Terrain Guarding is NP-Hard

James King* Erik Krohn†‡

October 20, 2009

Abstract

A set G of points on a 1.5-dimensional terrain, also known as an x-monotone polygonal chain, is said to guard the terrain if every point on the terrain is seen by a point in G. Two points on the terrain see each other if and only if the line segment between them is never strictly below the terrain. The minimum terrain guarding problem asks for a minimum guarding set for the given input terrain. Using a reduction from PLANAR 3-SAT we prove that the decision version of this problem is NP-hard. This solves a significant open problem and complements recent positive approximability results for the optimization problem.

1 Introduction

An instance of the *terrain guarding problem* contains a terrain T that is an x-monotone polygonal chain. An x-monotone chain in $\Re^2$ is a chain that intersects any vertical line at most once. The terrain is given by its set of vertices $P = \{v_1, v_2, ..., v_n\}$, where $v_i = (x_i, y_i)$. The vertices are ordered such that $x_i < x_{i+1}$. There is an edge connecting each (v_i, v_{i+1}) pair where $i = 1, 2, ..., n-1$. We say a point p on the terrain *sees* another point q on the terrain if the line segment $\overline{pq}$ is never strictly below the terrain T.

A set G of points on the terrain is called a *guarding set* if every point on the terrain is seen by some point in G. The optimization version of the terrain guarding problem is the problem of finding a minimum guarding set for a given terrain. There are two standard versions of the terrain guarding problem: a discrete version and a continuous version. The discrete version allows us to place guards only at the vertices of the terrain. The continuous version, which we have defined above, allows guards to be placed anywhere on the terrain. In other versions a subset of points on the terrain to guard is given with the input.

Motivation for guarding terrains comes from scenarios that include covering a road with street lights or security cameras. Other applications include finding a configuration for line-of-sight transmission networks for radio broadcasting, cellular telephony and other communication technologies [1].

The complexity of terrain guarding has been an open problem of interest since 1995, when an NP-completeness proof was proposed but never completed by Chen *et al.* [2]. With the problem's hardness strongly suspected but not known, a series of approximation algorithms have been developed over the last decade. The first constant factor approximation for the terrain guarding problem was shown by Ben-Moshe *et al.* in [1]. Clarkson and Varadarajan also give a constant factor approximation in [3]. A 4-approximation was proposed by King in [11] but further analysis increased the approximation factor to 5. A 4-approximation was given by Elbassioni *et al.* in [7]. Recently a PTAS was given by Gibson *et al.* in [9]. With the knowledge that the problem is not APX-complete, it is of even greater interest whether or not it is NP-complete, and this has been reiterated with each approximation algorithm developed.

The terrain guarding problem is closely related to the *art gallery problem* that involves guarding the interior of a polygon. The basic version of the art gallery problem is that of *vertex guarding* a simple polygon, where we are given a simple polygon and we wish to find the smallest subset of the vertices that see the entire polygon. The *point guarding* version allows guards to be placed anywhere inside the polygon.

The art gallery problem was shown to be NP-complete by Lee and Lin in [13]. Along with being NP-complete, the art gallery problem was shown to be APX-hard in [6]. This means that there exists a constant $\epsilon > 0$ such that no polynomial time algorithm can guarantee an approximation ratio of $1 + \epsilon$ unless $P = NP$. Ghosh provides a $O(\log n)$-approximation for the problem of vertex guarding an n-vertex simple polygon [8]. The point guarding problem seems to be much harder than the vertex guarding problem and precious little is known about it [5]. A restricted version of the point guarding

*School of Computer Science, McGill University, jking@cs.mcgill.ca

†Department of Computer Science, University of Iowa, eakrohn@cs.uiowa.edu

‡Corresponding author

problem where the polygon is x-monotone has been shown to have an $O(1)$-approximation by Nilsson in [17]. Based on his result Nilsson also provides a $O(OPT^2)$ approximation for rectilinear polygons.

Straightforward attempts to show NP-hardness for the terrain guarding problem run up against the large amount of restriction in the complexity of terrains. By far the most significant restriction is given by the following claim first noted by Ben-Moshe *et al.* [1]:

CLAIM 1. (ORDER CLAIM) *Let a, b, c, d be four points on the terrain in increasing order of x-coordinates. If a sees c and b sees d, then a sees d.*

The order claim is crucially exploited by all approximation algorithms for the problem. In this paper we develop a construction that overcomes the order claim obstacle and shows that the terrain guarding problem is NP-hard. Therefore, an exact polynomial time algorithm is not possible unless P = NP. The NP-hardness result is shown for the standard discrete and continuous variants of the problem.

According to Demaine and O'Rourke [4], the complexity of the terrain guarding problem was posed by Ben-Moshe. We quote from [4]:

> What is the complexity of computing the guard set of minimum size for a given x-monotone chain in the plane? According to the poser, "most tenured professors think the problem is NP-hard." This problem in fact goes back to 1995, when Chen *et al.* [2] claimed an NP-hardness result, but "the proof, whose details were omitted, was never completed successfully" [11].

Main result This paper contains a single result that resolves a long-standing open problem. Here we state the result and sketch the proof; the rest of the paper is dedicated to the full proof.

THEOREM 1. *Minimum terrain guarding is NP-hard.*

Proof. Let $\Phi = (X, C)$ be a Boolean formula in 3-CNF with $|X| = n$ and $|C| = m$. Specifically, we require that Φ is a planar formula in 3-CNF (see Definition 1). In our reduction we construct in polynomial time a terrain T_Φ that can be guarded by $f(\Phi)$ guards if and only if Φ is satisfiable. Here f is a function mapping planar 3-CNF formulae to the natural numbers, such that $f(\Phi)$ is polynomial in n and is computable in time polynomial in n.

T_Φ is built in several steps. First, we build a path representation of Φ, denoted P_Φ, in which each node stores a list of variables. The relationship between variable lists in adjacent nodes is strictly defined. Some nodes are specially marked as clause nodes or deletion nodes. This path representation, along with the PLANAR 3-SAT problem, is discussed in Section 2.

From the path representation P_Φ we then construct T_Φ. As this is an intricate process we separate the construction into two parts. In Section 3 we explain how to construct a terrain for n variables such that any minimum guarding set corresponds to a consistent truth assignment of the variables. In Section 4 we explain how additional gadgets are incorporated into such a terrain to construct T_Φ such that any guarding set of size $f(\Phi)$ corresponds to a consistent truth assignment satisfying Φ.

2 Path representations for PLANAR 3-SAT

In this section we introduce the PLANAR 3-SAT problem. We then describe how to express an instance of this problem in a format that lends itself naturally to embedding in a terrain using our truth assignment propagation framework from Section 3. The gadgets used for the embedding process are described in Section 4.

Defining PLANAR 3-SAT Many variants of 3-SAT are NP-hard; reductions from restricted variants are sometimes far simpler than reductions from general 3-SAT. In particular, PLANAR 3-SAT is often used to prove NP-hardness of geometric problems (see, *e.g.*, [16]). We adapt the following definition from Mulzer and Rote [16].

DEFINITION 1. (PLANAR 3-SAT) *Let Φ be a Boolean formula in 3-CNF. The **formula graph** of Φ, $G(\Phi)$, has one variable-vertex v_x for each variable x and one clause-vertex v_C for each clause C. The variable-vertices v_x are connected by edges to form a **variable cycle**, and for each clause-vertex v_C an edge (v_C, v_x) is added if C contains either literal x or $\overline{x}$. We say Φ is **planar** iff $G(\Phi)$ is planar. The PLANAR 3-SAT problem is equivalent to the 3-SAT problem restricted to planar formulae.*

THEOREM 2.1. (LICHTENSTEIN [14] THEOREM 2) *PLANAR 3-SAT is NP-complete.*

The variable cycle divides the plane into two regions, the interior and exterior of the cycle. Let $\mathcal{C}_-$ (resp. $\mathcal{C}_+$) be the set of clauses on the interior (resp. exterior) of the variable cycle. Let n be the number of variables. The most convenient way for us to now visualize Φ is that given by Knuth and Raghunathan[1] [12]

[1] The layout described by Knuth and Raghunathan actually has the variables on a horizontal line, but having them on a vertical

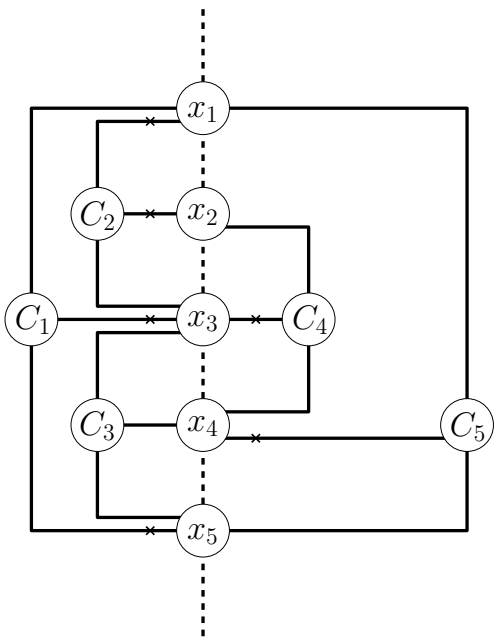

Figure 1: An instance Φ with three-legged clauses laid out to the left and right of the variables. Crosses on the lines indicate negations. For example, $C_1 = (x_1 \vee \overline{x_3} \vee \overline{x_5})$ and $C_4 = (x_2 \vee \overline{x_3} \vee x_4)$.

in which the variables $x_1, \ldots, x_n$ are laid out from top to bottom on a vertical line with three-legged clauses laid out to the left and right of this line. The edges are rectilinear, the clauses from $\mathcal{C}_-$ all lie to the left of the variables, and the clauses from $\mathcal{C}_+$ all lie to the right of the variables (see Figure 1).

Motivation for using PLANAR 3-SAT The only way we have been able to propagate a truth assignment around a terrain is in a linear 'highway', with each variable living in a 'lane' of the highway. We have been unable to reorder the variables in this highway, and we have only developed very restricted clause gadgets. Because of this, difficulties arose because each 3-CNF clause gadget would act like a 'roadblock' for the middle of the three variables involved, after which we could not continue to propagate that variable's truth assignment. Our solution is to arrange the variables and clauses in a way that ensures that each variable is the middle variable in at most two clauses, and that the variable will only be used in the length of the highway that is between these two clauses.

If, for some $i \in [2, n-1]$, x_i does not appear as the middle variable in a clause in $\mathcal{C}_-$, we add a *deletion node* to $G(\Phi)$ that is adjacent only to v_{x_i} and lies on the left side of the variable line. We do the same for $\mathcal{C}_+$ on the right side of the variable line. An example can be seen in Figure 2. This deletion node will be used in the following description of a *removal ordering*.

A removal ordering for Φ Considering Φ as laid out in Figure 1, it is not difficult to see that the clauses can be removed in an order such that a clause being removed has nothing 'between its legs'. We call such an ordering a *removal ordering* and order the sets $\mathcal{C}_-$ and $\mathcal{C}_+$ separately. Without loss of generality we can assume that x_1 and x_n are used in two common clauses, one in $\mathcal{C}_-$ and one in $\mathcal{C}_+$. If this is not the case, we can replace Φ with a new formula Φ' by adding a variable x_{n+1}, a clause $(x_1 \vee x_n \vee x_{n+1})$ in $\mathcal{C}_-$, and a clause $(\overline{x_1} \vee x_n \vee x_{n+1})$ in $\mathcal{C}_+$. An assignment of TRUE to x_{n+1} will satisfy both new clauses without affecting any other clauses, so $\Phi' \Leftrightarrow \Phi$ and the size of Φ' is linear in the size of Φ. Therefore it is safe to assume that x_1 and x_n are used in two common clauses and this additional clause is unnecessary.

We describe the removal ordering for $\mathcal{C}_-$. There is an associated list of variables from which one variable will be removed at each step. At the beginning of the process the variable list contains all variables. At each step we can remove:

- a clause with nothing between its legs, along with the clause's middle variable,
- a deletion node whose associated variable is not used in any remaining clauses in $\mathcal{C}_-$, along with its associated variable.

To avoid ambiguity, we always perform the action that removes the variable with the lowest index. At the end of the process the variable list contains only x_1 and x_n, and no clauses remain.

The key property of such a removal ordering is that, whenever a clause is removed, it involves three consecutive variables from the variable list. This will allow our reduction to work even with our extremely restricted clause gadgets.

Building a path from a removal ordering We construct two sequences α and β of variable lists corresponding to removal orderings of $\mathcal{C}_-$ and $\mathcal{C}_+$ respectively. For $0 \leq i \leq n-2$, the list α_i contains the variables remaining after the first i removals in the removal ordering for $\mathcal{C}_-$. In particular, this means that $\alpha_0 = (x_1, x_2, \ldots, x_n)$ and $\alpha_{n-2} = (x_1, x_n)$. β is built similarly from the removal ordering for $\mathcal{C}_+$.

We construct the path P_Φ based on the variable lists in the order $\alpha_{n-2}, \ldots, \alpha_1, \alpha_0, \beta_0, \beta_1, \ldots, \beta_{n-2}$. Such

line is more convenient for our explanation.

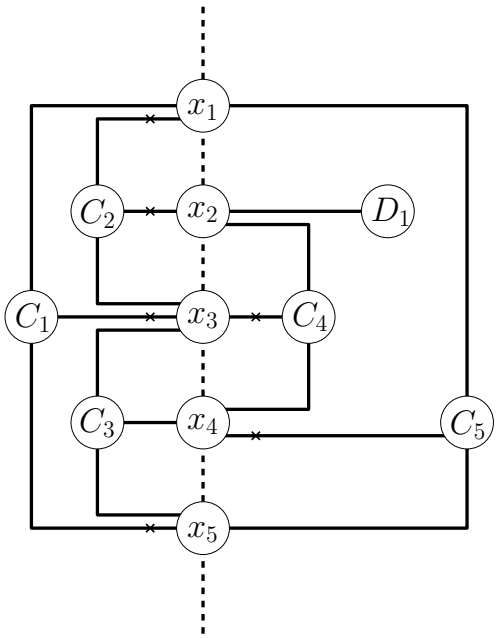

Figure 2: The same layout as in Figure 1 with an additional deletion node. The respective removal orderings are (C_2, C_3, C_1) and (C_4, D_1, C_5). Each variable is only used in between the two clauses that use it as the middle variable, so the issue of clauses acting as 'roadblocks' for middle variables is not a problem.

a path is shown in Figure 3. This path is a basic representation of how Φ can be turned into a linear 'highway' so it can be embedded in T_Φ. The deletion nodes ensure that, for each variable other than x_1 and x_n, the variable's lane is actually bounded by two 'roadblocks'.

Running time A planar embedding of a planar graph can be found in linear time [10, 15]. The other tasks involved in constructing T_Φ from Φ can be performed trivially in polynomial time.

Truth assignment propagation with clause evaluation We can think of the variable assignment as starting in the middle of P_Φ and being propagated out to the left and right. Our technique for propagating a consistent truth assignment in a variable highway is discussed in Section 3. This includes standard variable gadgets used for propagation, as well as deletion gadgets used as endpoints for variable lanes. For a reduction from SAT we need to determine if there is a consistent truth assignment that satisfies the clauses of Φ. Two of the main types of gadgets we need are for evaluating α-clauses while a truth assignment is propagated upwards and for evaluating β-clauses while a truth assignment is propagated downwards. These gadget types, along with the inversion gadget that inverts a variable (swaps the positions of guards representing TRUE and FALSE), are discussed in Section 4. The locations of the gadgets will be determined by P_Φ.

Variable lanes and general layout To propagate a variable assignment around the terrain, our reduction 'reflects' the assignment back and forth over a main valley. Each *reflector* has n *slots* – one for each variable lane. The slots in a reflector are stacked with the slot for x_1 being the highest and the slot for x_n being the lowest. Most reflectors will not transmit information about all n variables since most variable lists in P_Φ do not contain all variables. When a reflector does not transmit information about a variable x_i, the slot for x_i will be empty, *i.e.* it will be a straight line segment (see, *e.g.*, Figure 6). Thus empty slots act as space holders, and the positions of variable slots do not depend on which (or how many) slots are active in a given reflector.

Generally, multiple reflectors (though always a constant number) may be required to implement each step in the path P_Φ. Each reflector takes up the same amount of space, *i.e.* has the same size rectilinear bounding box. Interacting gadgets near the bottom of the terrain are closer to each other than interacting gadgets at the top of the terrain. To ensure that slopes of important lines of sight are of the same order of magnitude, we can 'pad'

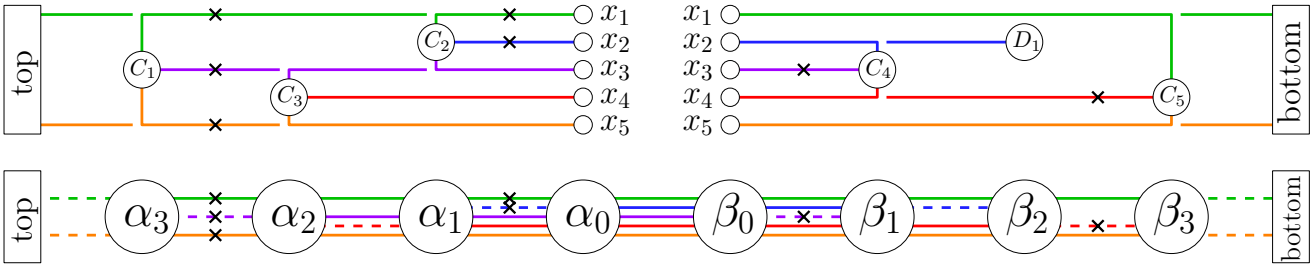

Figure 3: A horizontal layout of Φ (above) that alludes to its treatment as a variable highway. The path representation P_Φ (below). The variables $x_1, \ldots, x_5$ are shown as the lines from top to bottom, with dashed lines representing variables being deleted. Crosses on the lines indicate negations. Note in the α sequence that x_1 is negated because the negative literal $\overline{x_1}$ appears in C_2, and x_1 is later negated again because the positive literal x_1 appears in C_1.

the lower part of the main valley so that all of the reflectors are in the top half of the terrain. In this way, gadgets that interact with each other are always the same horizontal distance apart up to a constant multiplicative factor. We can also add padding to the walls between reflectors so that gadgets that interact with each other are always the same vertical distance apart up to a constant multiplicative factor. Both types of padding increase the size of the terrain by at most a constant factor; neither type is shown in our figures.

3 Truth assignment propagation in a terrain

When studying the computational complexity of a problem it is often useful to consider the problem's locality. If a local change in a terrain can have a global effect on the optimal solution we may be able to exploit this nonlocal behavior to transmit information in a reduction. Specifically, we may be able to use it to transmit a truth assignment to different clauses. With this in mind, our first goal is simply to *propagate a truth assignment* around a terrain. Our greatest concern is ensuring that the truth assigment is *consistent*, *i.e.* that variables have the same value wherever they are represented in the terrain.

In this section we will deal with two types of terrains that introduce some of the important principles used in our full reduction terrains. First we consider *truth assignment propagation terrains*. In these terrains, we have a variable highway for n variables, and every variable slot is active in every reflector. Our main conclusion for these terrains is given as Observation 1. We then introduce deletion gadgets so that variable lanes can have endpoints other than the top and bottom reflectors. Our main conclusion for these terrains with deletion gadgets is given as Observation 2.

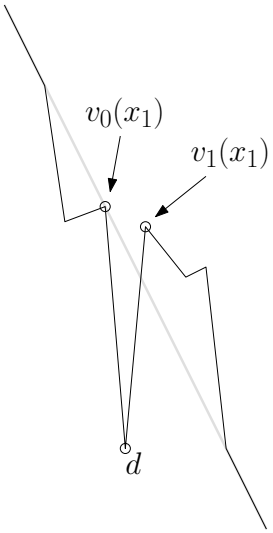

Figure 4: A variable gadget. Any point that sees the distinguished point d is dominated by $v_0(x)$ and $v_1(x)$.

Encoding a truth assignment We will start out as simply as possible, encoding a single Boolean variable without any propagation. An example is shown in Figure 5. The *variable gadget* (see Figure 4) has a distinguished point, d, that can be seen from only two other vertices. These two vertices, call them $v_0(x_1)$ and $v_1(x_1)$, respectively represent an assignment of FALSE and TRUE to the variable x_1. Any point that sees d is dominated by either $v_0(x_1)$ or $v_1(x_1)$. Therefore, for any minimum guarding set there exists a corresponding guarding set of the same size that contains either $v_0(x_1)$ or $v_1(x_1)$. We can assume without loss of generality that any minimum guarding set for the terrain contains a guard on at least one of these points. Similarly, any point that sees the rightmost vertex is dominated by

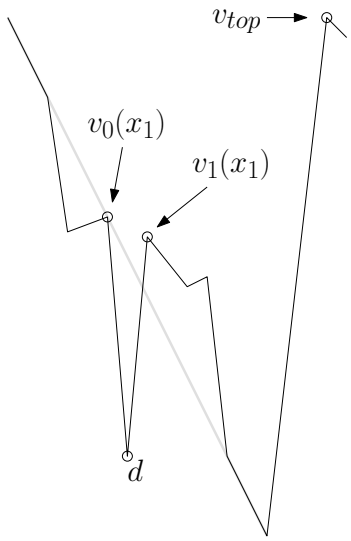

Figure 5: The simplest 'truth assignment propagation terrain' with one varible and no propagation.

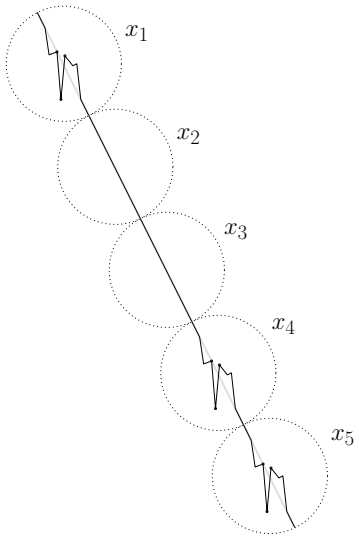

Figure 6: An assignment gadget. The slots corresponding to lanes for x_2 and x_3 are empty.

v_{top} so we can assume that any minimum guarding set contains v_{top}. We make these assumptions in order to discuss minimum guarding sets more cleanly; later on we will make similar assumptions without mention.

To encode an arbitrary number of variables, still without propagation, we simply stack variable gadgets on top of each other to create a basic reflector called an *assignment gadget*. An example is shown in Figure 7. A minimum guarding set for that terrain contains v_{top} as well as one guard for each of the three variable gadgets, corresponding to any truth assignment we want. This can be generalized to a truth assignment for any number of variables.

Distinguished points and internal guards The distinguished point d in a variable gadget cannot be seen from outside the gadget. This ensures that any guarding set contains at least one point in each variable gadget. This is essential for proving correctness of our reduction. Certain gadgets require internal guards, between 0 and 2 depending on the gadget type. If all internal guards in the terrain interact in the right way, *i.e.* if they correspond to a consistent truth assignment satisfying the clauses of Φ, then they are sufficient to guard the entire terrain. If Φ is not satisfiable, the same number of internal guards will be required to see the distinguished points, but at least one additional guard will be required to guard the rest of the terrain. Thus $f(\Phi)$ will simply be the number of internal guards required, and will be trivially computable from the numbers of gadgets of each type. In this accounting we consider the guards required at v_{top} and v_{bottom} to be internal guards.

Propagating a truth assignment Now that we can encode an arbitrary truth assignment, we want to be able to propagate it consistently around the terrain. We do this by *reflecting the assignment* back and forth across a central valley. Each assignment gadget will interact with two assignment gadgets on the opposite side of the valley, taking input from the one above and giving output to the one below. The assignment gadgets on the right side of the valley are mirror images of those on the left side, though the position of guards representing TRUE and FALSE is swapped. An example of the reflecting behavior is shown in Figure 8, a variable interaction is shown in Figure 9, and the details of the variable interaction are shown in Figure 10. The way a variable's assignment is propagated down the terrain holds the key to understanding the complexity of terrains, and is based on the relationship between what on the opposite side of the valley can be seen by guards at $v_0(x)$ or $v_1(x)$. It is possible for $v_0(x)$ to see things $v_1(x)$ cannot because $v_1(x)$ is 'too low'. Similarly, it is possible for $v_1(x)$ to see things $v_0(x)$ cannot because $v_0(x)$ is 'too far to the left'. The details in Figure 10 will be explained shortly.

It is important to point out that the direction in which a truth assignment is propagated is simply a matter of perspective. We can think of the truth assignment as starting at the top and being propagated downwards, as starting at the bottom and being propagated upwards, or as starting in the middle and being propagated both upwards and downwards.

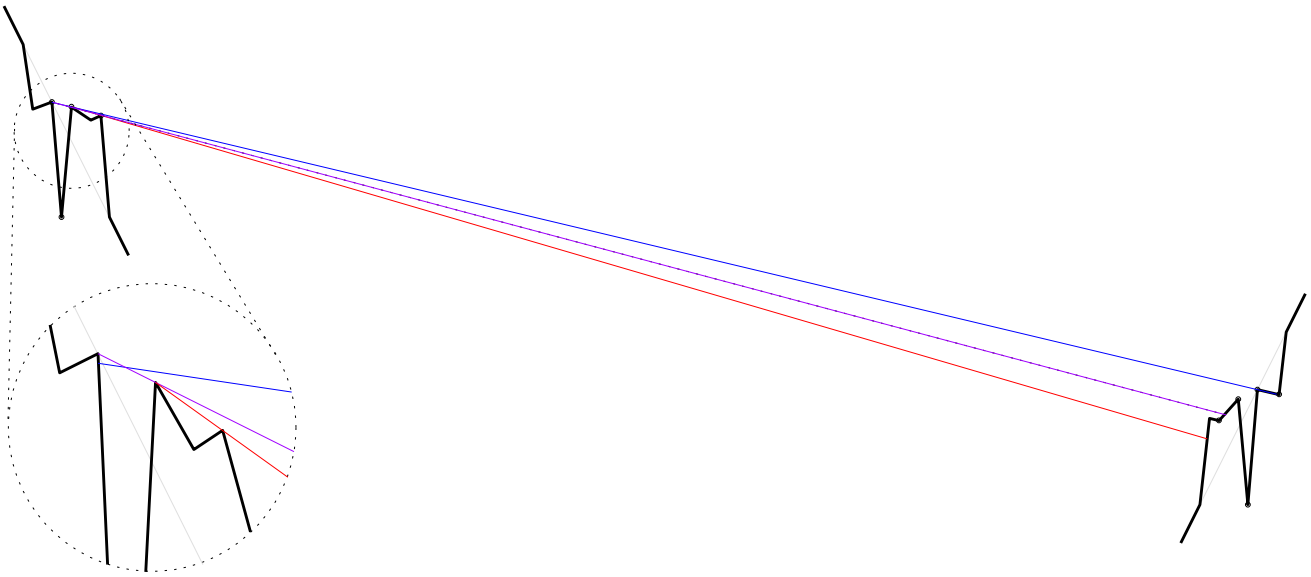

Figure 9: A variable interaction. The inset detail is exaggerated to show how specific lines of sight interact. For more detail see Figure 10.

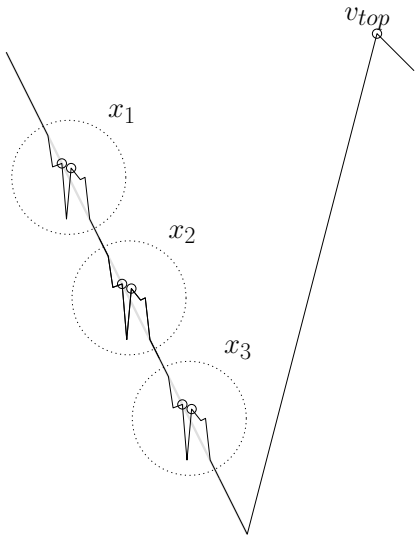

Figure 7: Another 'truth assignment propagation terrain', now with three variables, still no propagation.

Variable gadget interaction A variable gadget only interacts with other gadgets representing the same variable. This interaction is shown coarsely in Figure 8. The necessary guard at v_{top} sees enough of the first (*i.e.* top left) assignment gadget that the first assignment gadget can be optimally guarded by n guards corresponding to any truth assignment for the n variables. This will be the output of the first assignment gadget. After that, zig-zagging down the terrain, the interactions of the variable gadgets are designed to ensure that an assignment gadget can be guarded by n guards if and only if it encodes a truth assignment (its output) that matches the truth assignment encoded in the assignment gadget above (its input). Finally, a guard at v_{bottom} is necessary and sufficient to guard everything below the final assignment gadgets. Thus a 'truth assignment propagation terrain' (see, *e.g.*, Figure 8) with k assignment gadgets propagating n variables can be guarded with $nk + 2$ guards if and only if the truth assignment is consistent; furthermore, this works for any of the 2^n possible truth assignments. This count of $nk + 2$ only works for these truth assignment propagation terrains because every assignment gadget has a slot for each of the n variables so there are nk total variable gadgets; this will not be true in general.

In the detail in Figure 10 the four points $\{d, d', p, q\}$ are of particular interest. To guard d and d' we need at least one of $\{v_0(x), v_1(x)\}$ and at least one of $\{u_0(x), u_1(x)\}$ in our guarding set. The gadgets are configured such that, of these four potential guards, only $v_0(x)$ and $u_1(x)$ see q, and only $v_1(x)$ and $u_0(x)$ see p. Therefore the only pairs of guards that see $\{d, d', p, q\}$ are $\{v_0(x), u_0(x)\}$ and $\{v_1(x), u_1(x)\}$. These pairs correspond to the variable x being set to FALSE or TRUE respectively.

Special care is taken to ensure that guards in a guarding set of size $nk + 2$ do not interfere with the wrong gadgets. For each of the nk variable gadgets we have points of type p and q (see Figure 10). v_{top} can see

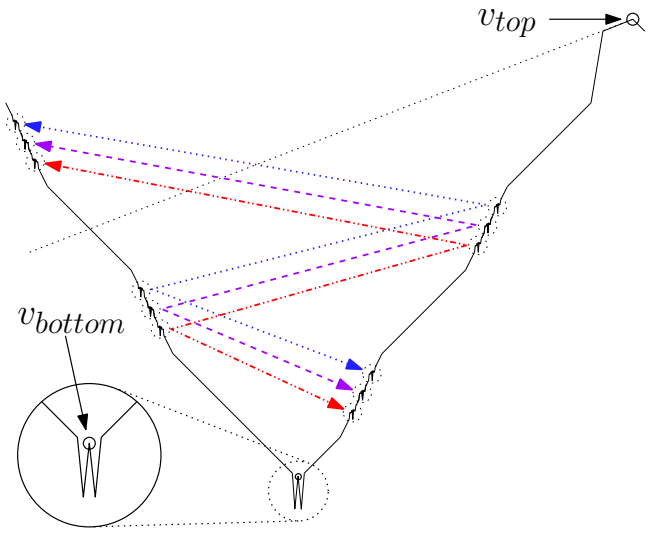

Figure 8: An assignment of three variables being propagated using four reflectors. Note that in an actual terrain used in our reduction, the top and bottom assignment gadgets will have every variable slot empty except for x_1 and x_n.

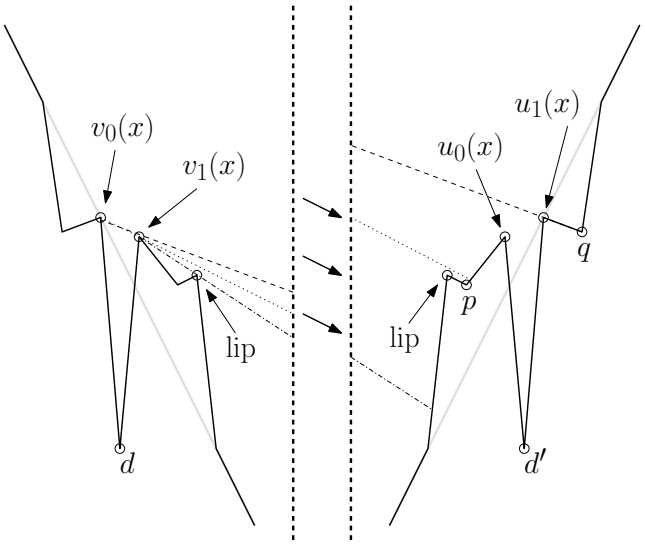

Figure 10: Interaction of variable gadgets. $v_0(x)$ cannot see p because the line of sight is blocked by $v_1(x)$. $v_1(x)$ cannot see q because the line of sight is blocked by $u_1(x)$. One internal guard is required for each gadget.

all such points in the first assignment gadget, but none from any other assignment gadget. v_{bottom} cannot see any at all. The $v_0(x)$ and $v_1(x)$ type guards can only see the appropriate points in their own variable gadget and in the variable gadget below that their variable gadget interacts with. The 'lip' on the gadget ensures $v_0(x)$ and $v_1(x)$ cannot see any variable gadgets further down, and they cannot see any higher points of type p and q because those points sit in 'dimples'.

Managing lines of sight For each pair of variable gadgets that interact, two tweaking phases need to be performed to ensure that important lines of sight either exist or do not exist, as required; tweaking is done by moving certain vertices vertically by small amounts. The first phase is done for each gadget, starting at the bottom of the terrain and proceeding upwards. Then the second phase is done for each gadget, starting at the bottom of the terrain and proceeding upwards. We describe these phases in reference to the interaction shown in Figure 10.

In the first phase, the two vertices to be adjusted vertically are $v_1(x)$ and the lip vertex to the right of $v_1(x)$. First $v_1(x)$ is adjusted so that the line of sight from $v_0(x)$ through $v_1(x)$ hits the terrain on the opposite side between $u_0(x)$ and p. After that, the lip vertex to the right of $v_1(x)$ is adjusted so that the line of sight from $v_1(x)$ passing through this lip vertex hits the terrain on the opposite side below the opposite lip vertex but above any variable gadget below.

In the second phase, the two vertices to be adjusted vertically are p and q. p is adjusted so that the line of sight from p through the adjacent lip vertex hits the terrain on the opposite side below the opposite lip vertex. q is then adjusted so the line of sight from q through $u_1(x)$ hits the terrain on the opposite side on the line segment $(d, v_0(x))$. The line of sight passes just over $v_1(x)$ and hits just below $v_0(x)$.

If the tweaking is done in this order, the tweaking process will not disturb variable gadgets that have already been tweaked. The other gadget types can be tweaked similarly.

We reiterate our main point regarding these truth assignment propagation terrains. Again, this only holds for these demonstrative truth assignment propagation terrains.

Observation 1. *A truth assignment propagation terrain with k assignment gadgets propagating n variables can be guarded using $nk+2$ guards corresponding to any consistent truth assignment. Any guarding set that does not correspond to a consistent truth assignment requires more guards.*

Deletion gadgets In the truth assignment propagation terrains shown thus far, each variable has a lane in the variable highway that spans every assignment gadget, from the top to the bottom. However, for our reduction we will need to be able to place 'roadblocks' to

manage the endpoints of each variable's lane. To end a lane, we use *deletion gadgets*. We need a *downward deletion gadget* for the bottom endpoint of a lane and an *upward deletion gadget* for the top endpoint of a lane.

Both downward and upward deletion gadgets are essentially simplified variable gadgets. A downward deletion gadget is actually just a flat region where a variable gadget would be; in a sense it is a variable gadget that has been simplified to a straight line (see Figure 11). An upward deletion gadget is only slightly more complicated; it requires a single guard that will function similarly to v_{top} but only for a single lane (see Figure 12).

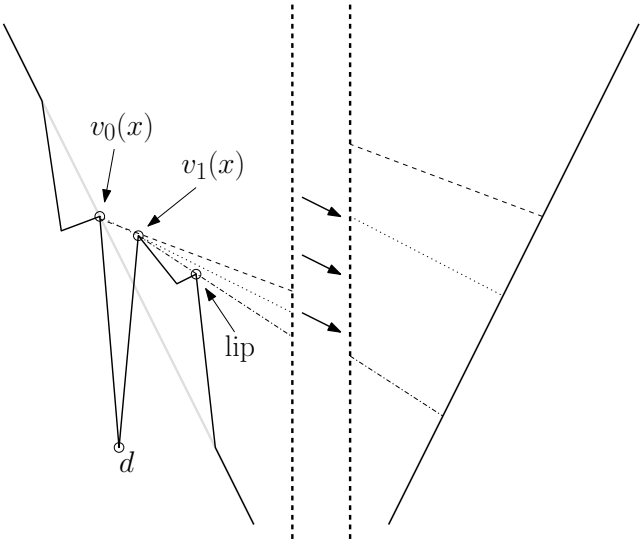

Figure 11: The bottom variable gadget in a variable lane (left) interacts with a downward deletion gadget (right). The downward deletion gadget is simply a flattened region. $v_1(x)$ and the lip vertex are positioned so that neither $v_0(x)$ nor $v_1(x)$ can see gadgets lower down the terrain. One internal guard is required on the left side, no internal guard is required on the right.

OBSERVATION 2. *A truth assignment propagation terrain with deletion gadgets with k_v total variable gadgets and k_{del} upward deletion gadgets can be guarded using $k_v+k_{del}+2$ guards corresponding to any consistent truth assignment. Any guarding set that does not correspond to a consistent truth assignment requires more guards.*

4 Evaluating clauses

Our basic truth assignment propagation process from Section 3 can be thought of as behaving in the following way. A truth assignment for the variables is set in one of the assignment gadgets (it does not matter which). Call this reflector the *starting gadget*. The way the assignment gadgets interact ensures that, in a minimum guarding set, this truth assignment is propagated consistently both upwards and downwards from the starting gadget.

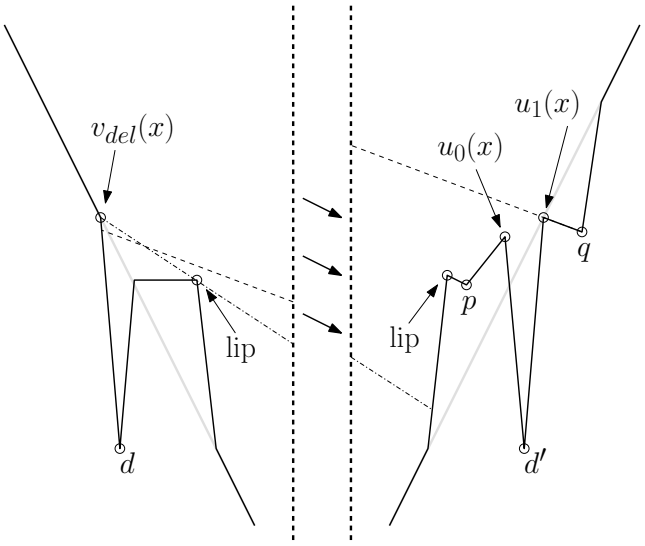

Figure 12: An upward deletion gadget (left) interacts with the top variable gadget in a variable lane (right). The upward deletion gadget is a simplified variable gadget. $v_{del}(x)$ can see both p and q and will be used to guard the distinguished point d. The lip vertex near v_{del} ensures that v_{del} cannot interact with variable gadgets lower down the terrain. One internal guard is required on the left side and one is required on the right.

With the starting gadget fixed, we can consider each assignment gadget to have an input truth assignment and an output truth assignment (though the starting gadget has no input and the topmost and bottommost assignment gadgets have no output). Each assignment gadget above the starting gadget takes input from below and sends its output upwards. Each assignment gadget below the starting gadget takes input from above and sends its output downwards. An assignment gadget can be thought of as an identity gadget since, in a minimum guarding set, its output is the same as its input.

When building T_Φ we will also have a starting gadget, and other gadgets will still have input and output. However, we need more than just an identity gadget. In this section we describe the types of gadgets required to propagate a truth assignment that satisfies Φ. The three gadget types in this section are:

1. **Inversion gadget** – in the variable lane for a variable x_i, this gadget switches the position of the guards representing TRUE and FALSE assignments.

2. **Upward clause gadget** – for three variables x_i, x_j, x_k, with $i < j < k$ and with x_j the only non-empty lane between x_i and x_k, asserts that the clause $(x_i \vee x_j \vee \overline{x_k})$ is satisfied, otherwise at least one extra guard is required. The x_j lane must be empty above this gadget.

3. **Downward clause gadget** – for three variables x_i, x_j, x_k, with $i < j < k$ and with x_j the only non-empty lane between x_i and x_k, asserts that the clause $(x_i \vee \overline{x_j} \vee \overline{x_k})$ is satisfied, otherwise at least one extra guard is required. The x_j lane must be empty below this gadget.

It should be clear from Sections 2 and 3 that these gadgets are sufficient to complete our reduction from PLANAR 3-SAT.

Gadget input and output Our reduction ensures that in a guarding set of size $f(\Phi)$, guards can only be on certain special types of internal vertices[2]:

1. v_{top} and v_{bottom}
2. vertices of type $v_{del}(x)$ in upward deletion gadgets
3. vertices of type $v_0(x)$, $v_1(x)$, $u_0(x)$, and $u_1(x)$[3] in standard and modified[4] variable gadgets
4. vertices of type $u_0(x)$, $u_1(x)$, $w_0(x)$ and $w_1(x)$ in inversion gadgets.

The output of a gadget only has to be valid in a guarding of size $f(\Phi)$, otherwise it is not necessarily valid and all assumptions are allowed to fall apart since a minimum guarding set must correspond to a truth assignment satisfying Φ if and only if it has size $f(\Phi)$. $f(\Phi)$ internal guards will still be necessary, but they will not necessarily correspond to a consistent and satisfying truth assignment if there are additional guards.

4.1 Gadget functions

Inversion gadget For a single variable, an *inversion gadget* swaps the positions of guards representing the TRUE and FALSE truth assignments. By default, in a variable gadget the vertex $v_0(x)$ (representing FALSE) will be above the vertex $v_1(x)$ (representing TRUE). However, if in the lane for x there are an odd number of inversion gadgets between the variable gadget in question and the start gadget, $v_1(x)$ will be above $v_0(x)$. An inversion gadget replaces a single variable gadget in the lane corresponding to the variable being inverted.

A standard variable gadget has two possible minimum guarding sets: $\{v_0(x)\}$ and $\{v_1(x)\}$ (see Figure 4). The special variable slot in an inversion gadget also has two possible minimum guarding sets, though each has two guards instead of just one. From each minimum guarding set, however, only one guard affects the output of the gadget. An inversion gadget is shown in Figure 13, with a detailed explanation in the caption. A larger view is shown in Figure 14.

Upward clause gadget An upward clause gadget takes as input (from below) a variable assignment and outputs a variable assignment with one variable removed. These gadgets are used to implement clauses in the α sequence of P_Φ. For three variables x_i, x_j, x_k that are adjacent[5] in the input highway ($i < j < k$), the gadget will delete the middle variable x_j. The gadget can be guarded with a minimum number of internal guards if and only if the following two conditions hold:

1. Each variable in the input (except x_j) must have the same value in the output.
2. The clause $(x_i \vee x_j \vee \overline{x_k})$ is satisfied by the input.

Another way of saying that the clause $(x_i \vee x_j \vee \overline{x_k})$ is satisfied by the input is to say that the input can include $\overline{x_j}$ only if $(x_i \vee \overline{x_k})$ evaluates to TRUE in the input and output.

In an upward clause gadget involving the variables x_i, x_j, x_k, the variable x_j is deleted from the assignment. This takes place in a single reflector, in which all active variable slots except x_j contain variable gadgets. Assuming the clause being evaluated is $(x_i \vee x_j \vee \overline{x_k})$, we explain what is put in the place of a variable gadget for x_j. The key is the special point q_j. Of the 'output' points $v_0(x_i)$, $v_1(x_i)$, $v_0(x_k)$ and $v_1(x_k)$, only $v_1(x_i)$ and $v_0(x_k)$ can see q_j. Of the 6 'input' points, only $u_1(x_j)$ can see q_j. Since the variables x_i and x_k will have the same output as input, this means that $\overline{x_j}$ can be in the input if and only if x_i or $\overline{x_k}$ is in the input. Thus a minimum guarding of the gadget ensures that the clause $(x_i \vee x_j \vee \overline{x_k})$ is satisfied.

Downward clause gadget A downward clause gadget takes as input (from above) a variable assignment and outputs a variable assignment with one variable removed. These gadgets are used to implement clauses in the β sequence of P_Φ. For three variables x_i, x_j,

[2] More precisely, in a guarding set of size $f(\Phi)$, any guard not on one of these point types can be replaced by a guard on one of these point types with no loss in visibility.

[3] In variable gadgets, vertices of type $u_0(x)$ and $u_1(x)$ are actually also vertices of type $v_0(x)$ and $v_1(x)$.

[4] Modified variable gadgets are used in clause gadgets.

[5] By adjacent we mean that there are no active lanes in the input between x_i and x_j or between x_j and x_k.

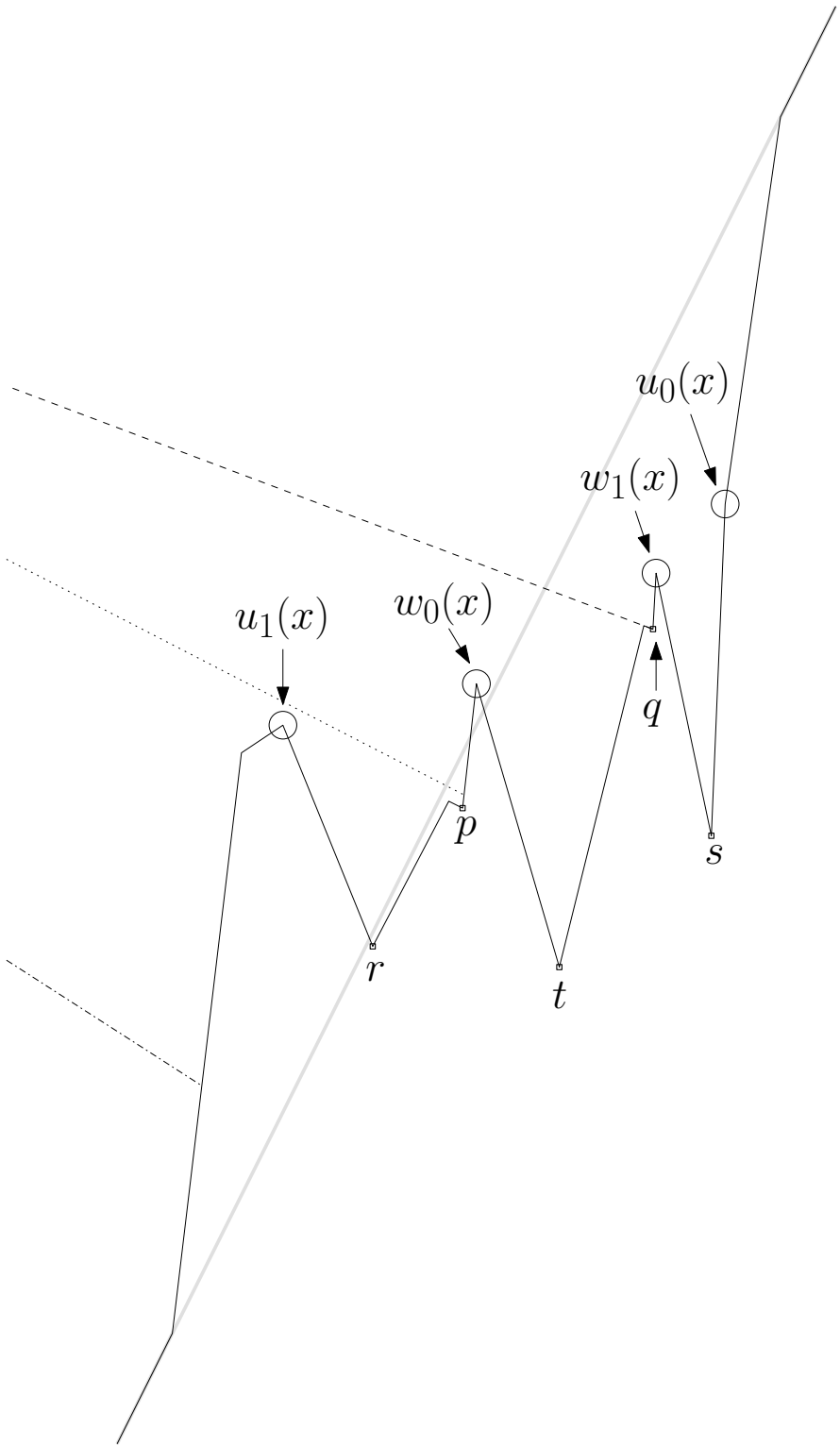

Figure 14: Detail of inversion gadget. See Figure 13 for details of its interaction with variable gadgets above and below it.

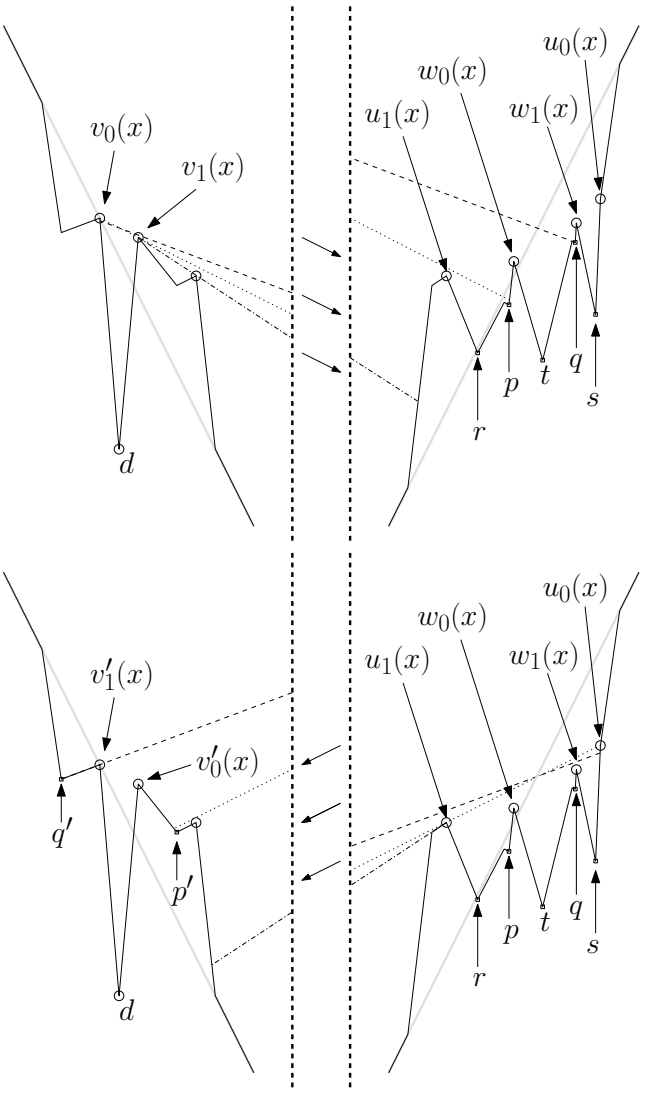

Figure 13: Inversion gadget detail. Input is shown above and output is shown below, with the same inversion gadget on the top right and bottom right. There are only two minimum guarding sets that include exactly one of the input guards $v_0(x)$ and $v_1(x)$. These sets are $\{v_0(x), u_0(x), w_0(x), v_0'(x)\}$ and $\{v_1(x), u_1(x), w_1(x), v_1'(x)\}$. Two internal guards are required in the inversion gadget (*i.e.* in the range $[u_1(x), u_0(x)]$), since r is not seen by anything outside the range $[u_1(x), w_0(x)]$ and s is not seen by anything outside the range $[w_1(x), u_0(x)]$. t is not seen by anything outside the range $[w_0(x), w_1(x)]$. Though it is difficult to see, p is seen by $v_1(x)$ and $w_0(x)$ but not by $u_1(x)$. q is seen by $v_0(x)$ and $w_1(x)$ but not by $w_0(x)$. r is seen by $u_1(x)$ and $w_0(x)$. The two potential output guards are $u_0(x)$ and $u_1(x)$. $w_0(x)$ and $w_1(x)$ are useless outside this gadget. See Figure 14 for a closer view.

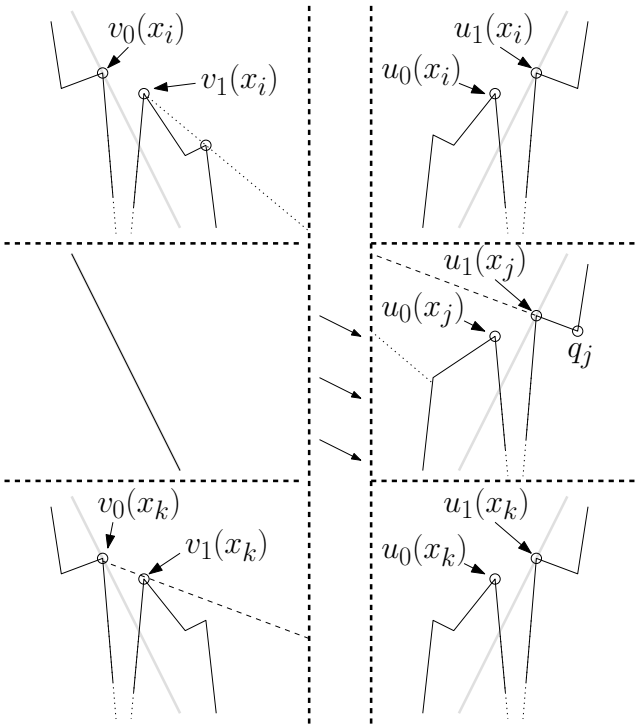

Figure 15: Upward clause gadget interaction. In every reflector above this gadget the x_j slot is empty. Each of the 6 slots shown, except the middle left, requires one internal guard. No extra guards are required iff the clause $(x_i \vee x_j \vee \overline{x_k})$ is satisfied, since the only internal guards that see q_j are $v_1(x_i)$, $u_1(x_j)$, and $v_0(x_k)$. Note that, in the top left slot, the lip vertex has been lowered so that $v_1(x_i)$ can see q_j. Also, q_j has been adjusted so that $v_0(x_k)$ can see it. The two lines of sight relevant to these adjustments are shown.

x_k that are adjacent in the input highway, the gadget will delete the middle variable x_j. The gadget can be guarded with a minimum number of internal guards if and only if the following two conditions hold:

1. Each variable in the input (except x_j) must have the same value in the output.
2. The clause $(x_i \vee \overline{x_j} \vee \overline{x_k})$ is satisfied by the input.

Another way of saying that the clause $(x_i \vee \overline{x_j} \vee \overline{x_k})$ is satisfied by the input is to say that the input can include x_j only if $(x_i \vee \overline{x_k})$ evaluates to TRUE in the input and output.

In a downward clause gadget involving the variables x_i, x_j, x_k, the variable x_j is deleted from the assignment. This takes place in a single reflector, in which all active variable slots except x_j contain variable gadgets. Assuming the clause being evaluated is $(x_i \vee \overline{x}_j \vee \overline{x}_k)$,

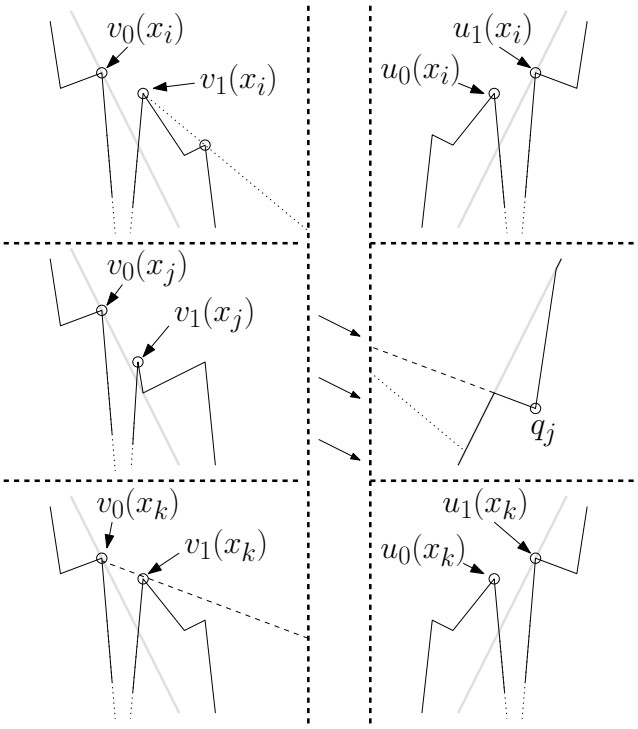

Figure 16: Downward clause gadget interaction. In every reflector below this gadget the x_j slot is empty. Each of the 6 slots shown, except the middle right, requires one internal guard. No extra guards are required iff the clause $(x_i \vee \overline{x_j} \vee \overline{x_k})$ is satisfied, since the only internal guards that see q_j are $v_1(x_i)$, $v_0(x_j)$, and $v_0(x_k)$. Note that, in the top left slot, the lip vertex has been lowered so that $v_1(x_i)$ can see q_j. Also, q_j has been adjusted so that $v_0(x_k)$ can see it. The two lines of sight relevant to these adjustments are shown.

we explain what is put in the place of the variable gadget for x_j. The key is the special point q_j. Of the 'input' points $v_0(x_i)$, $v_1(x_i)$, $v_0(x_j)$, $v_1(x_j)$, $v_0(x_k)$ and $v_1(x_k)$, only $v_1(x_i)$, $v_0(x_j)$ and $v_0(x_k)$ can see q_j. Thus a minimum guarding of the gadget ensures that the clause $(x_i \vee \overline{x}_j \vee \overline{x}_k)$ is satisfied. All variables except x_j will have the same output as input.

5 Conclusions and future work

We have shown that terrain guarding is NP-hard. With the PTAS for terrain guarding given by Gibson *et al.* [9], this essentially resolves the approximability of the problem. The biggest remaining question regarding the complexity of terrain guarding is whether or not it is fixed-parameter tractable.

Acknowledgements

The authors would like to thank Kasturi Varadarajan, Bengt Nilsson, and Will Evans for their valuable comments, discussions and suggestions.

References

[1] B. Ben-Moshe, M. Katz, and J. Mitchell. *A Constant-Factor Approximation Algorithm for Optimal 1.5D Terrain Guarding*. SIAM Journal on Computing, 36(6): 1631–1647, 2007.

[2] D. Z. Chen, V. Estivill-Castro, and J. Urrutia. *Optimal guarding of polygons and monotone chains*. Canadian Conference on Computational Geometry, 1995.

[3] K. L. Clarkson, K. Varadarajan. *Improved Approximation Algorithms for Geometric Set Cover*. Proc. 21st ACM Symposium on Computational Geometry, 2005.

[4] E. D. Demaine and J. ORourke. *Open problems: Open problems from CCCG 2005*. Proceedings of the 18th Canadian Conference on Computational Geometry, pages 75–80, 2006.

[5] A. Deshpande, T. Kim, E. D. Demaine, and S. E. Sarma. *A Pseudopolynomial Time $O(\log n)$-Approximation Algorithm for Art Gallery Problems*. WADS, 2007.

[6] S. Eidenbenz. *Inapproximability Results for Guarding Polygons without Holes*. Lecture Notes in Computer Science, vol. 1533 (ISAAC'98), 427–436, 1998.

[7] K. Elbassioni, E. Krohn, D. Matijevic, J. Mestre, D. Severdija. *Improved Approximations for Guarding 1.5-Dimensional Terrains*. Symposium on Theoretical Aspects of Computer Science, 2009.

[8] S. Ghosh. *Approximation algorithms for art gallery problems*. Proc. Canadian Information Processing Society Congress, 1987.

[9] M. Gibson, G. Kanade, E. Krohn, K. Varadarajan. *An Approximation Scheme for Terrain Guarding*. APPROX, 2009.

[10] J. Hopcroft and R. Tarjan. *Efficient planarity testing.* J. ACM, 21(4): 549–568, 1974.

[11] J. King. *A 4-Approximation Algorithm for Guarding 1.5-Dimensional Terrains.* Lecture Notes in Computer Science (3887), 629–640, 2006.

[12] D. E. Knuth and A. Raghunathan. *The problem of compatible representatives.* SIAM J. Discret. Math., 5(3): 422–427, 1992.

[13] D. Lee and A. Lin. *Computational complexity of art gallery problems.* IEEE Trans. Inform. Theory, vol. 32, 276–282, 1986.

[14] D. Lichtenstein. *Planar Formulae and their uses*, SIAM Journal on Computing, 11(2): 329–343, 1982.

[15] K. Mehlhorn and P. Mutzel. *On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm.* Algorithmica, 16(2): 233–242, 1996.

[16] W. Mulzer and G. Rote. *Minimum-weight triangulation is NP-hard.* J. ACM, 55(2): 1–29, 2008.

[17] B. Nilsson. *Approximate guarding of monotone and rectilinear polygons.* Proceedings of ICALP 2005, 1362–1373, 2005.