A Faster Algorithm for Finding Tarski Fixed Points

JOHN FEARNLEY, University of Liverpool, United Kingdon
DÖMÖTÖR PÁLVÖLGYI, Eötvös Loránd University (ELTE), Hungary
RAHUL SAVANI, University of Liverpool, United Kingdon

Dang et al. have given an algorithm that can find a Tarski fixed point in a k-dimensional lattice of width n using $O(\log^k n)$ queries [2]. Multiple authors have conjectured that this algorithm is optimal [2, 7], and indeed this has been proven for two-dimensional instances [7]. We show that these conjectures are false in dimension three or higher by giving an $O(\log^2 n)$ query algorithm for the three-dimensional Tarski problem. We also give a new decomposition theorem for k-dimensional Tarski problems which, in combination with our new algorithm for three dimensions, gives an $O(\log^{2\lceil k/3 \rceil} n)$ query algorithm for the k-dimensional problem.

CCS Concepts: • **Theory of computation** → Design and analysis of algorithms.

Additional Key Words and Phrases: query complexity, Tarski fixed points, total function problem

1 INTRODUCTION

Tarski's fixed point theorem states that every order preserving function on a complete lattice has a greatest and least fixed point [12], and therefore in particular, every such function has at least one fixed point. Recently, there has been interest in the complexity of finding such a fixed point. This is due to its applications, including computing Nash equilibria of supermodular games and finding the solution of a simple stochastic game [7].

Prior work has focused on the complete lattice L defined by a k-dimensional grid of width n. Dang, Qi, and Ye [2] give an algorithm that finds a fixed point of a function $f : L \to L$ using $O(\log^k n)$ queries to f. This algorithm uses recursive binary search, where a k-dimensional problem is solved by making $\log n$ recursive calls on $(k-1)$-dimensional sub-instances. They conjectured that this algorithm is optimal.

Later work of Etessami, Papadimitriou, Rubinstein, and Yannakakis took the first step towards proving this [7]. They showed that finding a Tarski fixed point in a two-dimensional lattice requires $\Omega(\log^2 n)$ queries, meaning that the Dang et al. algorithm is indeed optimal in the two-dimensional case. Etessami et al. conjectured that the Dang et al. algorithm is optimal for constant k, and they leave as an explicit open problem the question of whether their lower bound can be extended to dimension three or beyond.

Our contribution. In this paper we show that, surprisingly, the Dang et al. algorithm is not optimal in dimension three, or any higher dimension, and so we falsify the prior conjectures. We do this by giving an algorithm that can find a Tarski fixed point in three dimensions using $O(\log^2 n)$ queries, thereby beating the $O(\log^3 n)$ query algorithm of Dang et al.

The Dang et al. algorithm solves a three-dimensional instance by making recursive calls to find a fixed point of $\log n$ distinct two-dimensional sub-instances. Our key innovation is to point out that one does not need to find a fixed point of the two-dimensional sub-instance to make progress. Instead, we define the concept of an *inner algorithm* (Definition 3.1) that, given a two-dimensional sub-instance, is permitted to return any point that lies in the up or down set of the three-dimensional instance (defined formally later). This is a much larger set of points, so whereas finding a fixed point of a two-dimensional instance requires $\Omega(\log^2 n)$ queries [7], we give a $O(\log n)$ query inner algorithm for two-dimensional instances. This inner algorithm is quite involved, and is the main technical contribution of the paper.

Authors' addresses: John Fearnley, University of Liverpool, United Kingdon, john.fearnley@liverpool.ac.uk; Dömötör Pálvölgyi, Eötvös Loránd University (ELTE), Hungary, dom@cs.elte.hu; Rahul Savani, University of Liverpool, United Kingdon, rahul.savani@liverpool.ac.uk.

We show that, given an inner algorithm for dimension $k-1$, a reasonably straightforward *outer algorithm* can find a Tarski fixed point by making $O(k \cdot \log n)$ calls to the inner algorithm. Thus we obtain a $O(\log^2 n)$ query algorithm for the case where $k = 3$. We leave as an open problem the question of whether efficient inner algorithms exist in higher dimensions.

For higher-dimensional instances, we show a decomposition theorem: if a-dimensional Tarski problems can be solved in q_a queries, and b-dimensional Tarski problems can be solved in q_b queries, then $(a \cdot b)$-dimensional Tarski can be solved in $q_a \cdot (q_b + 2)$ queries. This then allows us to use our new algorithm for three-dimensional Tarski problems to obtain an algorithm that solves k-dimensional Tarski problems using $O(\log^{2\lceil k/3 \rceil} n)$ queries, a substantial improvement over the $O(\log^k n)$ algorithm of Dang et al.[2]. A preliminary version of this paper [10] used a weaker decomposition theorem and gave a correspondingly weaker improvement.

Though we state our results in terms of query complexity for the sake of simplicity, it should be pointed out that all of our algorithms run in polynomial time. Specifically, our algorithms will run in $O(\text{poly}(\log n, k) \cdot \log^{2\lceil k/3 \rceil} n)$ time when the function is presented as a Boolean circuit of size $\text{poly}(\log n, k)$.

Related work. Etessami et al. also studied the computational complexity of the Tarski problem [7], showing that the problem lies in PPAD and PLS. However, the exact complexity of the problem remains open. It is not clear whether the problem is PPAD ∩ PLS-complete [8], or contained in some other lower class such as EOPL or UEOPL [9].

Tarski's fixed point theorem has been applied in a wide range of settings within Economics [11, 13, 14], and in particular to settings that can be captured by supermodular games, which are in fact equivalent to the Tarski problem [7]. In terms of algorithms, Echenique [6] studied the problem of computing all pure equilibria of a supermodular game, which is at least as hard as finding the greatest or least fixed point of the Tarski problem, which is itself NP-hard [7]. There have also been several papers that study properties of Tarski fixed points, such as the complexity of deciding whether a fixed point is unique [2–5]. The Tarski problem has also been studied in the setting where the partial order is given by an oracle [1].

2 PRELIMINARIES

Lattices. We work with a complete lattice defined over a k-dimensional grid of points. We define $\text{Lat}(n_1, n_2, \ldots, n_k)$ to be the k-dimensional lattice with side-lengths given by $n_1, \ldots, n_k$. That is, $\text{Lat}(n_1, n_2, \ldots, n_k)$ contains every $x \in \mathbb{N}^k$ such that $1 \leq x_i \leq n_i$ for all $i = 1, \ldots, k$. Throughout, we use k to denote the dimensionality of the lattice, and $n = \max_{i=1}^{k} n_i$ to be the width of the widest dimension. We use $\preceq$ to denote the natural partial order over this lattice with $x \preceq y$ if and only if $x, y \in L$ and $x_i \leq y_i$ for all $i \leq k$.

The Tarski fixed point problem. Given a lattice L, a function $f : L \to L$ is *order preserving* if $f(x) \preceq f(y)$ whenever $x \preceq y$. A point $x \in L$ is *fixed point* of f if $f(x) = x$. A weak version of Tarski's theorem can be stated as follows.

Theorem 2.1 ([12]). *Every order preserving function on a complete lattice has a fixed point.*

Thus, we can define a total search problem for Tarski's fixed point theorem.

Definition 2.2 (Tarski). Given a lattice L, and a function $f : L \to L$, find one of:

(T1) A point $x \in L$ such that $f(x) = x$.
(T2) Two points $x, y \in L$ such that $x \preceq y$ and $f(x) \not\preceq f(y)$.

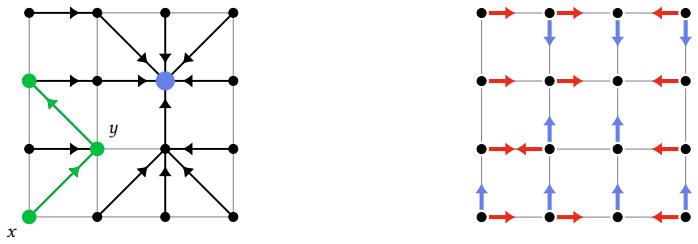

Fig. 1. Left: a TARSKI instance. Right: our diagramming notation for the same instance.

Solutions of type (T1) are fixed points of f, whereas solutions of type (T2) witness that f is not an order preserving function. By Tarski's theorem, if a function f has no solutions of type (T2), then it must have a solution of type (T1), and so TARSKI is a total problem.

The left-hand picture in Figure 1 gives an example of a two-dimensional TARSKI instance. The least element of the lattice is the bottom-left point in the grid, while the greatest element is the top-right point in the grid. The blue point is a fixed point, and so is a (T1) solution, while the highlighted green arrows give an example of an order preservation violation, and so (x, y) is a (T2) solution.

Throughout the paper we will use a diagramming notation, shown on the right in Figure 1, that decomposes the dimensions of the instance. The red arrows correspond to dimension 1, where an arrow pointing to the left indicates that $f(x)_1 \le x_1$, while an arrow to the right[1] indicates that $x_1 \le f(x)_1$. Blue arrows do the same thing for dimension 2, and we will use green arrows for dimension 3 in the cases where this is relevant.

The up and down sets. Given a function f over a lattice L, we define $\text{Up}(f) = \{x \in L \ : \ x \preceq f(x)\}$, and $\text{Down}(f) = \{x \in L \ : \ f(x) \preceq x\}$. We call $\text{Up}(f)$, the *up set*, which contains all points in which f goes up according to the ordering $\preceq$, and likewise we call $\text{Down}(f)$ the *down set*. Note that the set of fixed points of f is exactly $\text{Up}(f) \cap \text{Down}(f)$.

Slices. A *slice* of the lattice L is defined by a tuple $s = (s_1, s_2, \ldots, s_k)$, where each $s_i \in \mathbb{N} \cup \{*\}$. The idea is that, if $s_i \neq *$, then we fix dimension i of L to be s_i, and if $s_i = *$, then we allow dimension i of L to be free. Formally, we define the sliced lattice $L_s = \{x \in L \ : \ x_i = s_i \text{ whenever } s_i \neq *\}$. We say that a slice is a *principle slice* if it fixes exactly one dimension and leaves the others free. For example $(1, *, *)$, $(*, 33, *)$, and $(*, *, 261)$ are all principle slices of a three-dimensional lattice.

Given a slice s, and a function $f : L \to L$, we define $f_s : L_s \to L_s$ to be the *restriction* of f to L_s. Specifically, for each $x \in L_s$, we define $(f_s(x))_i = f(x)_i$ if $s_i = *$, and $(f_s(x))_i = s_i$ otherwise. This definition projects the function f down onto the slice s.

A fact that we will use repeatedly in the paper is that an order preservation violation in a slice s is also an order preservation violation for the whole instance. More formally, if $x, y \in L_s$ satisfy $x \preceq y$ and $f_s(x) \not\preceq f_s(y)$, then we also have $f(x) \not\preceq f(y)$, since there exists a dimension i such that $f(x)_i = f_s(x)_i > f_s(y)_i = f(y)_i$.

Sub-instances. A *sub-instance* of a lattice L is defined by two points $x, y \in L$ that satisfy $x \preceq y$. Informally, the sub-instance defined by x and y is the lattice containing all points between x and y. Formally, we define $L_{x,y} = \{a \in L \ : \ x \preceq a \preceq y\}$.

[1] If $x_1 = f(x)_1$ we could use either arrow, but will clarify in the text whenever this ambiguity matters.

3 THE OUTER ALGORITHM

The task of the outer algorithm is to find a solution to the Tarski instance by making $O(k \cdot \log n)$ calls to the inner algorithm. We state our results for dimension k, even though we only apply the outer algorithm with $k = 3$, since, in the future, an efficient inner algorithm in higher dimensions may be found. Formally, an inner algorithm is defined as follows.

Definition 3.1 (Inner algorithm). An inner algorithm for a Tarski instance takes as input a sub-instance $L_{a,b}$ with $a \in \text{Up}(f)$ and $b \in \text{Down}(f)$, and a principle slice s of that sub-instance. It outputs one of the following.

- A point $x \in L_{a,b} \cap L_s$ such that $x \in \text{Up}(f)$ or $x \in \text{Down}(f)$.
- Two points $x, y \in L_{a,b}$ that witness a violation of the order preservation of f.

It is important to understand that here we are looking for points that lie in the up or down set of the *three-dimensional* instance, a point that lies in $\text{Up}(f_s)$ but for which f goes down in the third dimension would not satisfy this criterion.

The algorithm. Throughout the outer algorithm, we will maintain two points $x, y \in L$ with the invariant that $x \preceq y$ and $x \in \text{Up}(f)$ and $y \in \text{Down}(f)$. The following lemma implies that if x and y satisfy the invariant, then $L_{x,y}$ must contain a solution to the Tarski problem. This will allow us to focus on smaller and smaller instances that are guaranteed to contain a solution.

Lemma 3.2. *Let L be a lattice and $f : L \to L$ be a Tarski instance. If there are two points $a, b \in L$ satisfying $a \preceq b$, $a \in \text{Up}(f)$, and $b \in \text{Down}(f)$, then one of the following exists.*

- *A point $x \in L_{a,b}$ satisfying $f(x) = x$.*
- *Two points $x, y \in L_{a,b}$ satisfying $x \preceq y$ and $f(x) \not\preceq f(y)$.*

Moreover, there is an algorithm that finds one of the above using $O(\sum_{i=1}^{k}(a_i - b_i))$ queries.

Before we prove Lemma 3.2, we first prove the following auxiliary lemma. It states that if x is in the up set, and i is a dimension for which f moves strictly upwards, then either the point x' that is directly above x in dimension i is also in the upset, or x and x' witness a violation of the order preservation of f.

Lemma 3.3. *Let L be a lattice, $f : L \to L$ be a Tarski instance, $x \in L$ be a point satisfying $x \in \text{Up}(f)$, and i be a dimension such that $x_i < f(x)_i$. Define the point x' so that*

$$x'_j = \begin{cases} x_j + 1 & \text{if } j = i, \\ x_j & \text{if } j \neq i. \end{cases}$$

Either $x' \in \text{Up}(f)$, or x and x' witness a violation of the order preservation of f.

Proof. For dimension i, if $f(x')_i < x'_i$ then we have

$$f(x')_i < x'_i \leq f(x)_i,$$

where the second inequality follows from the assumption that $f(x)_i > x_i$, and the fact that $x'_i = x_i + 1$. Hence, if $f(x')_i < x'_i$, then x and x' witness a violation of the order preservation of f since $x \preceq x'$ but $x \not\preceq f(x')$.

For a dimension $j \neq i$, if $f(x')_j < x'_j$ then we have

$$f(x')_j < x'_j = x_j \leq f(x)_j,$$

where the equality follows from the fact that $x_j = x'_j$ by definition, and the final inequality follows from the assumption that $x \in \mathrm{Up}(f)$. Hence, if $f(x')_j < x'_j$, then we have that $x \preceq x'$ while $f(x) \not\preceq f(x')$, and so x and x' witness a violation of the order preservation of f.

From what we have proved above, we know that if x and x' do not violate order preservation, then we have $x'_j \leq f(x')_j$ for all dimensions j, which implies that $x' \in \mathrm{Up}(f)$. □

We now prove Lemma 3.2. Here we repeatedly apply Lemma 3.3 to generate a sequence of points in the up set that start at a and monotonically increase according to $\preceq$. If the path ends inside $L_{a,b}$ then we show that it must end at a solution, while if it leaves $L_{a,b}$, then we show that there is a violation of order preservation between b and the point at which the path leaves the sub-instance.

Proof of Lemma 3.2. Since $a \in \mathrm{Up}(f)$, we have that either a is a fixed point of f, or there exists a dimension i such that $a_i < f(a)_i$. In the former case we are done, so let us assume that the latter is true. This means that we can apply Lemma 3.3 to obtain a point $a' = a + e_j$, where e_j is the unit vector in dimension j, such that either a and a' witness an order preservation violation of f, or $a' \in \mathrm{Up}(f)$.

By repeatedly applying the argument above, we can construct a sequence of points

$$a = a_1, a_2, a_3, \ldots, a_p$$

such that for all $i < p$ we have that and $a_i = a_{i-1} + e_j$ for some dimension j, and $a_i \in \mathrm{Up}(f)$. The sequence ends at the point a_p where we can no longer apply the argument, which means that either a_p and a_{p-1} witness the order preservation violation of f, or there is no index j such that $(a_p)_j < f(a_p)_j$, meaning that a_p is a fixed point since $a_p \in \mathrm{Up}(f)$. Note that the sequence cannot be infinite since $a_i \prec a_{i+1}$ for all i, and the lattice is finite.

If $a_p \preceq b$ then we are immediately done, since this implies that we have one of the required solutions in the sub-instance $L_{a,b}$. On the other hand, if $a_p \not\preceq b$, then let i be the largest index such that $a_i \preceq b$, and let j be the dimension such that $a_{i+1} = a_i + e_j$. Note that we have $b_j = (a_i)_j$, since otherwise we would have $a_{i+1} \preceq b$. We have

$$f(b)_j \leq b_j = (a_i)_j < f(a_i)_j,$$

where the first inequality holds because $b \in \mathrm{Down}(f)$, and the final inequality holds because $a_{i+1} = a_i + e_j$, which can only occur when $(a_i)_j < f(a_i)_j$. Hence we have $a_i \preceq b$, but $f(a_i) \not\preceq f(b)$, and so a_i and b witness a violation of the order preservation of f. Furthermore, both a_i and b lie in the sub-instance $L_{a,b}$, and so the proof is complete.

For the algorithm, note that the arguments above imply that it is sufficient to find either a_p, or the index i such that $a_i \preceq b$ and $a_{i+1} \not\preceq b$ (i must be unique if it exists, since $a_j \prec a_{j+1}$ for all j). Each element of the sequence can be constructed by making a single query to f, and since each step of the sequence strictly increases one coordinate of the point, we have that there can be at most $\sum_{i=1}^{k}(a_i - b_i)$ points a_i of the sequence satisfying $a_i \preceq b$. Thus the algorithm makes at most $O(\sum_{i=1}^{k}(a_i - b_i))$ queries. □

We now describe the algorithm. Initially we set $x = (1, 1, \ldots, 1)$, which is the least element, and $y = (n_1, n_2, \ldots, n_k)$, which is the greatest element. Note that $x \preceq f(x)$ holds because x is the least element, and likewise $f(y) \preceq y$ holds because y is the greatest element, so the invariant holds for these two points.

Each iteration of the outer algorithm will reduce the number of points in $L_{x,y}$ by a factor of two. To do this, the algorithm selects a largest dimension of that sub-instance, which is a dimension i that maximizes $y_i - x_i$. It then makes a call to the inner algorithm for the principle slice s defined so that $s_i = \lfloor (y_i - x_i)/2 \rfloor$ and $s_j = *$ for all $j \neq i$.

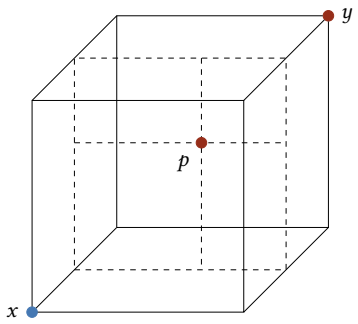

Fig. 2. One iteration of the outer algorithm. The dashed lines show the principle slice chosen by the algorithm, and the point p is the point returned by the inner algorithm. In this case $p \in \text{Down}(f)$, and so the algorithm focuses on the sub-instance $L_{x,p}$.

(1) If the inner algorithm returns a violation of order preservation in the slice, then this is also an order preservation violation in L, and so the algorithm returns this and terminates.
(2) If the inner algorithm returns a point p in the slice such that $p \in \text{Up}(f)$, then the algorithm sets $x := p$ and moves to the next iteration.
(3) If the inner algorithm returns a point p such that $p \in \text{Down}(f)$, then the algorithm sets $y := p$ and moves to the next iteration.

Figure 2 gives an example of this procedure.

The algorithm can continue as long as there exists a dimension i such that $y_i - x_i \geq 2$, since this ensures that there will exist a principle slice strictly between x and y in dimension i that cuts the sub-instance in half. Note that there can be at most $k \cdot \log n$ iterations of the algorithm before we arrive at the final sub-instance $L_{x,y}$ with $y_i - x_i < 2$ for all i. Lemma 3.2 gives us an efficient algorithm to find a solution in this final instance, which uses at most $O(\sum_{i=1}^{k}(y_i - x_i)) = O(k)$ queries. So we have proved the following theorem.

Theorem 3.4. *Suppose that there exists an inner algorithm that makes at most q queries. Then a solution to the Tarski problem can be found by making $O(q \cdot k \cdot \log n + k)$ queries.*

4 THE INNER ALGORITHM

We now describe an inner algorithm for three dimensions that makes $O(\log n)$ queries. Throughout this section we assume that the inner algorithm has been invoked on a sub-instance $L_{u,d}$ and principle slice s, and without loss of generality we assume that $s = (*, *, s_3)$.

Down set witnesses. Like the outer algorithm, the inner algorithm will also focus on smaller and smaller sub-instances that are guaranteed to contain a solution by an invariant, but now the invariant is more complex. To define the invariant, we first introduce the concept of a *down set witness* and an *up set witness*. The points d and b in the second example in Figure 3 give an example of a down set witness. Note that the following properties are satisfied.

- f weakly increases at d and b in dimension 3.
- d and b have the same coordinate in dimension 2.
- d weakly increases in dimension 1 while b weakly decreases in dimension 1.

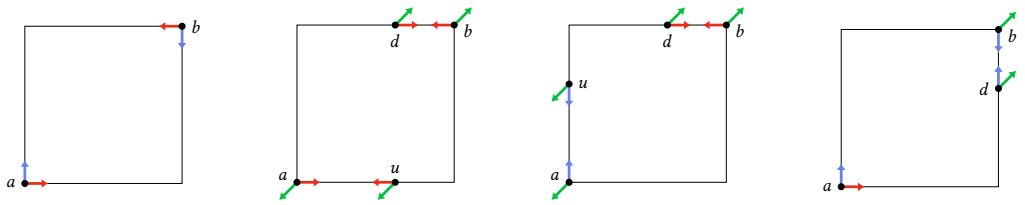

Fig. 3. Four example sub-instances that satisfy the inner algorithm invariant.

We also allow down set witnesses like those given by d and b in the fourth example of Figure 3 that satisfy the same properties with dimensions 1 and 2 swapped. Thus, the formal definition of a down set witness abstracts over dimensions 1 and 2.

Definition 4.1 (Down set witness). A down set witness is a pair of points (d, b) with $d, b \in L_s$ such that both of the following are satisfied.

- $d_3 \leq f(d)_3$ and $b_3 \leq f(b)_3$.
- $\exists\, i, j \in \{1, 2\}$ with $i \neq j$ s.t. $d_i = b_i$ and $d_j \leq b_j$, while $d_j \leq f(d)_j$ and $f(b)_j \leq b_j$.

If (d, b) is a down set witness and $d_2 = b_2$, then we call (d, b) a *top-boundary* witness, while if $d_1 = b_1$, then we call (d, b) a *right-boundary* witness.

The following lemma states that if we have a down set witness (d, b), then between d and b we can find either a solution that can be returned by the inner algorithm (cases 1 and 2 of the lemma), or a point that is in the down set of the slice s (case 3 of the lemma).

Informally, the proof for a top-boundary witness (d, b) uses the fact that d and b point towards each other in dimension 1 to argue that there must be a fixed point p (or an order preservation violation) of the one-dimensional slice between d and b. Then, it is shown that either this point is in $\mathrm{Up}(f)$, and so is a solution for the inner algorithm, or it is in $\mathrm{Down}(f_s)$, or that p violates order preservation with d or b.

Lemma 4.2. *If (d, b) is a down set witness, then one of the following exists.*

(1) A point c satisfying $d \preceq c \preceq b$ such that $c \in \mathrm{Up}(f)$.
(2) Two points x, y satisfying $d \preceq x \preceq y \preceq b$ that witness order preservation violation of f.
(3) A point c satisfying $d \preceq c \preceq b$ such that $c \in \mathrm{Down}(f_s)$.

Proof. Observe that, since $d_i = b_i$, we have that d and b both lie in the same one-dimensional slice. Moreover, the fact that $d_j \leq f(d)_j$ implies that d lies in the up set of this slice, while $f(b)_j \leq b_j$ implies that b lies in the down set of this slice. Hence, we can apply Lemma 3.2 to either find a fixed point p of this one-dimensional slice, or a violation of order preservation. In the latter case, we are done since case two of the lemma will have been fulfilled, so we will proceed assuming that p exists.

There are now two cases to consider.

- **Case 1: $\mathbf{p_i \leq f(p)_i}$.** There are two sub-cases.
  - **Sub-case 1: $\mathbf{p_3 \leq f(p)_3}$.** Note that $p_j = f(p)_j$, since p is the fixed point of the one-dimensional slice. Hence $p \in \mathrm{Up}(f)$, and so the first case of the lemma has been fulfilled.

- **Sub-case 2:** $\mathbf{p_3 > f(p)_3}$. Now we have

$$f(p)_3 < p_3 = d_3 \leq f(d)_3,$$

where the equality holds because d and p both lie in s, and the final inequality holds by the requirements of a down set witness. Therefore we have $d \preceq p$ but $f(d) \not\preceq f(p)$, and so d and p witness a violation of the order preservation of f, and so the second case of the lemma is satisfied.

- **Case 2:** $p_i > f(p)_i$. Note that $p_j = f(p)_j$ since p is a fixed point of the one-dimensional slice. Hence p is in the down set of the slice s, and so the third condition of the lemma has been fulfilled.

□

Up set witnesses. An up set witness is simply a down set witness in which all inequalities have been flipped. The second and third diagrams in Figure 3 show the two possible configurations of an up set witness (a, u). Note that for up set witnesses, dimension 3 is now required to weakly decrease.

Definition 4.3 (Up set witness). An up set witness is a pair of points (a, u) with $a, u \in L_s$ such that both of the following are satisfied.

- $a_3 \geq f(a)_3$ and $u_3 \geq f(u)_3$.
- $\exists\, i, j \in \{1, 2\}$ with $i \neq j$ s.t. $a_i = u_i$ and $u_j \geq a_j$, while $u_j \geq f(u)_j$ and $f(a)_j \geq a_j$.

We say that an up set witness (a, b) is a *left-boundary* witness if $a_1 = b_1$, while we call it a *bottom-boundary* witness if $a_2 = b_2$.

The following lemma is the analogue of Lemma 4.2 for up set witnesses. The proof simply flips all inequalities in the proof of Lemma 4.2.

Lemma 4.4. *If (a, u) is an up set witness, then one of the following exists.*

(1) A point c satisfying $a \preceq c \preceq u$ such that $c \in \text{Down}(f)$.
(2) Two points x, y satisfying $a \preceq x \preceq y \preceq u$ that witness order preservation violation of f.
(3) A point c satisfying $a \preceq c \preceq u$ such that $c \in \text{Up}(f_s)$.

Proof. This proof is exactly the same as the proof of Lemma 4.2, but all inequalities have been flipped. We include it only for the sake of completeness.

Observe that, since $a_i = u_i$, we have that a and u both lie in the same one-dimensional slice. Moreover, the fact that $u_j \geq f(u)_j$ implies that u lies in the down set of this slice, while $f(a)_j \geq a_j$ implies that a lies in the up set of this slice. Hence, we can apply Lemma 3.2 to either find a fixed point p of this one-dimensional slice, or a violation of order preservation. In the latter case, we are done since case two of the lemma will have been fulfilled, so we will proceed assuming that p exists.

There are now two cases to consider.

- **Case 1:** $\mathbf{p_i \geq f(p)_i}$. There are two sub-cases.
  - **Sub-case 1:** $\mathbf{p_3 \geq f(p)_3}$. Note that $p_j = f(p)_j$, since p is the fixed point of the one-dimensional slice. Hence $p \in \text{Down}(f)$, and so the first case of the lemma has been fulfilled.
  - **Sub-case 2:** $\mathbf{p_3 < f(p)_3}$. Now we have

$$f(p)_3 > p_3 = a_3 \geq f(a)_3,$$

where the equality holds because a and p both lie in s, and the final inequality holds by the requirements of an up set witness. Therefore we have $a \preceq p$ but $f(a) \not\preceq f(p)$, and so a and p witness a violation of the order preservation of f, and so the second case of the lemma is satisfied.

- **Case 2: $\mathbf{p_i} < \mathbf{f(p)_i}$.** Note that $p_j = f(p)_j$ since p is a fixed point of the one-dimensional slice. Hence p is the up set of the slice s, and so the third condition of the lemma has been fulfilled.

□

The invariant. At each step of the inner algorithm, we will have a sub-instance $L_{a,b}$ that satisfies the following invariant.

Definition 4.5 (Inner algorithm invariant). The instance $L_{a,b}$ satisfies the invariant if

- Either $a \in \mathrm{Up}(f_s)$ or there is a known up set witness (a, u) with $u \preceq b$.
- Either $b \in \mathrm{Down}(f_s)$ or there is a known down set witness (d, b) with $a \preceq d$.

If we have both an up set witness and a down set witness then we also require that $u \preceq d$.

Figure 3 gives four example instances that satisfy the invariant. Note that there are actually nine possible configurations, since the first point of the invariant can be satisfied either by a point in the up set, a left-boundary up set witness, or a bottom-boundary up set witness, and the second point of the invariant likewise has three possible configurations.

The following lemma shows that, if the invariant is satisfied, then the sub-instance $L_{a,b}$ contains a solution that can be returned by the inner algorithm. The proof invokes Lemmas 4.2 and 4.4 to either immediately find a solution for the inner algorithm, or find two points $x \preceq y$ in the sub instance where x is in the up set and y is in the down set. The latter case allows us to invoke Lemma 3.2 to argue that the sub-instance contains a fixed point p of the slice s. If p weakly increases in the third dimension, then $p \in \mathrm{Up}(f)$, while if p decreases in the third dimension then $p \in \mathrm{Down}(f)$.

Lemma 4.6. *If $L_{a,b}$ satisfies the invariant then one of the following exists.*

- *A point $x \in L_{a,b}$ such that $x \in \mathrm{Up}(f)$ or $x \in \mathrm{Down}(f)$.*
- *Two points $x, y \in L_{a,b}$ that witness a violation of the order preservation of f.*

Proof. We can find a point $x \in L_{a,b}$ that satisfies $x \in \mathrm{Up}(f_s)$ in the following way. By the invariant, either a already satisfies this condition, or we can apply Lemma 4.4 to obtain one of the three possible cases from that lemma. Cases one and two immediately fulfill the requirements of this lemma, and so we are done immediately in those cases, while the third case gives us the point x.

Symmetrically, we can find a point $y \in L_{a,b}$ that satisfies $y \in \mathrm{Down}(f_s)$, since either b is such a point, or we can invoke Lemma 4.2 to either immediately fulfil the requirements of this lemma, or produce the point y.

Note further that we have $x \preceq y$. If either $a = x$ or $b = y$ then this holds due to the promises given by the invariant. When we have both an up and down set witness we have $x \preceq u \preceq d \preceq y$, where the first and third inequalities are come from Lemmas 4.4 and 4.2, while $u \preceq d$ is promised by the invariant.

Hence we can apply Lemma 3.2 to the sub-instance $L_{x,y} \subseteq L_{a,b}$, which will either give us a violation of order preservation, which will immediately satisfy the second condition of this lemma, or a fixed point $p \in L_{x,y}$ of the slice s. We now do a case analysis on the third dimension.

- If $p_3 \leq f(p)_3$, then $p \in \mathrm{Up}(f)$ since $p_1 = f(p_1)$ and $p_2 = f(p_2)$.
- If $p_3 > f(p)_3$, then $p \in \mathrm{Down}(f)$ since $p_1 = f(p_1)$ and $p_2 = f(p_2)$.

Hence, in either case the first condition of this lemma is satisfied. □

A special case. There is a special case that we will encounter in the inner algorithm that requires more effort to deal with. One example of this case is shown in Case 3.a.ii of Figure 6. Here we have a point p on the right-hand boundary of the instance that satisfies $p_1 < f(p)_1$, meaning that f moves p outside of the instance. If $b \in \text{Down}(f)$, or if there is a top-boundary down set witness, then it is straightforward to show that p and b violate order preservation.

However, if we have a right-boundary down set witness (d, b) with $p \preceq d$ then we need to do further work[2]. Note that the properties of a down set witness ensure that $d_2 \leq f(d)_2$ and $d_3 \leq f(d)_3$. But there are two possibilities for dimension 1. If $d_1 \leq f(d)_1$ then $d \in \text{Up}(f)$ and it can be returned by the inner algorithm. On the other hand, if $d_1 > f(d)_1$, then we can show that p and d violate order preservation. We prove this formally in the following lemma.

Lemma 4.7. *Let $L_{a,b}$ be a sub-instance that satisfies the invariant, and let p be a point satisfying $a \preceq p \preceq b$ that also satisfies one of the following conditions.*

(1) $p_1 = b_1$ and $p_1 < f(p)_1$.
(2) $p_2 = b_2$ and $p_2 < f(p)_2$.
(3) $p_1 = a_1$ and $p_1 > f(p)_1$.
(4) $p_2 = a_2$ and $p_2 > f(p)_2$.

Suppose further that, if there exists a down-set witness (d, b) then $p \preceq d$, and if there exists an up-set witness (a, u) then $u \preceq p$. Then there exists a solution for the inner algorithm that can be found using constantly many queries.

Proof. We shall begin by providing a proof for the case where $p_1 = b_1$ and $p_1 < f(p)_1$. The other three cases will be proved symmetrically. There are two cases to consider.

(1) If $b \in \text{Down}(f_s)$, or if there is a top-boundary down set witness (d, b), then we have that $f(b)_1 \leq b_1$. Therefore we have
$$f(b)_1 \leq b_1 = p_1 < f(p)_1,$$
so $p \preceq b$ but $f(p) \not\preceq f(b)$, and therefore we have a violation of order preservation.
(2) If there is a right-boundary down set witness (d, b), then note that by the properties of a down set witness we have $d_2 \leq f(d)_2$ and $d_3 \leq f(d)_3$. There are now two cases to consider.
   (a) If $d_1 \leq f(d)_1$, then $d \in \text{Up}(f)$, and so d can be returned by the inner algorithm.
   (b) If $d_1 > f(d)_1$ then we have
   $$f(d)_1 < d_1 = p_1 < f(p)_1,$$
   so we have $p \preceq d$ but $f(p) \not\preceq f(d)$, and therefore we have a violation of order preservation.

The other three cases can be proved by the same reasoning.

- The case where $p_2 = b_2$ and $p_2 < f(p)_2$ can be proved by exchanging dimensions 1 and 2.
- The case where $p_1 = a_1$ and $p_1 > f(p)_1$ can be proved by flipping all inequalities.
- The case where $p_2 = a_2$ and $p_2 > f(p)_2$ can be proved by exchanging dimensions 1 and 2, and also flipping all inequalities.

□

[2]The case where $p \succ d$ will never occur in our algorithm, so we can ignore it.

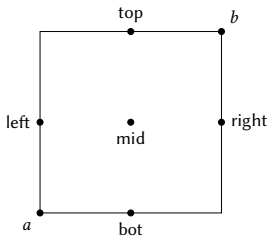

Fig. 4. The five points used by the inner algorithm.

Initialization. The input to the algorithm is a sub-instance $L_{x,y}$, and recall that we have fixed the principle slice $s = (*, *, s_3)$. The initial values for a and b are determined as follows. For each dimension i we have $a_i = s_3$ if $i = 3$, and $a_i = x_i$ otherwise, and we have $b_i = s_3$ if $i = 3$, and $b_i = y_i$ otherwise. That is, a and b are the projections of x and y onto s.

The following lemma states that either a and b satisfy the invariant, or that we can easily find a violation of order preservation.

Lemma 4.8. *Either $L_{a,b}$ satisfies the invariant, or there is violation of order preservation between a and x, or between b and y.*

Proof. We will show that either $a \in \mathrm{Up}(f_s)$ and $b \in \mathrm{Down}(f_s)$, or that a violation of order preservation can be found.

We start by showing that either we have a violation of order preservation, or we have that $a \in \mathrm{Up}(f_s)$. To check this, we only need to inspect dimensions $i \in \{1, 2\}$. Note that if $f(a)_i < a_i$ for some i, then

$$f(a)_i < a_i = x_i \leq f(x)_i,$$

where the equality holds from the definition of a, and the final inequality holds since $x \in \mathrm{Up}(f)$ is a requirement for calling the inner algorithm. Thus, if $f(a)_i < a_i$ for some dimension $i \in \{1, 2\}$ then we have $x \preceq a$ and $f(x) \not\preceq f(a)$, and so we have that x and a witness a violation of the order preservation of f. Otherwise, we have $f(a)_i \leq a_i$ for all $i \in \{1, 2\}$, and therefore $a \in \mathrm{Up}(f_s)$.

We can apply the same reasoning symmetrically to b and y. If $b_i > f(b)_i$ for some $i \in \{1, 2\}$ then

$$f(b)_i > b_i = y_i \geq f(y)_i,$$

where the equality holds from the definition of b and the final inequality holds since $y \in \mathrm{Down}(f)$ is a requirement for calling the inner algorithm. Thus we would have $b \preceq y$ and $f(b) \not\preceq f(y)$, and so either b and y witness a violation of the order preservation of f, or $b \in \mathrm{Down}(f_s)$.

At this stage we have either satisfied the second or third conditions of the lemma, or we have that $a \in \mathrm{Up}(f_s)$ and $b \in \mathrm{Down}(f_s)$ and so the invariant on $L_{a,b}$ is satisfied. □

The algorithm. Now suppose that we have an instance $L_{a,b}$ that satisfies the invariant. We now describe how to execute one iteration of the algorithm, which will make constantly many queries, and then either find a violation of order preservation, or find a new instance whose size is at most half the size of the $L_{a,b}$.

We begin by defining some important points. We define $\text{mid} = \lfloor (a+b)/2 \rfloor$ to be the *midpoint* of the instance, and we define the following points, which are shown in Figure 4:

$$\text{bot} = (\lfloor (a_1 + b_1)/2 \rfloor, a_2), \qquad \text{left} = (a_1, \lfloor (a_2 + b_2)/2 \rfloor),$$
$$\text{top} = (\lfloor (a_1 + b_1)/2 \rfloor, b_2), \qquad \text{right} = (b_1, \lfloor (a_2 + b_2)/2 \rfloor).$$

Since all of these points lie in the slice s, we do not explicitly specify a third coordinate for these points, or for the other points that we build in the following steps.

Step 1: Fixing the up and down set witnesses. Suppose that $L_{a,b}$ satisfies the invariant with a top-boundary down set witness (d, b), We would like to ensure that $\text{top} \preceq d$, since otherwise if we cut the instance in half in a later step, the witness may no longer be within the sub-instance. For the same reason, we would like to ensure that (d, b) satisfies $\text{right} \preceq d$ for a right-boundary down set witness, that (a, u) satisfies $u \preceq \text{left}$ for a left-boundary up set witness, and that (a, u) satisfies $u \preceq \text{bot}$ for a bottom-boundary up set witness. By the end of Step 1 we will have either found a violation of order preservation, moved into the next iteration with a sub-instance of half the size, or all inequalities above will hold.

Step 1 consists of the following procedure. The procedure should be read alongside Figure 5, which gives a diagram for every case presented below.

(1) If (d, b) is a top-boundary down set witness and $\text{top} \preceq d$ then there is no need to do anything. On the other hand, if $d \prec \text{top}$ we use the following procedure.
   (a) We first check if $\text{top}_3 > f(\text{top})_3$. If this is the case, then since the invariant ensures that $d_3 \leq f(d)_3$ we have $f(d)_3 \geq d_3 = \text{top}_3 > f(\text{top})_3$ so $d \preceq \text{top}$ but $f(d) \not\preceq f(\text{top})$, and an order preservation violation has been found and the inner algorithm terminates.
   (b) We next check the whether $\text{top}_1 > f(\text{top})_1$. In this case, we can use (d, top) as a down set witness for the sub-instance $L_{a,\text{top}}$, where we observe that top satisfies the requirements since $\text{top}_3 \leq f(\text{top})_3$ and $\text{top}_1 > f(\text{top})_1$. Hence, $L_{a,\text{top}}$ satisfies the invariant (if $L_{a,b}$ also has an up set witness (a, u) then note that $u \preceq d$ continues to hold), and so the algorithm moves into the next iteration with the sub-instance $L_{a,\text{top}}$.
   (c) In this final case we have $\text{top}_3 \leq f(\text{top})_3$ and $\text{top}_1 \leq f(\text{top})_1$. Therefore (top, b) is a valid down set witness for $L_{a,b}$ (if $L_{a,b}$ also has an up set witness (a, u) then note that $u \preceq d \prec \text{top}$). So we can replace (d, b) with (top, b) and continue, noting that our down set witness now satisfies the required inequality.

(2) If (d, b) is a right-boundary down set witness and $\text{right} \preceq d$ then there is no need to do anything. On the other hand, if $d \prec \text{right}$ then we use the same procedure as case 1, where dimensions 1 and 2 are exchanged and the point top is replaced by the point right.

(3) If (a, u) is a bottom-boundary up set witness and $u \preceq \text{bot}$ then there is no need to do anything. On the other hand, if $\text{bot} \prec u$ then we use the following procedure, which is the same as the procedure from case 1, where all inequalities have been flipped.
   (a) We first check if $\text{bot}_3 < f(\text{bot})_3$. If this is the case, then since the invariant ensures that $u_3 \geq f(u)_3$ we have $f(u)_3 \leq u_3 = \text{bot}_3 < f(\text{bot})_3$ so $u \succeq \text{bot}$ but $f(u) \not\succeq f(\text{bot})$, and an order preservation violation has been found and the inner algorithm terminates.
   (b) We next check the whether $\text{bot}_1 < f(\text{bot})_1$. In this case, we can use (bot, u) as an up set witness for the sub-instance $L_{\text{bot},b}$, where we observe that bot satisfies the requirements since $\text{bot}_3 \geq f(\text{bot})_3$ and $\text{bot}_1 < f(\text{bot})_1$.

Hence, $L_{\mathsf{bot},b}$ satisfies the invariant (if $L_{a,b}$ also has a down set witness (d,b) then note that $u \preceq d$ continues to hold), and so the algorithm moves into the next iteration with the sub-instance $L_{\mathsf{bot},b}$.

(c) In this final case we have $\mathsf{bot}_3 \geq f(\mathsf{bot})_3$ and $\mathsf{bot}_1 \geq f(\mathsf{bot})_1$. Therefore (a, bot) is a valid up set witness for $L_{a,b}$ (if $L_{a,b}$ also has a down set witness (d,b) then we note that $\mathsf{bot} \preceq u \preceq d$). So we can replace (a,u) with (a, bot), noting that our up set witness now satisfies the required inequality.

(4) If (a,u) is a left-boundary up set witness and $u \preceq \mathsf{left}$ then there is no need to do anything. On the other hand, if $u \succ \mathsf{left}$ then we use the same procedure as case 3, where dimensions 1 and 2 are exchanged and the point left is replaced by the point bot.

Step 2: Find a smaller sub-instance. If Step 1 of the algorithm did not already move us into the next iteration of the algorithm with a smaller instance, we apply Step 2. This step performs a case analysis on the point mid. The following procedure should be read in conjunction with Figure 6, which provides a diagram for every case.

(1) Check if $\mathsf{mid}_1 \preceq f(\mathsf{mid})_1$ and $\mathsf{mid}_2 \preceq f(\mathsf{mid})_2$. If this is the case then $\mathsf{mid} \in \mathrm{Up}(f_s)$, and so we can move to the next iteration of the algorithm with the sub-instance $L_{\mathsf{mid},b}$. Note that if $L_{a,b}$ has a down-set witness (d,b), then Step 1 of the algorithm has ensured that $\mathsf{mid} \preceq d$, and so (d,b) is also a valid down-set witness for $L_{\mathsf{mid},b}$.

(2) Check if $\mathsf{mid}_1 \succeq f(\mathsf{mid})_1$ and $\mathsf{mid}_2 \succeq f(\mathsf{mid})_2$. If this is the case then $\mathsf{mid} \in \mathrm{Down}(f_s)$, and so we can move to the next iteration of the algorithm with the sub-instance $L_{a,\mathsf{mid}}$. Note that if $L_{a,b}$ has an up-set witness (a,u), then Step 1 of the algorithm has ensured that $u \preceq \mathsf{mid}$, and so (a,u) is also a valid down-set witness for $L_{a,\mathsf{mid}}$.

(3) Check if $\mathsf{mid}_1 \preceq f(\mathsf{mid})_1$ and $\mathsf{mid}_2 \succ f(\mathsf{mid})_2$. If so, we use the following procedure.

(a) Check if $\mathsf{mid}_3 \preceq f(\mathsf{mid})_3$. If so, do the following.

(i) Check if $\mathsf{right}_3 > f(\mathsf{right})_3$. If this holds then we have $f(\mathsf{mid})_3 \geq \mathsf{mid}_3 = \mathsf{right}_3 > f(\mathsf{right})_3$, meaning that $\mathsf{mid} \preceq \mathsf{right}$ but $f(\mathsf{mid}) \not\preceq f(\mathsf{right})$. Thus we have found a violation of order preservation and the algorithm terminates.

(ii) Check if $\mathsf{right}_1 < f(\mathsf{right})_1$. If this holds then we use Lemma 4.7 (with $p := \mathsf{right}$) to find a solution that can be returned by the inner algorithm.

(iii) If we reach this case then we have $\mathsf{mid}_3 \preceq f(\mathsf{mid})_3$ and $\mathsf{right}_3 \preceq f(\mathsf{right})_3$, while we also have $\mathsf{mid}_1 \preceq f(\mathsf{mid})_1$ and $\mathsf{right}_1 \succeq f(\mathsf{right})_1$. Thus $(\mathsf{mid}, \mathsf{right})$ is a valid down set witness for the instance $L_{a,\mathsf{right}}$. Note that if $L_{a,b}$ has an up set witness (a,u), then Step 1 ensures that $u \preceq \mathsf{mid}$, and so $L_{a,\mathsf{right}}$ satisfies the invariant. So the algorithm moves to the next iteration with sub-instance $L_{a,\mathsf{right}}$.

(b) In this case we have $\mathsf{mid}_3 > f(\mathsf{mid})_3$. The following three steps are symmetric to those used in Case 3.a, but with all inequalities flipped, dimension 1 substituted for dimension 2, the point bot substituted for right, and the point a substituted for b.

(i) Check if $\mathsf{bot}_3 > f(\mathsf{bot})_3$. If this holds then we have $f(\mathsf{mid})_3 \preceq \mathsf{mid}_3 = \mathsf{bot}_3 < f(\mathsf{bot})_3$, meaning that $\mathsf{mid} \succeq \mathsf{bot}$ but $f(\mathsf{mid}) \not\succeq f(\mathsf{bot})$. Thus we have found a violation of order preservation and the algorithm terminates.

(ii) Check if $\mathsf{bot}_2 > f(\mathsf{bot})_2$. If this holds then we can use Lemma 4.7 (with $p := \mathsf{bot}$) to find a solution that can be returned by the inner algorithm.

(iii) If we reach this case then we have $\mathsf{mid}_3 \succeq f(\mathsf{mid})_3$ and $\mathsf{bot}_3 \succeq f(\mathsf{bot})_3$, while we also have $\mathsf{mid}_2 \succeq f(\mathsf{mid})_2$ and $\mathsf{bot}_2 \preceq f(\mathsf{bot})_2$. Thus $(\mathsf{bot}, \mathsf{mid})$ is a valid up set witness for the instance $L_{\mathsf{bot},b}$. Note that if $L_{a,b}$ has a down set witness (d,b), then Step 1 of the algorithm ensures that $d \succeq \mathsf{mid}$, and so $L_{\mathsf{bot},b}$ satisfies the invariant. The algorithm will therefore move to the next iteration with the sub-instance $L_{\mathsf{bot},b}$.

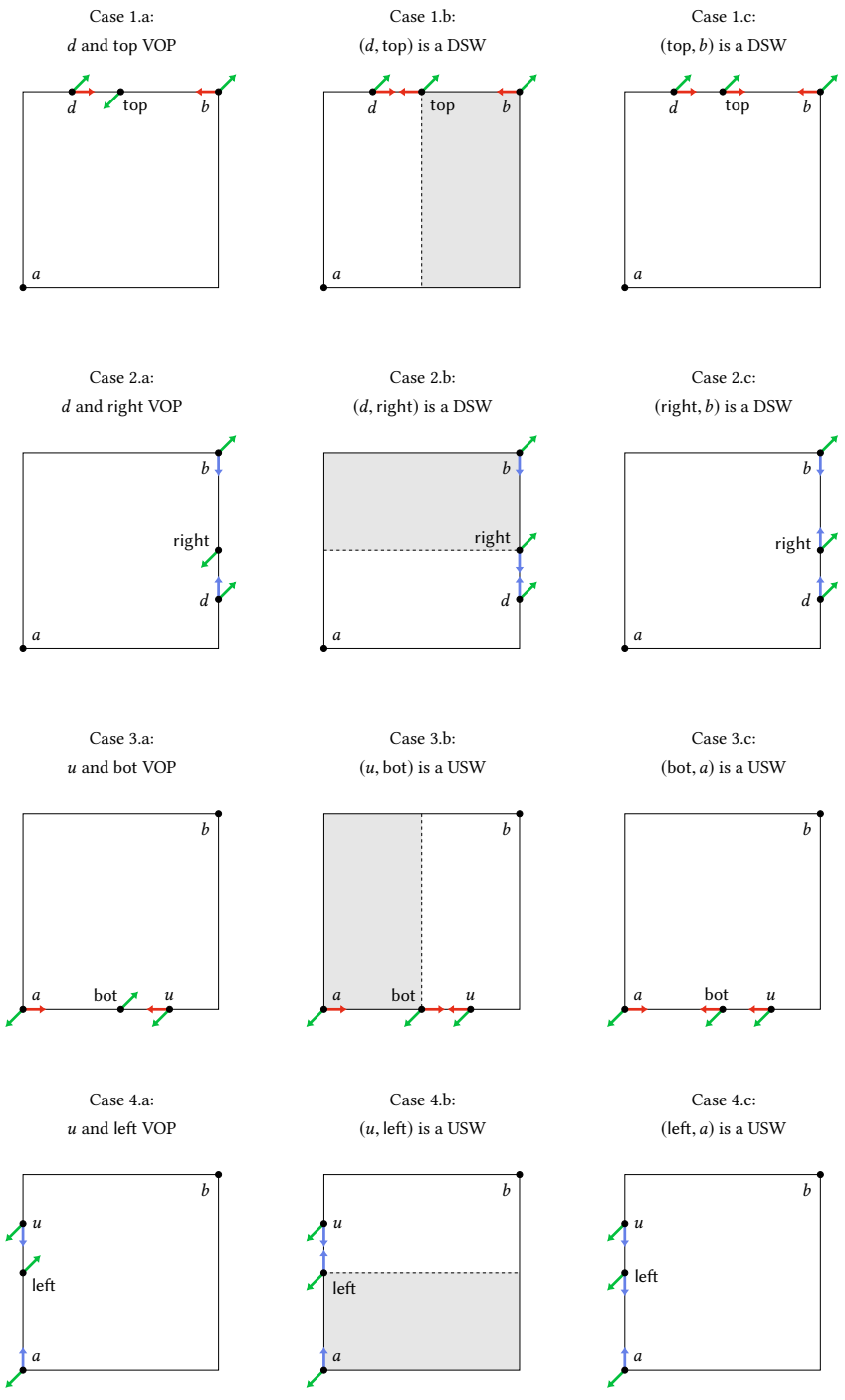

Fig. 5. All cases used in Step 1 of the algorithm. In the labels, VOP is short for "violate order preservation", DSW is short for "down set witness", and USW is short for "up set witness".

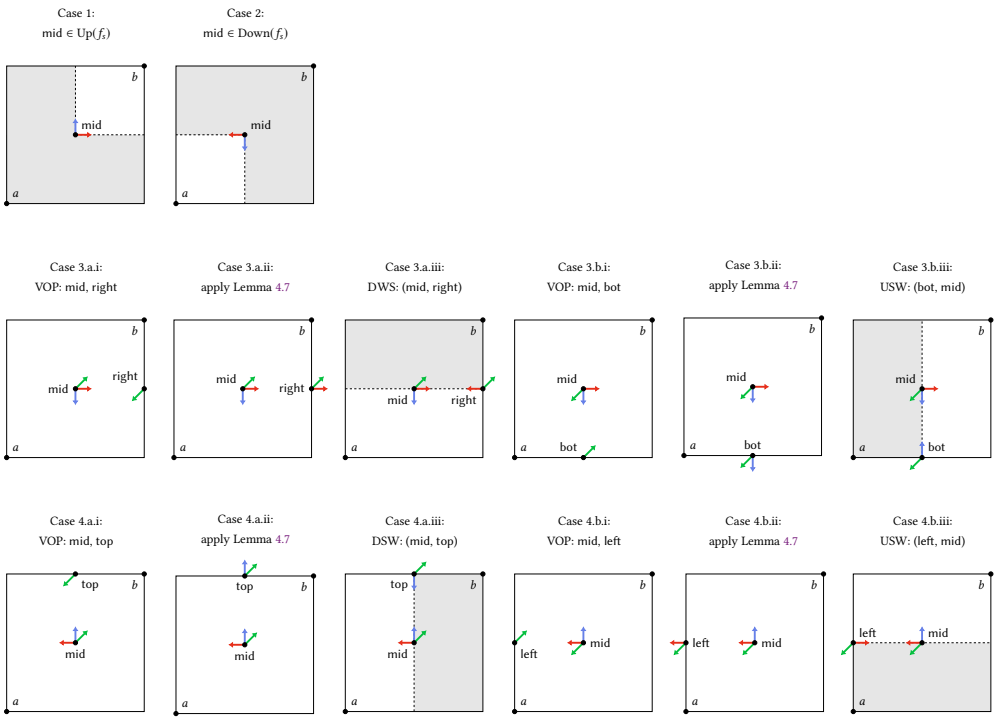

Fig. 6. All cases used in Step 2 of the algorithm. In the labels, VOP is short for "violate order preservation", DSW is short for "down set witness", and USW is short for "up set witness".

(4) In this final case we have $\text{mid}_1 > f(\text{mid})_1$ and $\text{mid}_2 \leq f(\text{mid})_2$. Here we follow the same procedure as Case 3, but with dimensions 1 and 2 exchanged, every instance of the point right replaced with top, and every instance of bot replaced with left.

The terminal phase of the algorithm. The algorithm can continue so long as $b_1 \geq a_1 + 2$ and $b_2 \geq a_2 + 2$, since this ensures that all cases will cut the width of one of the dimensions in half. The algorithm terminates once we have both $b_1 \leq a_1 + 1$ *and* $b_2 \leq a_2 + 1$. However, once there exists only one dimension i for which $b_i \leq a_i + 1$, we must be careful, since now the midpoint lies on the boundary of the instance, and some of the cases of the algorithm may not rule out anything. We deal with this scenario separately.

There are two distinct cases that we must deal with. The first case is a *width-one instance*, in which $b_i = a_i + 1$ for some index $i \in \{1, 2\}$, and $b_j > a_j + 1$ for the index $j \in \{1, 2\}$ with $j \neq i$, meaning that the width of the shortest dimension is exactly one. These instances are problematic because the midpoint mid will now lie on the boundary of the instance, and due to this, it is possible that the algorithm may be unable to proceed.

We must also deal with *width-zero instances*, in which $b_i = a_i$ for some index $i \in \{1, 2\}$, and $b_j > a_j + 1$ for the index $j \in \{1, 2\}$ with $j \neq i$. These are one-dimensional subinstances, and once again it is possible for the algorithm to be unable to proceed.

We will use special procedures for width-one and width-zero instances, which we outline below.

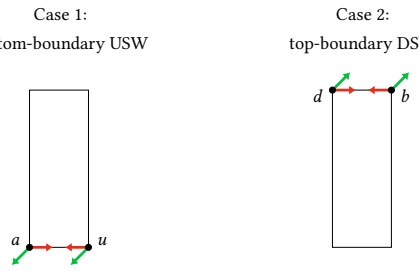

Fig. 7. The two cases that trigger the preprocessing step for width-one instances.

Width-one instances. In the presentation below we will assume that the index $i = 1$, meaning that $b_1 = a_1 + 1$ (and hence the left-right width of the instance is one). The case for $i = 2$ is symmetric.

When the algorithm is presented with a width-one instance, it first performs some preprocessing to ensure that there is no bottom-boundary up set witnesses, or top-boundary down set witness. The preprocessing considers the following two cases, which are shown in Figure 7.

(1) If the instance has a bottom-boundary up set witness (a, u), then note that a and u are directly adjacent in dimension 1, and so Lemma 4.4 implies that we can either return a or u as a solution for the inner algorithm, or that $a \in \text{Up}(f_s)$, or $u \in \text{Up}(f_s)$.
   (a) If $a \in \text{Up}(f_s)$, then the instance $L_{a,b}$ continues to satisfy the invariant if we delete the up set witness.
   (b) If $u \in \text{Up}(f_s)$, then the width-zero instance $L_{u,b}$ satisfies the invariant, where we note that if $L_{a,b}$ has a down set witness (d, b), then since $u \preceq d$, we have that (d, b) is also a valid down set witness for $L_{u,b}$.

(2) If the instance has a top-boundary down set witness (d, b), then note that d and b are directly adjacent in dimension 1, and so Lemma 4.2 implies that we can either return d or b as a solution for the inner algorithm, or that $d \in \text{Down}(f_s)$, or $b \in \text{Down}(f_s)$.
   (a) If $b \in \text{Down}(f_s)$, then the instance $L_{a,b}$ continues to satisfy the invariant if we delete the down set witness.
   (b) If $d \in \text{Down}(f_s)$, then the width-zero instance $L_{a,d}$ satisfies the invariant, where we note that if $L_{a,b}$ has an up set witness (a, u), then since $u \preceq d$, we have that (a, u) is also a valid up set witness for $L_{a,d}$.

With the preprocessing completed, the algorithm uses two separate runs of Steps 1 and 2, which each use a different midpoint. In the first run we use midone $= \lfloor (a + b)/2 \rfloor$ as normal, while in the second run we use midtwo $= \lceil (a + b)/2 \rceil$ as the midpoint, and we also change the definitions of bot, top, left, and right to round up instead of down. If either of the two runs decrease the size of the instance, then we move to the next iteration on the smaller instance, where the reasoning given in Steps 1 and 2 ensures that the instance continues to satisfy the invariant.

However, it could be the case that both runs do not decrease the size of the instance. Due to the preprocessing, if Step 1 attempts to recurse on a smaller sub-instance then it must succeed, since the only problematic cases are Case 1.b and Case 3.b, which both depend on the existence of a top or bottom-boundary witness, and the preprocessing ensures that these cannot exist.

On the other hand, Step 2 can fail to make progress in both runs. For the case where $i = 1$, this can only occur if Case 3.b.iii of Step 2 triggered for the run with midone and Case 4.a.iii triggered for the run with midtwo. But we can argue that in this case a solution to the inner algorithm is easy to find.

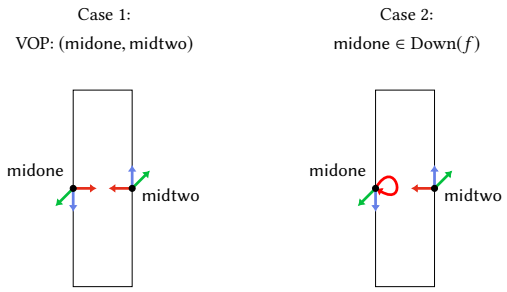

Fig. 8. The two cases that are considered for width-one instances, when both runs of the algorithm fail to make progress. In the left instance, $f(\text{midone})$ strictly increases in dimension 1, while in the right instance $f(\text{midone})$ does not move in dimension one, which we indicate with the self loop.

Note that Case 3.b.iii can only trigger for midone when $\text{midone}_1 \leq f(\text{midone})_1$, while Case 4.a.iii can only trigger for midtwo when $f(\text{midtwo})_1 < \text{midtwo}$. Both of the following cases are shown in Figure 8.

(1) If $\text{midone}_1 < f(\text{midone})_1$ then we have

$$f(\text{midtwo})_1 \leq \text{midtwo}_1 - 1 = \text{midone}_1 < f(\text{midone})_1,$$

so we have $\text{midone} \preceq \text{midtwo}$ but $f(\text{midone}) \not\preceq f(\text{midtwo})$, meaning that midone and midtwo witness a violation of order preservation.

(2) If $\text{midone}_1 = f(\text{midone})_1$, then note that Case 3.b.iii ensures that $\text{midone}_2 \geq f(\text{midone})_2$ and $\text{midone}_3 \geq f(\text{midone})_3$. Therefore midone is in $\text{Down}(f)$, so can be returned by the inner algorithm.

So in both cases a solution to the inner algorithm has been found.

Width-zero instances. We again describe the procedure for the case where $i = 1$, meaning that the instance has width zero in the left-right dimension. The case where $i = 2$ is symmetric.

The algorithm begins by performing a preprocessing step that removes any top-boundary down set witnesses or bottom-boundary up set witnesses. If there is a bottom-boundary up set witness (a, u) then note that $a = u$, and therefore Lemma 4.4 implies that either a is a solution that can be returned by the inner algorithm, or that $a \in \text{Up}(f_s)$. Likewise, if there is a top-boundary down set witness (d, b), then $d = b$, and Lemma 4.2 implies that either d can be returned by the inner algorithm, or $d \in \text{Down}(f_s)$. Thus, the preprocessing step can either find a solution for the inner algorithm, or produce an instance that satisfies the invariant that has no top-boundary down set witness and no bottom-boundary up set witness.

Once the preprocessing has taken place, the algorithm proceeds through Step 1 and Step 2 as normal. If those steps make progress, then we continue on the smaller width-zero instance. If they do not make progress, then we will show that the inner algorithm can terminate after making at most $O(\log n)$ further queries.

We first observe that the only cases of Step 1 that would fail to make progress are Case 1.b and Case 3.b, but neither of those cases can trigger because the preprocessing step ensures that there is no bottom-boundary up set witness or top-boundary down set witness.

On the other hand, Case 3.b.iii and Case 4.a.iii of Step 2 can fail to make progress. We show how, in each of these cases, a solution for the inner algorithm can be found by making at most $O(\log n)$ extra queries. All of the following cases are depicted in Figure 9.

(1) If Case 3.b.iii is triggered, then note that $\mathsf{mid}_1 \leq f(\mathsf{mid})_1$. There are two cases to consider.
   (a) If $\mathsf{mid}_1 = f(\mathsf{mid})_1$, then since $\mathsf{mid}_2 \geq f(\mathsf{mid})_2$ and $\mathsf{mid}_3 \geq f(\mathsf{mid}_3)$, we have that $\mathsf{mid} \in \mathrm{Down}(f)$, meaning that mid can be returned by the inner algorithm.
   (b) If $\mathsf{mid}_1 < f(\mathsf{mid})_1$ then we argue that an order preservation violation can be found using at most $O(\log n)$ queries.
      (i) If $b \in \mathrm{Down}(f_s)$, then we have
      $$f(b)_1 \leq b_1 = \mathsf{mid}_1 < f(\mathsf{mid})_1,$$
      meaning that $\mathsf{mid} \preceq b$, but $f(\mathsf{mid}) \not\preceq f(b)$, and so mid and b violate order preservation.
      (ii) If instead there is a down set witness (d, b), then note that due to the preprocessing, it must be a right-boundary down set witness, and due to Step 1, we must have $\mathsf{mid} \preceq d$. By Lemma 4.2 there exists a point x satisfying $d \preceq x \preceq b$ that can either be returned by the inner algorithm, or that satisfies $x \in \mathrm{Down}(f_s)$. Furthermore, using we can find this point in $O(\log n)$ queries using binary search. Thus we can spend $O(\log n)$ queries and either immediatly terminate, or in the case where we find a point $x \in \mathrm{Down}(f_s)$, we can repeat the argument above to show that mid and x violate order preservation.

(2) If Case 4.a.iii is triggered, then note that $\mathsf{mid}_1 > f(\mathsf{mid})_1$, and we argue that an order preservation violation can be found using at most $O(\log n)$ queries.
   (a) If $a \in \mathrm{Up}(f_s)$, then we have
   $$f(\mathsf{mid})_1 < \mathsf{mid}_1 = a_1 \leq f(\mathsf{mid})_1,$$
   meaning that $a \preceq \mathsf{mid}$, but $f(a) \not\preceq f(\mathsf{mid})$, and so a and mid violate order preservation.
   (b) If instead there is an up set witness (a, u), then note that due to the preprocessing, it must be a left-boundary up set witness, and due to Step 1, we must have $u \preceq \mathsf{mid}$. By Lemma 4.4 there exists a point x satisfying $a \preceq x \preceq u$ that can either be returned by the inner algorithm, or that satisfies $x \in \mathrm{Up}(f_s)$. Furthermore, we can find this point in $O(\log n)$ queries using binary search. Thus we can spend $O(\log n)$ queries and either immediatly terminate, or in the case where we find a point $x \in \mathrm{Up}(f_s)$, we can repeat the argument above to show that mid and x violate order preservation.

Termination. If the algorithm does not hit any of the cases that return a solution immediately, then it will continue until it finds an instance $L_{a,b}$ with $b_1 \leq a_1 + 1$ and $b_2 \leq a_2 + 1$ that satisfies the invariant. Lemma 4.6 implies that any sub-instance that satisfies the invariant contains a solution that can be returned by the inner algorithm. Since then $L_{a,b}$ contains at most four points, we can check all of them and then return the solution that must exist.

Query complexity. Observe that each iteration of the algorithm either finds a violation of order preservation, finds a solution after spending $O(\log n)$ further queries, or reduces the size of one of the dimensions by a factor of two. Moreover, each non-terminating iteration of the algorithm queries at most five points. Hence, if the algorithm is run on a sub-instance $L_{a,b}$ with $n_1 = b_1 - a_1$ and $n_2 = b_2 - a_2$, then the algorithm will terminate after making at most $O(\log n_1 + \log n_2 + \log n)$ queries. So the overall query complexity of the algorithm is $O(\log n)$, and we have shown the following theorem.

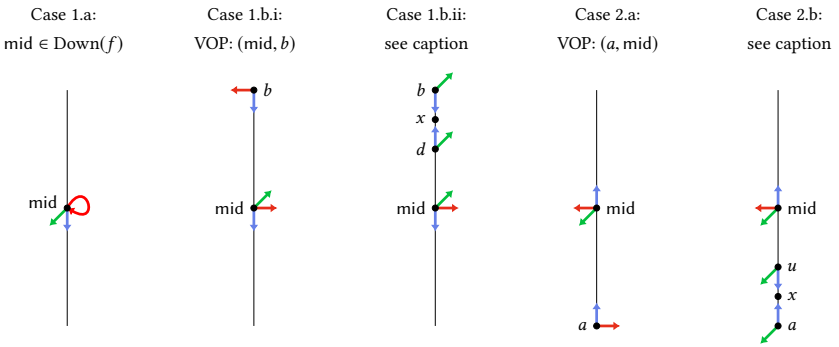

Fig. 9. The five cases that can be encountered if the algorithm fails to make progress for a width-zero instance. In Cases 1.b.ii and 2.b we spend $O(\log n)$ queries to find the point x, which then allows us to terminate.

Theorem 4.9. *There is an $O(\log n)$-query inner algorithm for 3-dimensional Tarski.*

Theorems 3.4 and 4.9 imply that 3-dimensional Tarski can be solved using $O(\log^2 n)$ queries, and this can be combined with the $\Omega(\log^2 n)$ lower bound for two-dimensional Tarski [7], to give the following theorem.

Theorem 4.10. *The deterministic query complexity of three-dimensional Tarski is $\Theta(\log^2 n)$.*

Proof. We will show that, if solving a $(k-1)$-dimensional Tarski instance requires q queries, then solving k-dimensional Tarski also requires q queries.

Let $L^{k-1} = \mathrm{Lat}(n_1, n_2, \ldots, n_{k-1})$ be a $(k-1)$-dimensional lattice, and let $f^{k-1} : L^{k-1} \to L^{k-1}$ be a Tarski instance over L^{k-1} that requires q queries to solve. Let $L^k = \mathrm{Lat}(n_1, n_2, \ldots, n_k)$ be a k-dimensional lattice, where n_k is any positive integer. We build the function $f^k : L^k \to L^k$ in the following way. For each point $x \in L^k$ we define

$$f^k(x)_i = \begin{cases} f^{k-1}(x)_i & \text{if } i < k, \\ x_k & \text{if } i = k. \end{cases}$$

Observe that $x = f^k(x)$ if and only if x is also a fixed point of f^{k-1}, and that x and y violate the order preservation of f^k if and only if x and y violate the order preservation of f^{k-1}. Moreover, every query to f^k can be answered by making exactly one query to f^{k-1}. Hence, in order solve the Tarski problem for f^k, we must make at least q queries.

Thus, Theorem 4.10 follows from the $\Omega(\log^2 n)$ query lower bound of Etessami et al. [7] for Tarski in dimension 2. □

5 HIGHER DIMENSIONS

In this section we will show that k-dimensional Tarski can be solved using $O(\log^{2\lceil k/3 \rceil} n)$ queries. We will prove this by showing that if a-dimensional Tarski can be solved in q_a queries, and b-dimensional Tarski can be solved in q_b queries, then $(a \cdot b)$-dimensional Tarski can be solved in $q_a \cdot (q_b + 2)$ queries. Using this fact along with our new algorithm for three dimensional Tarski yields the result.

So, let $L = \mathrm{Lat}(n_1, n_2, \ldots, n_k)$ be a k-dimensional lattice. Observe that for any pair of natural numbers $a, b \in \mathbb{N}$ such that $a + b = k$ we have that $L = A \times B$ where $A = \mathrm{Lat}(n_1, n_2, \ldots, n_a)$ and $B = \mathrm{Lat}(n_{a+1}, n_{a+2}, \ldots, a_{a+b})$. Given a point $x \in A$ and a point $y \in B$, we will use (x, y) to denote the point in L that is obtained by concatenating x and y.

Suppose that we have an algorithm $\mathcal{A}$ that solves Tarski on A using q_a queries, and an algorithm $\mathcal{B}$ that solves Tarski on B using q_b queries. Our goal is to construct an algorithm $\mathcal{L}$ that solves Tarski on L using $q_a \cdot (q_b + 2)$ queries.

The algorithm. At a high level, algorithm $\mathcal{L}$ will execute algorithm $\mathcal{A}$ on the lattice A. We will use $f : L \to L$ to denote the Tarski instance on $\mathcal{L}$, and $f_A : A \to A$ to denote the Tarski instance on $\mathcal{A}$.

Every time $\mathcal{A}$ queries a point $x \in A$, algorithm $\mathcal{L}$ will fix the following slice s_x of L

$$(s_x)_i = \begin{cases} x_i & \text{if } i \leq a, \\ * & \text{otherwise,} \end{cases}$$

which is the slice defined by fixing the dimensions specified by x and letting all other dimensions be free. The slice s_x defines a b-dimensional lattice B, and so we can apply $\mathcal{B}$ to solve the Tarski problem on the slice s_x. If $\mathcal{B}$ returns a violation of order preservation, then this is also a violation of order preservation in L and so $\mathcal{L}$ can terminate. Otherwise, it will return a fixed point $y \in B$ of the slice s_x. Given y, algorithm $\mathcal{L}$ can then respond to the query of $\mathcal{A}$ by setting $f_A(x)_i = f((x, y))_i$ for each dimension $i \leq a$.

With this approach it is clear that if $\mathcal{A}$ returns a fixed point $x \in A$, and if y is the corresponding fixed point of s_x, then (x, y) is a fixed point of L. However, if $\mathcal{A}$ returns a violation of order preservation, then the approach so far is not sufficient to produce an order-preservation violation in L. This is because, if $x_1 \preceq x_2$ witness a violation of order preservation in A, then the pair of points $(x_1, y_1), (x_2, y_2) \in L$, where each y_i is the fixed point of s_{x_i}, do not necessarily violate order preservation, because we have no guarantee that $(x_1, y_1) \preceq (x_2, y_2)$.

To address this, whenever we ask algorithm $\mathcal{B}$ to find a fixed point of the slice s_x, we restrict it to a sub-instance of the slice to ensure that the fixed point y can later be used to witness order-preservation violations if required. More concretely, the algorithm $\mathcal{L}$ does the following.

- Algorithm $\mathcal{L}$ maintains a set $P \subseteq A \times B$ of past points, which is initially set so that $P := \emptyset$. Whenever algorithm $\mathcal{A}$ queries $x \in A$ and algorithm $\mathcal{B}$ responds with $y \in B$, the pair (x, y) is added to P.
- Whenever $\mathcal{A}$ queries $x \in A$, algorithm $\mathcal{L}$ does the following.
  - It finds the set $D = \{y' \mid \text{There exists an } x' \text{ such that } (x', y') \in P \text{ and } x' \preceq x\}$, which contains the y values associated with all previously queried points that are below x in the $\preceq$ ordering. It then constructs the point $l \in B$ that is the least upper bound of the set D.
  - Symmetrically, it finds the set $U = \{y' \mid \text{There exists an } x' \text{ such that } (x', y') \in P \text{ and } x \preceq x'\}$, which contains the y values associated with all previously queried points that are above x in the $\preceq$ ordering. It then constructs the point $u \in B$ that is the greatest lower bound of the set U.
- Then, if B is the b-dimensional lattice defined by the slice s_x, algorithm $\mathcal{L}$ calls algorithm $\mathcal{B}$ on $B_{l,u}$, which is the sub-instance of B that contains all points between l and u in the $\preceq$ ordering.

Other than this change, the algorithm proceeds as described earlier.

Correctness. In order for the algorithm to work, we must show that the sub-instance $B_{l,u}$ is non-empty, and contains a solution for the Tarski problem. We verify both of these properties in the following pair of lemmas. The first lemma shows that $B_{l,u}$ is not empty.

Lemma 5.1. *In every iteration of the algorithm we have $l \preceq u$, which implies that $B_{l,u}$ is non-empty.*

Proof. We begin by observing that, by construction, the algorithm ensures that for every pair of points $(x_1, y_1), (x_2, y_2) \in P$, we have that $x_1 \preceq x_2$ implies $y_1 \preceq y_2$. Hence, for every $y \in D$ and $y' \in U$ we know that $y \preceq y'$.

We can use this to prove that $l \preceq u$. Suppose, for the sake of contradiction, that this is not the case, and therefore that there is some index i for which $l_i > u_i$. Let $y \in D$ be a point such that $y_i = l_i$, which must exist since otherwise l would not be the least upper bound of D. Likewise, let $y' \in U$ be a point such that $y'_i = u_i$, which must exist since otherwise u would not be the greatest lower bound of U. We have

$$y_i = l_i > u_i = y'_i,$$

and therefore we have $y \in D$, $y' \in U$, but $y \not\preceq y'$, giving us our contradiction. □

The second lemma shows that $B_{l,u}$ contains a solution to the Tarski problem.

Lemma 5.2. *Either a violation of order preservation in L can be found between l or u with some previously queried point or the sub-instance $B_{l,u}$ contains a solution to the Tarski problem.*

Proof. We begin by showing that either l is in the up set of B, or that a violation of order preservation can be found. Suppose that there is some index i satisfying $a < i \leq n$ such that $f(l)_i < l_i$. Then let $y \in D$ be a point such that $y_i = l_i$, which must exist since otherwise l would not be the least upper bound of D. Moreover, since l is the least upper bound of D, we know that $y \preceq l$. Since y was found in a previous iteration by $\mathcal{B}$, we know that it is a fixed point in dimensions $a+1$ through n and therefore $f(y)_i = y_i$. So we have

$$f(y)_i = y_i = l_i > f(l)_i,$$

meaning that we have $y \preceq l$ but $f(y) \not\preceq f(l)$, and therefore y and l violate order preservation. So we have proved that either $y_i \leq f(y)_i$ for all i in $a < i \leq n$, or there exists an order-preservation violation between l and some point y in D.

Symmetrically we can show that either u is in the down set of B, or that a violation of order preservation can be found. Suppose that there is some index i satisfying $a < i \leq n$ such that $f(u)_i > u_i$. Then let $y \in U$ be a point such that $y_i = u_i$, which must exist since otherwise u would not be the greatest lower bound of U. Moreover, since u is the greatest lower bound of U, we know that $u \preceq y$. Since y was found in a previous iteration by $\mathcal{B}$, we know that it is a fixed point in dimensions $a+1$ through n and therefore $f(y)_i = y_i$. So we have

$$f(u)_i > u_i = y_i = f(y)_i,$$

meaning that we have $u \preceq y$ but $f(u) \not\preceq f(y)$, and therefore u and y violate order preservation. So we have proved that either $u_i \geq f(y)_i$ for all i in $a < i \leq n$, or there exists an order-preservation violation between u and some point y in U.

Assuming that we have not found an order-preservation violation so far, we have shown that l is in the up set of B, and u is in the down set of B. Therefore we can apply Lemma 3.2 to show that there must exist a solution to the Tarski problem in $B_{l,u}$. □

To complete the correctness argument we just need to observe that if algorithm $\mathcal{A}$ returns a solution to the Tarski problem defined over A, then we can find a solution to the Tarski problem defined over L.

- If $\mathcal{A}$ returns a point $x \in A$ that is a fixed point of A, then the point (x, y), where y was the fixed point returned by algorithm $\mathcal{B}$ as the fixed point of s_x, is a fixed point of L.
- If $\mathcal{A}$ returns two points $x_1, x_2 \in A$ that violate order preservation in A, meaning that $x_1 \preceq x_2$ but $f_A(x_1) \not\preceq f_A(x_2)$, then let y_1 and y_2 be the points found by algorithm $\mathcal{B}$ for x_1 and x_2 respectively. Due to the procedure above, we know that $x_1 \preceq x_2$ implies that $(x_1, y_1) \preceq (x_2, y_2)$. Moreover, since there is some index i in the range $1 \leq i \leq a$

such that $f(x_1)_i > f(x_2)_i$, we have that $f(x_1, y_1)_i > f(x_2, y_2)_i$. Therefore, the points (x_1, y_1) and (x_2, y_2) violate order preservation in L.

Query Complexity. Recall that algorithm $\mathcal{A}$ solves Tarski using q_a queries, and that algorithm $\mathcal{B}$ solves Tarski using q_b queries. In algorithm $\mathcal{L}$, each time $\mathcal{A}$ makes a query we make queries to l and u, to verify that they do not violate order preservation in the sense of Lemma 5.2, and then make one call to $\mathcal{B}$. So in total $\mathcal{L}$ makes $q_a \cdot (q_b + 2)$ queries, and we have shown the following theorem.

Theorem 5.3. *If a-dimensional Tarski can be solved in q_a queries, and b-dimensional Tarski can be solved in q_b queries, then $(a + b)$-dimensional Tarski can be solved in $q_a \cdot (q_b + 2)$ queries.*

We should remark that, although we have proved Theorem 5.3 for grid lattices, since those are the lattices that we study in this paper, a more general version of the theorem for arbitrary lattices can also be shown to be true using essentially the same techniques: if any lattice $L = A \times B$, and there exist algorithms for Tarski for A and B that use q_a and q_b queries, respectively, then Tarski can be solved on L using $O(q_a \cdot q_b)$ queries.

Theorem 5.3 allows us to apply our $O(\log^2 n)$ query algorithm for three-dimensional Tarski to obtain the following algorithm for k dimensional Tarski.

Theorem 5.4. *k-dimensional Tarski can be solved using $O(\log^{2\lceil k/3 \rceil} n)$ queries.*

Proof. We can decompose the k-dimensional lattice L into a product of lattices

$$L = L_1 \times L_2 \times \cdots \times L_{\lceil k/3 \rceil},$$

where each lattice L_i has dimension at most three. Since we have an $O(\log^2 n)$ query algorithm for three-dimensional Tarski, we can repeatedly apply Theorem 5.3 to solve k-dimensional Tarski in $O(\log^{2\lceil k/3 \rceil} n)$ queries. □

Time complexity. To obtain time complexity results, note that writing down a point in the lattice L already requires $k \cdot \log n$ time. We assume that f is implemented by a Boolean circuit of size that is polynomial in k and $\log n$. With this assumption, we observe that all of our algorithms run in polynomial time with respect to k and $\log n$, and so our time complexity result can be stated as follows.

Theorem 5.5. *If f is presented as a Boolean circuit of size* $\text{poly}(\log n, k)$*, then there is an algorithm for Tarski that runs in time* $O(\text{poly}(\log n, k) \cdot \log^{2\lceil k/3 \rceil} n)$.

6 CONCLUSION

Our $O(\log^{2\lceil k/3 \rceil} n)$ query algorithm for k-dimensional Tarski falsifies prior conjectures that the problem required $\Omega(\log^k n)$ queries [2, 7]. This, of course, raises the question of what is the query complexity of finding a Tarski fixed point? While our upper bound is tight in three dimensions, it seems less likely to be the correct answer in higher dimensions. Indeed, there seems to be a fairly wide range of possibilities. Is it possible to show a $\log^{\Omega(k)} n$ query lower bound for the problem? Or perhaps there exists a fixed parameter tractable algorithm that uses $O(f(k) \cdot \log^2 n)$ queries? Both of those would be consistent with the known upper and lower bounds, and so further research will be needed to close the gap.

Theorem 5.3 provides a powerful new tool for reducing the query complexity of the Tarski problem in high dimensions, since it allows a faster algorithm for a constant-dimensional problem to reduce the query complexity of the higher-dimensional problem. This means that determining the query complexity of four- or five-dimensional Tarski problems could lead to even faster algorithms for higher dimensions, and so further study of these problems seems warranted.

ACKNOWLEDGMENTS

We would like to thank Kousha Etessami, Thomas Webster, and an anonymous reviewer for pointing out that the proof of Lemma 12 could be drastically simplified from its original version, and we would like to thank Balázs Keszegh for useful discussions on this topic.

REFERENCES

[1] Ching-Lueh Chang, Yuh-Dauh Lyuu, and Yen-Wu Ti. 2008. The complexity of Tarski's fixed point theorem. *Theor. Comput. Sci.* 401, 1-3 (2008), 228–235.

[2] Chuangyin Dang, Qi Qi, and Yinyu Ye. 2020. Computations and Complexities of Tarski's Fixed Points and Supermodular Games. *CoRR* abs/2005.09836 (2020). Stanford tech report version appeared in 2012.

[3] Chuangyin Dang and Yinyu Ye. 2018. *On the complexity of a class of Discrete Fixed Point Problems under the Lexicographic Ordering.* Technical Report. Technical Report, City University of Hong Kong, CY2018-3, 17 pages.

[4] Chuangyin Dang and Yinyu Ye. 2018. On the complexity of an expanded Tarski's fixed point problem under the componentwise ordering. *Theor. Comput. Sci.* 732 (2018), 26–45.

[5] Chuangyin Dang and Yinyu Ye. 2020. Erratum/Correction to "On the complexity of an expanded Tarski's fixed point problem under the componentwise ordering" [Theor. Comput. Sci. 732 (2018) 26-45]. *Theor. Comput. Sci.* 817 (2020), 80.

[6] Federico Echenique. 2007. Finding all equilibria in games of strategic complements. *J. Econ. Theory* 135, 1 (2007), 514–532.

[7] Kousha Etessami, Christos H. Papadimitriou, Aviad Rubinstein, and Mihalis Yannakakis. 2020. Tarski's Theorem, Supermodular Games, and the Complexity of Equilibria. In *Proc. of ITCS*, Vol. 151. 18:1–18:19.

[8] John Fearnley, Paul W. Goldberg, Alexandros Hollender, and Rahul Savani. 2021. The Complexity of Gradient Descent: CLS = PPAD ∩ PLS. In *Proc. of STOC (to appear).*

[9] John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. 2020. Unique end of potential line. *J. Comput. Syst. Sci.* 114 (2020), 1–35.

[10] John Fearnley and Rahul Savani. 2021. A Faster Algorithm for Finding Tarski Fixed Points. In *Proc. of STACS*. 29:1–29:16.

[11] Paul Milgrom and John Roberts. 1990. Rationalizability, Learning, and Equilibrium in Games with Strategic Complementarities. *Econometrica* 58, 6 (1990), 1255–1277.

[12] Alfred Tarski. 1955. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.* 5, 2 (1955), 285–309.

[13] Donald M. Topkis. 1979. Equilibrium Points in Nonzero-Sum n-Person Submodular Games. *SIAM J. Control Optim* 17 (1979), 773–787.

[14] Donald M. Topkis. 1998. *Supermodularity and Complementarity.* Princeton University Press.