case, 2 in the second, and 3 in the third. We know that $|G| \leq 3|P|$, but we can also say that if Case 1 occurs at least as often as Case 3 then $|G| \leq 2|P|$. Case 1 will occur if and only if p is a local minimum. Case 3 will occur if and only if p is an *intermediate maximum*, where we define an intermediate maximum as a local maximum that is neither the leftmost nor the rightmost vertex in the terrain. Any other situation falls into Case 2.

Now we show that Case 1 occurs more often than Case 3, bringing the approximation factor below 2. The number of local minima is one more than the number of intermediate maxima, since there is exactly one local minimum between any two intermediate maxima, as well as one to the left of the leftmost and one to the right of the rightmost. Also, every local minimum is in P since a local minimum can only be guarded by itself. Therefore Case 1 must occur strictly more often than Case 3, so our algorithm's approximation factor is below 2. It is not difficult to come up with sample terrains in which our algorithm achieves an approximation factor as bad as $2 - \Theta(1/n)$, and that is the upper bound.

Pseudocode for our algorithm is as follows:

Algorithm 1 TG-UP(T)

```
$G \leftarrow \emptyset$;
while unguarded points remain do
    let $p$ be the lowest unguarded vertex;
    add $p$, $L(p)$, and $R(p)$ to $G$;
end while
```

3.3 Modifications for TG-TT-UP

To make the TG-UP algorithm work for TG-TT-UP we need to consider guarding edges instead of guarding vertices. The following two observations are crucial. First of all, if p is the lowest unguarded point on the terrain, any guard that sees p must see every unguarded point on the same edge as p. Secondly, the lowest unguarded point on the terrain will always be the lowest point on some edge. These observations together mean that if a guard sees part of an edge but not all of it then, for our purposes, it may as well see none of that edge. We are therefore only concerned about whether a guard sees all of an edge or not.

Consider an edge (x, y) where, without loss of generality, x is below y. A point sees all of (x, y) if and only if it sees x and is not below the line passing through x and y. The leftmost point that sees this edge will either be $L(x)$ if it not below the line passing through x and y and will be x otherwise. The rightmost point that sees this edge will be x. It is therefore clear that there are no more than $2n$ potential guard locations that we need to consider.

3.4 Time Complexity

For TG-VV-UP we first compute the visibility graph of the terrain. As Ben-Moshe points out [3] this can be done in $O(n^2)$ time. From here on it is simple to show that the algorithm runs in quadratic time. Our algorithm goes through $O(n)$ iterations. In each iteration we find the lowest unguarded vertex p, find $L(p)$ and $R(p)$, place guards at these vertices, then mark all vertices seen by our new guards as guarded. All of the work in an iteration can clearly be done in linear time. Our algorithm for TG-UP therefore runs in $O(n^2)$ time.

For TG-TT-UP we do the same. The only difference is that the visibility graph will be a bipartite graph with points on the terrain seeing edges (see Section 3.3). The size of the graph will still be $O(n^2)$ and can be constructed in $O(n^2)$ time. Our algorithm therefore runs in $O(n^2)$ time for TG-TT-UP.

Chapter 4

The General Case

4.1 Introduction to the General Case

Our algorithm for the general case works by repeatedly finding an unguarded vertex u and a set S of up to 4 vertices such that S must dominate $g(u)$. By doing so, we achieve an approximation factor of 4. Our algorithm does not require any knowledge of previously placed guards other than which vertices are unguarded.

GuardRight is a recursive subroutine that does most of the algorithm's work. GuardRight takes two vertices x and c as parameters and it places guards appropriately until every vertex in $[x, R(x))$ is guarded. The preconditions are that x is unguarded, x is not on $CH(T)$, and every unguarded vertex in $[L(R(x)), x)$ is seen by c. We note that if x is not on $CH(T)$ then $x \neq R(x)$, which is a fact we will need later. The other significant consequences of x not being on the convex hull are that $L(R(x)) \leq L(x) < x$ and, due to Corollary 2, no vertex outside of $[L(R(x)), R(x)]$ can see an unguarded vertex y in $[x, R(x))$.

In the outermost loop of our algorithm we consider the two leftmost unguarded vertices p and q with $p < q$. In the case where p is not on $CH(T)$ we call GuardRight(p, p) and the preconditions are satisfied. If p is on $CH(T)$ but q is not we call GuardRight(q, p) and the preconditions are satisfied. If both p and q are on the convex hull we can simply place a guard at $R(p)$ and this will dominate $g(p)$. Clearly if there is only one unguarded vertex p remaining we can just place a guard on p to dominate $g(p)$.

We say that an interval I is *independent* if no unguarded interior vertex in I can be seen from outside I. For a call to GuardRight(x, c) we say that the interval $[L(R(x)), R(x)]$ is *pseudo-independent* since we could make it independent with the placement of a single guard (at c in this case). GuardRight will either find an unguarded vertex $u \in [x, R(x))$ for which $g(u)$ can be dominated by 4 guards or will find a pseudo-independent subinterval of $[x, R(x)]$ that it can recurse on.

4.2 Preliminaries

Here we introduce some lemmas that hold in the general case.

Lemma 4. (Lip Lemma) *For an interval $[a, b]$ where a sees b, if there are no unguarded vertices in (a, b) then $\{a, b\}$ dominates any guard in $[a, b]$.*

Proof. This follows easily from Lemma 2 since a sees b and b sees a. □

Lemma 5. *For a vertex x, any guard v in $T_L(x)$ is dominated with regard to $T_R(x)$ by a guard in $CH_L(x)$. One such dominating guard is the rightmost vertex in $T_L(v) \cap CH_L(x)$.*

Proof. Let u be the rightmost vertex in $T_L(v) \cap CH_L(x)$. If v is on $CH_L(x)$ then $v = u$ and the lemma clearly holds. Otherwise let w be the first vertex on $CH_L(x)$ to the right of u. Now u dominates v with regard to $T_R(x) \subseteq T_R(w)$ by Lemma 2. $\square$

Corollary 3. *For vertices x and v, if $L(v) \leq x \leq v$ then $L(v)$ is on $CH_L(x)$.*

At this point we introduce some new terminology and notation that depends on the parameters of GuardRight. It should be emphasized that this notation applies only to this particular call to GuardRight. We say that a *left vertex* is a vertex in $CH([L(R(x)), x]) - \{x\}$. A *right vertex* is a vertex in $[x, R(x)]$. An *open vertex* is an unguarded vertex in $[x, R(x))$ that can be seen by a left vertex. A *closed vertex* is an unguarded vertex in $[x, R(x))$ that cannot be seen by a left vertex. For an open vertex v we provide additional notation: $R'(v)$ is the rightmost left vertex that sees v and $L'(v)$ is the leftmost right vertex that sees v. $R'(v)$ and $L'(v)$ are undefined unless v is an open vertex.

Lemma 6. **(a)** *If v is a closed vertex then $L(R(x)) \leq x \leq L(v) \leq v \leq R(v) \leq R(x)$.* **(b)** *If v is an open vertex then $L(R(x)) \leq L(v) \leq R'(v) < x \leq L'(v) \leq v \leq R(v) \leq R(x)$.*

Proof. **(a)** $L(R(x)) \leq x$ since $R(x)$ sees x. $x \leq L(v)$ otherwise v would be an open vertex. $L(v) \leq v$ by definition. $R(v) \leq R(x)$ by Corollary 2. **(b)** $L(R(x)) \leq L(v)$ by Corollary 1 if $x < v$ and by the Order Claim otherwise. $L(v) \leq R'(v) < x \leq L'(v) \leq v$ by definition. $R(v) \leq R(x)$ by Corollary 2. $\square$

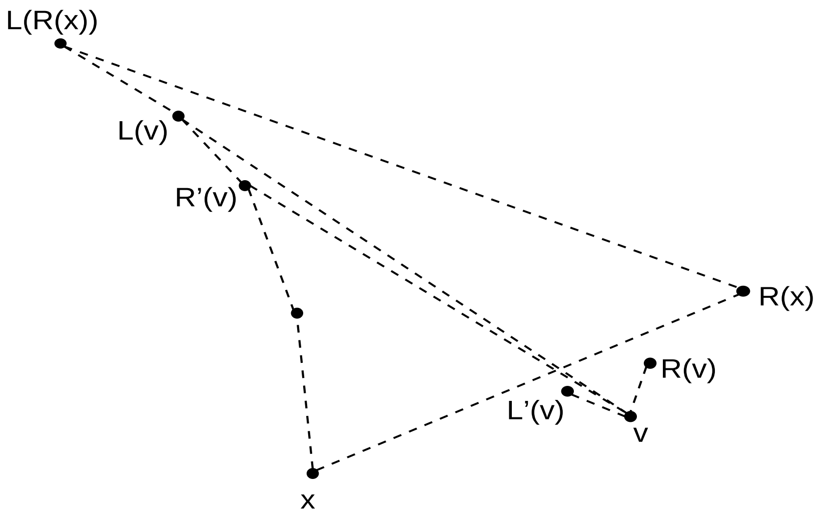

Figure 4.1: The order of some vertices related to an open vertex v.

Lemma 7. *For an open vertex v, $L'(v)$ sees $R(v)$.*

Proof. If $v = x$ this is clearly true since $x = L'(x)$. Otherwise, it is easy to see that this is true as long as v is not above the line passing through $L'(v)$ and $R(v)$. $L'(v)$ cannot be below the line passing through x and v otherwise v would be seen by a vertex in $[x, L'(v))$ which contradicts the definition of $L'(v)$. Similarly, $R(v)$ cannot be below the line passing through v and $R(v)$. It should now be clear that v is not above the line passing through $L'(v)$ and $R(v)$ (see Figure 4.2). The rest follows trivially. □

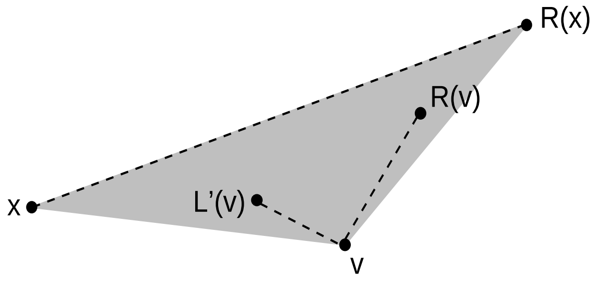

Figure 4.2: $L'(v)$ and $R(v)$ must be in the shaded region.

Lemma 8. *For an open vertex v the set of left vertices that see v is contiguous,* i.e. *every left vertex in $[L(v), R'(v)]$ sees v.*

Proof. Consider $CH([L(v), R'(v)])$. This is a subset of $CH([L(R(x)), x]) - \{x\}$ since $L(v)$ and $R'(v)$ are both on $CH([L(R(x)), x])$. So we can see that $CH([L(v), R'(v)])$ is a set of left vertices and we know that no left vertex not in the set can see v. Now we will show that if w is a vertex in the set, $w \neq L(v)$, w sees v, and u is the first vertex in the set to the left of w, then u also sees v. Consider the line passing through v and w. u must be above this line since $w \neq L(v)$. Since u and w are consecutive points on the convex hull, w sees u. Now we can see that $\overline{uw}$ and $\overline{wv}$ are line segments that do not interfere with the terrain, so $\overline{uv}$ cannot interfere with the terrain since it is above $\overline{uw}$ and $\overline{wv}$. Therefore u sees v. It is easy to extend this into an induction proof for the lemma. □

Lemma 9. *For every vertex v in $[L(R(x)), x)$ there is a left vertex u such that $\{c, u\}$ dominates v. This guard u is the rightmost vertex in $T_L(v) \cap CH_L(x)$.*

Proof. By Lemma 5 we know that u dominates v with regard to $T_R(x)$. If $u \neq v$ then v cannot see any vertex to the left of u so u dominates v with regard to $T_L(u)$. c can see every unguarded vertex in $[L(R(x)), x)$. Since $L(R(x)) \leq u$, we can see that $T_L(u) \cup [L(R(x)), x) \cup T_R(x) = T$. Therefore $\{c, u\}$ dominates v. □

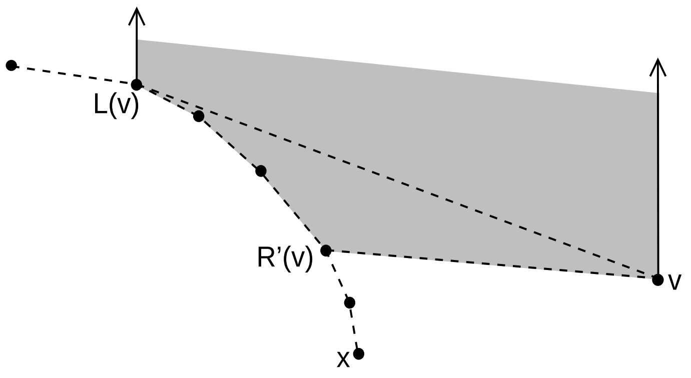

Figure 4.3: The shaded region is terrain free, so every left vertex in $[L(v), R'(v)]$ must see v.

Lemma 9 tells us that, as long as we place a guard at c when we place other guards, we needn't place any guard in $[L(R(x)), x)$ unless it is on a left vertex.

4.3 Finding a Good Left Vertex

The first thing we note is that there must be at least one open vertex in $[x, R(x))$, namely x. There may or may not be a closed vertex in $[x, R(x))$. We define b as the leftmost left vertex such that some open vertex v is seen by b but not by any left vertex to the right of b. In other words, b is the minimum $R'(v)$ over all open vertices v. We define d as the leftmost open vertex for which $R'(d) = b$.

Lemma 10. *Every open vertex in $(L'(d), R(x))$ is seen by $L(d)$.*

Proof. If $d = x$ then the proof follows easily from the symmetric version of Corollary 1, so we will assume this is not the case. First we will prove that there are no open vertices in $(L'(d), d)$. Assume for the sake of contradiction that there is an open vertex v in $(L'(d), d)$. We can apply the order claim to $R'(v), L'(d), v, d$ to see that $R'(v)$ sees d. This tells us that $R'(v) \leq R'(d)$, which violates the definition of d, so there cannot be any such vertex v. Now we show that $L(d)$ sees every open vertex in $(d, R(x))$. Let u be an open vertex in $(d, R(x))$. We have $L(u) < d < u$ so by the symmetric version of Corollary 1 we know that $L(u) \leq L(d)$. By the definition of d we know that $R'(d) \leq R'(u)$. Therefore $L(d) \in [L(u), R'(u)]$, so by Lemma 8 we know that $L(d)$ sees u. □

Lemma 11. *Any guard in $[L(R(x)), x)$ that sees d is dominated by $\{L(d), c\}$.*

Proof. Let v be a guard in $[L(R(x)), x)$ that sees d. Since c sees every unguarded vertex in $[L(R(x)), x)$ it suffices to prove that $L(d)$ dominates v with regard to

$T_R(x)$. $L(d) \leq v$, so by Lemma 2 $L(d)$ dominates v with regard to $T_R(d)$. Now we show that no left vertex that sees d can see any open vertex to the left of d. It follows from the definition of $b = R'(d)$ and from Lemma 8 that any open vertex seen from the left of $R'(d)$ must be seen by $R'(d)$. However, d is the leftmost open vertex seen by $R'(d)$, so no open vertex to the left of d can be seen by $R'(d)$. In other words, no open vertex to the left of d can be seen by a left vertex that sees d. This, along with Lemma 5, tells us that v cannot see any unguarded vertices in $[x, d)$. Since v cannot see any closed vertices at all, this means that $L(d)$ dominates v with regard to $T_R(x)$. v cannot see anything left of $L(R(x))$ except possibly if $v = L(d)$, so $\{L(d), c\}$ dominates v over the entire terrain. □

Recall that, while searching for a suitable vertex u for which we can dominate $g(u)$ with 4 guards, either we find one right away or we find some pseudo-independent pocket (*i.e.* a subinterval of $[x, R(x))$) that we can recurse upon.

4.4 The Terminal Case

We first consider the case where there are no closed vertices in $(L'(d), R(d))$. We place guards at $\{c, L(d), L'(d), R(d)\}$ and claim that these guards dominate any guard that sees d. Lemma 10 tells us that every open vertex in $(L'(d), R(d))$ is seen by $L(d)$, and since there are no closed vertices in $(L'(d), R(d))$ there are no longer any unguarded vertices in $(L'(d), R(d))$. $L'(d)$ sees $R(d)$ by Lemma 7. Therefore, by Corollary 4, any guard in $[L'(d), R(d)]$ is dominated by $\{L(d), L'(d), R(d), c\}$. By Lemma 11 any guard in $[L(R(x)), x]$ that sees d is dominated by $\{L(d), L'(d), R(d), c\}$. Any guard that sees d must either be in $[L(R(x)), x]$ or in $[L'(d), R(d)]$, so $\{L(d), L'(d), R(d), c\}$ dominates any guard that can see d.

4.5 The Recursive Case

If there are closed vertices in $(L'(d), R(d))$ our job is slightly more complicated and requires recursion (this is where we find our 'independent pocket'). We require another subroutine, GuardLeft, that is like a mirror image of GuardRight. For GuardLeft(x', c') the precondition on c' is flipped horizontally: every unguarded vertex in $(x', R(L(x'))]$ must be seen by c'.

Let y be the rightmost closed vertex (note that y is not necessarily in the interval $(L'(d), R(d))$, but it must be in $(L'(d), R(x))$). We will show that the preconditions are satisfied if we call GuardLeft$(y, L(d))$. By Corollary 2 $R(L(y)) \leq R(x)$ and by the definition of y any unguarded vertex in $(y, R(x))$ is an open vertex. Therefore by Lemma 10 every unguarded vertex in $(y, R(L(y)))$ is seen by $L(d)$. If $R(L(y)) < R(x)$ then either $R(L(y))$ is already guarded or it is an open vertex and is seen by $L(d)$. If $R(L(y)) = R(x)$ then $L(d)$ sees

$R(L(y))$ since every vertex in $CH([L(R(x)), x])$ sees $R(x)$. Therefore every unguarded vertex in $(y, R(L(y))]$ is seen by $L(d)$. We know y is unguarded and $y \in (x, R(x))$ (and is therefore not on $CH(T)$), so the preconditions are satisfied.

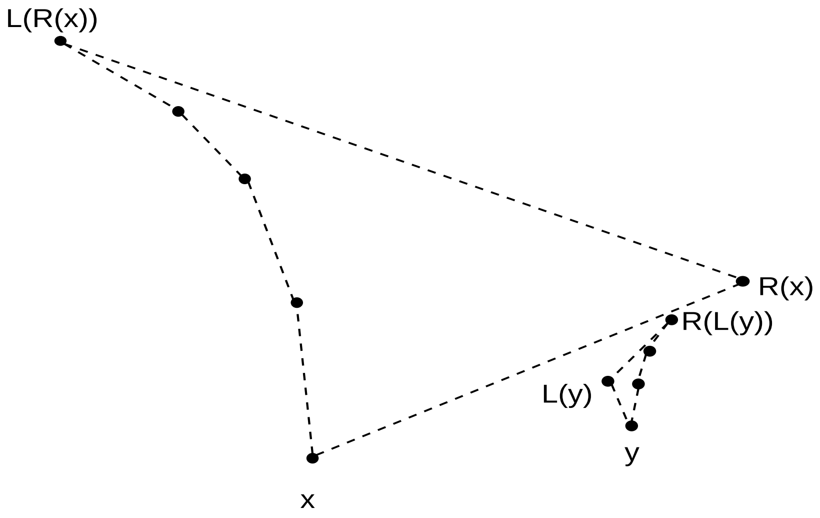

Figure 4.4: The nested interval $[L(y), R(L(y))]$ can be handled independently with the help of a dominant outside vertex.

In this way we can do a sort of recursive zig-zagging where each call to GuardRight will spawn a call to GuardLeft and each call to GuardLeft will spawn a call to GuardRight. It is not difficult to see that eventually, after at most a linear number of these zig-zagging steps, we will find a suitable u. At this point we can simply place our 4 guards and, if we need to, start a brand new call to GuardRight. Pseudocode for GuardRight and the main algorithm Guard that calls it is given as Algorithm 3 and Algorithm 2 respectively.

4.6 Modifications for TG-TT

No real modifications need to be made to apply our TG algorithm to TG-TT. However, we need to keep track of more information if we want our algorithm to run as efficiently as possible. This is discussed in Section 4.7.

4.7 Time Complexity

It is clear that at most $O(n)$ initial calls to GuardRight can be made. For TG-VV it is also easy to see that an initial call to GuardRight will result in a

Algorithm 2 GUARD(T)

```
while $T$ is not completely guarded do
    let $p$ be the leftmost unguarded vertex;
    if $p$ is the only unguarded vertex then
        place a guard at $p$;
        charge the guard to $g(p)$;
    else if $p$ is not on $CH(T)$ then
        GUARDRIGHT$(p, p)$;
    else
        let $q$ be the second leftmost unguarded vertex;
        if $q$ is not on $CH(T)$ then
            GUARDRIGHT$(q, p)$;
        else
            place a guard at $R(p)$;
            charge the guard to $g(p)$;
        end if
    end if
end while
```

Algorithm 3 GUARDRIGHT(x, c)

```
          • $c$ sees every unguarded vertex in $(L(R(x)), x)$
Require:  • $x$ is unguarded
          • $x \notin CH(T)$
    $b \leftarrow \min\{R'(v) : v \text{ is open}\}$;
    $d \leftarrow \min\{v : R'(v) = b\}$;
    if there is a closed vertex in $(L'(d), R(d))$ then
        let $y$ be the rightmost such closed vertex;
        GUARDLEFT$(y, L(d))$;
    else
        place guards at $c$, $L(d)$, $L'(d)$, and $R(d)$;
        charge the guards to $g(d)$;
    end if
```

number of guards being placed in $O(n^2)$ time. We can therefore give an upper bound of $O(n^3)$ for the running time of TG-VV.

If we want TG-VV to be more efficient, we can make GuardRight(x, c) continue placing guards until $[x, R(x))$ has been completely guarded. This changes things slightly; on a given iteration, x is not necessarily unguarded so there is not necessarily an open vertex. If there is no open vertex, however, we can just recurse immediately by calling GuardRight(y, c) so this is not a problem. To increase efficiency, we can sort the open vertices v by $R'(v)$ (breaking ties using the x-coordinates of open vertices) to find an appropriate b and d faster in each iteration.

A call to GuardRight(x, c), ignoring all recursive calls that it spawns, can now run in $O(n + m \log m)$ time, where m is the number of open vertices in $[x, R(x))$. It is easy to see that the 'n' terms, added up over the entire course of the algorithm, will cost $O(n^2)$ time since there will be at most $O(n)$ calls to GuardRight. Any vertex will be an open vertex for at most one call to GuardRight, so the sum of all $m \log m$ factors encountered will actually be bounded by $O(n \log n)$. All other overhead incurred by the algorithm can be dealt with in $O(n^2)$ time, so the running time of TG-VV is bounded by $O(n^2)$. Pseudocode for the efficient but less elegant version of GuardRight is given as Algorithm 4.

When dealing with TG-TT the only real problem is finding b and d at each iteration of a call to GuardRight. Instead of open vertices and closed vertices, we consider open edge sections and closed edge sections. It is not difficult to see that for each edge of the terrain, at most one contiguous section will be open and at most one will be closed. From left to right on an edge, we can have a guarded section, a closed section, an open section, and another guarded section, though not all of these sections will necessarily exist. For an open section, the leftmost point will have the leftmost R'.

$L(p)$ is always a vertex regardless of where p is. If we keep track of the transition points for the function $L(p)$ (there are only $O(n)$ of them [11]) then we can know where open sections end and closed sections begin. For every edge, our algorithm also keeps track of where the unguarded section starts and ends (it must be contiguous). After placing a guard, updating the unguarded section on every edge can be done quite easily in linear time. Assume we have just placed a guard at g. To the left of g call the first vertex x_1 and consider the edge e_1 whose left endpoint is x_1. Mark down that every point on e_1 is guarded. Now, moving left from x_1, find the first vertex above the line going through g and x_1; call this x_2, define e_2 appropriately and mark down that every point on e_2 above the line going through g and x_1 is guarded. It is easy to see how we can proceed to update the unguarded section of each edge in linear time. Since

Algorithm 4 GuardRight(x, c) (Efficient Version)

```
Require: • $c$ sees every unguarded vertex in $(L(R(x)), x)$
         • $x \notin CH(T)$
  while there are unguarded vertices in $[x, R(x))$ do
    if there is a closed vertex then
      let $y$ be the rightmost closed vertex;
      if there is an open vertex in $(y, R(x))$ then
        $b \leftarrow \min\{R'(v) : v \text{ is open}\}$;
        $d \leftarrow \min\{v : R'(v) = b\}$;
        if $y$ is in $(L'(d), R(x))$ then
          GuardLeft($y, L(d)$);
        else
          place guards at $c$, $L(d)$, $L'(d)$, and $R(d)$;
        end if
      else
        GuardLeft($y, y$);
      end if
    else
      $b \leftarrow \min\{R'(v) : v \text{ is open}\}$;
      $d \leftarrow \min\{v : R'(v) = b\}$;
      place guards at $c$, $L(d)$, $L'(d)$, and $R(d)$;
    end if
  end while
```

we place $O(n)$ guards the total cost of updating guarded edge sections of the terrain is $O(n^2)$.

If we do all of the aforementioned maintenance, we will only need to consider the leftmost point in each open section when looking for b and d. Therefore we do not need to worry about asymptotically more points in TG-TT than in TG-VV. The running time therefore remains $O(n^2)$.

Chapter 5

Domination of Directed Acyclic Graphs

5.1 Motivation

Dominating Set is a well-known NP-complete problem for general undirected graphs [20]. Every undirected graph can be represented as a directed graph, so the problem is NP-complete for general directed graphs as well. For undirected acyclic graphs, *i.e.* trees, there is a simple linear-time algorithm for Dominating Set [12]. In this section we show that the Dominating Set problem on directed acyclic graphs (DAGs), which we will abbreviate as Dom-DAG, is NP-complete and cannot be efficiently approximated within a sub-logarithmic factor.

While the general 1.5D terrain guarding problem is a restriction of Dominating Set on general graphs, the upward-looking case is a restriction of Dom-DAG (assuming the vertices are in general position). It is therefore natural to consider the complexity of Dom-DAG before trying to solve TG-Up. The hardness and inapproximability of Dom-DAG support the relevance of a 2-approximation algorithm for TG-Up.

5.2 NP-Hardness and Inapproximability

Lemma 12. Dominating Set *on directed acyclic graphs is NP-complete. Furthermore, if k is the size of the minimum dominating set, it cannot be approximated to within a factor of $o(\log k)$ in polynomial time unless* $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{\log\log n})$.

Proof. The proof follows from a fairly trivial gap-preserving reduction from Set Cover.

Recall that an instance I of Set Cover is a set S along with a collection C of subsets of S, and the problem is to find the smallest subset C' of C such that every element of S is in at least one of the subsets in C'. Our reduction creates a DAG G with $|S| + |C| + 1$ vertices that is an instance of Dom-DAG. For each element in S we create a vertex with no outgoing edges. We will call these *element vertices*. For each $X \in C$ (recall that $X \subset S$), we create a vertex with an outgoing edge to each element vertex that represents an element in X. We will call these *set vertices*. We then create one final vertex, the source, with no incoming edges and with an outgoing edge to every set vertex.

There are several things to note about G. First of all, the source is in every dominating set of G since it has no incoming edges. Secondly, for any dominating set A of G that contains an element vertex, a dominating set B of equal or lesser cardinality can be constructed trivially by replacing each element vertex v in A with one of the set vertices that has an outgoing edge to v.

Finding a minimum dominating set of G that contains the source and does not contain any element vertices is therefore no harder than finding any minimum dominating set of G. So a minimum dominating set of G consists of the source, plus the minimum subset of set nodes required to cover all of the element nodes. It should be clear that the minimum dominating set for G has size $k+1$ if and only if the minimum set cover of I contains exactly k elements of C.

Set Cover is NP-complete and cannot be approximated to within a factor of $o(\log |S|)$ in polynomial time unless NP $\subseteq$ DTIME($n^{\log \log n}$)[19]. Also, the size of the minimum dominating set for the graph obtained by our reduction is at most $|S|$. Since our reduction is gap-preserving and is clearly polynomial, Dom-DAG must be NP-complete and cannot be approximated to within a factor of $o(\log |k|)$ in polynomial time unless NP $\subseteq$ DTIME($n^{\log \log n}$). $\square$

Chapter 6

Future Work

6.1 NP-Completeness

The most pressing and obvious question regarding the 1.5D terrain guarding problem is whether or not it is NP-complete. All of our attempts at an NP-hardness proof have been stymied by the Order Claim. On the other hand, attempts at designing an exact polynomial-time algorithm have also been unsuccessful.

If the problem is not NP-hard, we would be interested in a polynomial-time algorithm. If the problem is NP-hard, we would be interested in approximability thresholds, *e.g.* whether it is APX-complete or admits a PTAS or even an FPTAS.

6.2 Characterization of Terrain Graphs

We define an *ordered graph* as a graph G in which the vertices $v_1 < v_2 < \ldots < v_n$ have an ordering and there is an edge between v_i and v_{i+1} for every $i \in [1, n-1]$. We define a *terrain graph* as a graph G that is the visibility graph of a 1.5D terrain (specifically, an instance of TG-VV). It is easy to see that every terrain graph is an ordered graph, but not every ordered graph is a terrain graph.

The Order Claim seems to capture much of the restriction of terrain graphs. However, it is not the case that every ordered graph obeying the Order Claim is a terrain graph (consider, for example, a cycle of 4 or more vertices). What additional restrictions must we place on ordered graphs to ensure they are terrain graphs? Consider the following lemma for 1.5D terrains:

Lemma 13. (Midpoint Claim) *For any vertices v_i and v_j such that $j > i + 1$ and v_i sees v_j, there is some vertex in (v_i, v_j) that is seen by both v_i and v_j.*

Proof. Let x be the vertex in (v_i, v_j) closest to the line passing through v_i and v_j (note that all vertices in this interval are below the line). It is easy to see that v_i and v_j both see x. □

We would like to know whether the Order Claim and the Midpoint Claim together are restrictive enough that every ordered graph obeying both claims is a terrain graph. If this is not the case, there may be some other useful property of terrain graphs that we could exploit in our search for a polynomial-time terrain guarding algorithm.

6.3 Reductions Between Restricted Visibility Problems

Terrain guarding is easy when guards can only see to the right. It also seems that many troublesome scenarios are eliminated when guards can only see upwards. When guards can only look downwards, however, the problem seems as complex as the unrestricted case if not more so, though it seems that our algorithm for the general case can solve this variant.

We would be interested in reductions between upward-looking, downward-looking, and general terrain guarding. For example, would a polynomial-time algorithm for the downward looking case imply that the general case is in P? Would an NP-hardness proof for the upward-looking case imply that the general case is NP-hard? In particular it seems that the upward-looking case should be at least as hard as the general case. However, our efforts to provide reductions of this kind have been unsuccessful.

Bibliography

[1] P. Agarwal, S. Bereg, O. Daescu, H. Kaplan, S. Ntafos, and B. Zhu. Guarding a terrain by two watchtowers. In *Symposium on Computational Geometry*, pages 346–355, 2005.

[2] A. Aggarwal. *The art gallery problem: Its variations, applications, and algorithmic aspects.* PhD thesis, Johns Hopkins University, 1984.

[3] B. Ben-Moshe. *Geometric Facility Location Optimization.* PhD thesis, Ben-Gurion University, 2004.

[4] B. Ben-Moshe. Personal communication, 2005.

[5] B. Ben-Moshe, M. Katz, and J. Mitchell. A constant-factor approximation algorithm for optimal terrain guarding. In *Symposium on Discrete Algorithms*, 2005.

[6] P. Bose, D. Kirkpatrick, and Z. Li. Efficient algorithms for guarding or illuminating the surface of a polyhedral terrain. In *Proceedings of the 8th Canadian Conference on Computational Geometry*, pages 217–222, 1996.

[7] P. Bose, T. Shermer, G. Toussaint, and B. Zhu. Guarding polyhedral terrains. *Computational Geometry: Theory and Applications*, 7, 1997.

[8] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete & Computational Geometry*, 14, 1995.

[9] D. Z. Chen, V. Estivill-Castro, and J. Urrutia. Optimal guarding of polygons and monotone chains (extended abstract), 1996.

[10] V. Chvátal. A combinatorial theorem in plane geometry. *J. Comb. Theory Series B*, 18:39–41, 1975.

[11] K. L. Clarkson and K. Varadarajan. Improved approximation algorithms for geometric set cover. In *Symposium on Computational Geometry*, 2005.

[12] E. Cockayne, S. Goodman, and S. Hedetniemi. A linear algorithm for the domination number of a tree. *Information Processing Letters*, 4(2):41–44, November 1975.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2001.

[14] S. Eidenbenz. *(In-)Approximability of Visibility Problems on Polygons and Terrains*. PhD thesis, ETH Zurich, 2000.

[15] S. Eidenbenz. Approximation algorithms for terrain guarding. *Information Processing Letters*, 82(2):99–105, April 2002.

[16] S. Eidenbenz, C. Stamm, and P. Widmayer. Positioning guards at fixed height above a terrain — an optimum inapproximability result. *Lecture Notes in Computer Science*, 1461, 1998.

[17] S. Eidenbenz, C. Stamm, and P. Widmayer. Inapproximability results for guarding polygons and terrains. *Algorithmica*, 31(1):79–113, 2001.

[18] H. Everett and E. Rivera-Campo. Edge guarding polyhedral terrains. *Computational Geometry: Theory and Applications*, 7, 1997.

[19] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, July 1998.

[20] M. Garey and D. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., 1979.

[21] D. Haussler and E. Welzl. ϵ-Nets and Simplex Range Queries. *Discrete & Computational Geometry*, 2:127–151, 1987.

[22] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences (JCSS)*, 9:256–278, 1974.

[23] A. A. Kooshesh and B. M. E. Moret. Three-coloring the vertices of a triangulated simple polygon. *Pattern Recognition*, 25, 1992.

[24] D. T. Lee and A. K. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32:276–282, 1986.

[25] M. Sharir and P. K. Agarwal. Davenport-Schinzel sequences and their geometric applications. Technical report, 1995.