Unique End of Potential Line*

John Fearnley[1], Spencer Gordon[2], Ruta Mehta[3], and Rahul Savani[1]

[1]University of Liverpool, {john.fearnley, rahul.savani}@liverpool.ac.uk
[2]California Institute of Technology, slgordon@caltech.edu
[3]University of Illinois at Urbana-Champaign, rutamehta@cs.illinois.edu

May 29, 2020

Abstract

The complexity class CLS was proposed by Daskalakis and Papadimitriou in 2011 to understand the complexity of important NP search problems that admit both path following and potential optimizing algorithms. Here we identify a subclass of CLS – called UniqueEOPL – that applies a more specific combinatorial principle that guarantees unique solutions. We show that UniqueEOPL contains several important problems such as the P-matrix Linear Complementarity Problem, finding fixed points of Contraction Maps, and solving Unique Sink Orientations (USOs). We identify a problem – closely related to solving contraction maps and USOs – that is complete for UniqueEOPL.

1 Introduction

Total function problems in NP. The complexity class TFNP contains search problems that are guaranteed to have a solution, and whose solutions can be verified in polynomial time [43]. While it is a semantically defined complexity class and thus unlikely to contain complete problems, a number of syntactically defined subclasses of TFNP have proven very successful at capturing the complexity of total search problems. In this paper, we focus on two in particular, PPAD and PLS. The class PPAD was introduced in [49] to capture the difficulty of problems that are guaranteed total by a parity argument. It has attracted intense attention in the past 15 years, culminating in a series of papers showing that the problem of computing a Nash-equilibrium in two-player games is PPAD-complete [8, 12], and more recently a conditional lower bound that rules out a PTAS for the problem [51]. No polynomial-time algorithms for PPAD-complete problems are known, and recent work suggests that no such algorithms are likely to exist [3, 24]. PLS is the class of problems that can be solved by local search algorithms (in perhaps exponentially-many steps). It has also attracted much interest since it was introduced in [36], and looks similarly unlikely to have polynomial-time algorithms. Examples of problems that are complete for PLS include the problem of computing a pure Nash equilibrium in a congestion game [18], a locally optimal max cut [52], or a stable outcome in a hedonic game [23].

*A preliminary version of this paper appears in the proceedings of ICALP 2019 [20].

Continuous Local Search. If a problem lies in both PPAD and PLS then it is unlikely to be complete for either class, since this would imply an extremely surprising containment of one class in the other. In their 2011 paper [13], Daskalakis and Papadimitriou observed that there are several prominent total function problems in PPAD ∩ PLS for which researchers have not been able to design polynomial-time algorithms. Motivated by this they introduced the class CLS, a syntactically defined subclass of PPAD∩PLS. CLS is intended to capture the class of optimization problems over a continuous domain in which a continuous potential function is being minimized and the optimization algorithm has access to a polynomial-time continuous improvement function. They showed that many classical problems of unknown complexity are in CLS, including the problem of solving a simple stochastic game, the more general problems of solving a Linear Complementarity Problem with a P-matrix, finding an approximate fixpoint to a contraction map, finding an approximate stationary point of a multivariate polynomial (the KKT problem), and finding a mixed Nash equilibrium of a congestion game.

CLS problems with unique solutions. In this paper we study an interesting subset of problems that lie within CLS and have *unique* solutions.

Contraction. In this problem we are given a function $f : \mathbb{R}^d \to \mathbb{R}^d$ that is purported to be c-contracting, meaning that for all points $x, y \in [0,1]^n$ we have $d(f(x), f(y)) \leq c \cdot d(x, y)$, where c is a constant satisfying $0 < c < 1$, and d is a distance metric. Banach's fixpoint theorem states that if f is contracting, then it has a unique *fixpoint* [2], meaning that there is a unique point $x \in \mathbb{R}^d$ such that $f(x) = x$.

P-LCP. The *P-matrix Linear Complementarity Problem* (P-LCP) is a variant of the linear complementarity problem in which the input matrix is a P-matrix [10]. An interesting property of this problem is that, if the input matrix actually is a P-matrix, then the problem is guaranteed to have a unique solution [10]. Designing a polynomial-time algorithm for P-LCP has been open for decades, at least since the 1978 paper of Murty [47] that provided exponential-time examples for *Lemke's algorithm* [40] for P-LCPs.

USO. A *Unique Sink Orientation* (USO) is an orientation of the edges of an n-dimensional hypercube such that every face of the cube has a unique sink. Since the entire cube is a face of itself, this means that there is a unique vertex of the cube that is a sink, meaning that all edges are oriented inwards. The USO problem is to find this *unique* sink.

All of these problems are most naturally stated as *promise* problems. This is because we have no way of verifying up front whether a function is contracting, whether a matrix is a P-matrix, or whether an orientation is a USO. Hence, it makes sense, for example, to study the contraction problem where it is promised that the function f is contracting, and likewise for the other two problems.

However, each of these problems can be turned into non-promise problems that lie in TFNP. In the case of Contraction, if the function f is not contracting, then there exists a short certificate of this fact. Specifically, any pair of points $x, y \in \mathbb{R}^d$ such that $d(f(x), f(y)) > c \cdot d(x, y)$ give an explicit proof that the function f is not contracting. We call these *violations*, since they witness a violation of the promise that is inherent in the problem.

So Contraction can be formulated as the non-promise problem of either finding a solution, or finding a violation. This problem is in TFNP because in the case where there is not a unique solution, there must exist a violation of the promise. The P-LCP and USO problems also have violations

that can be witnessed by short certificates, and so they can be turned into non-promise problems contained in the same way, and these problems also lie in TFNP.

For Contraction and P-LCP we actually have the stronger result that both problems are in CLS [13]. Prior to this work USO was not known to lie in any non-trivial subclass of TFNP, and placing USO into a non-trivial subclass of TFNP was identified as an interesting open problem by Kalai [37, Problem 6].

We remark that not every problem in CLS has the uniqueness properties that we identify above. For example, the KKT problem [13] lies in CLS, but has no apparent notion of having a unique solution. The problems that we identify here seem to share the special property that there is a natural promise version of the problem, and that promise problem always has a unique solution.

1.1 Our contribution

The complexity class UniqueEOPL. The main conceptual contribution of the paper is to define the complexity class UniqueEOPL, which naturally captures problems that lie at the intersection of PPAD and PLS *and* have a promise version that has a unique solution. As a first step, we define the complexity class EOPL, which combines in a natural way the canonical problems of PPAD and PLS. We have EOPL $\subseteq$ PPAD $\cap$ PLS by definition, and we also show that EOPL $\subseteq$ CLS (Corollary 5), by providing a polynomial-time reduction to the EndOfMeteredLine problem defined by Hubáček and Yogev [34], which they have shown to lie in CLS.

While the class EOPL captures problems that lie at the intersection of PPAD and PLS, it says nothing about problems with unique solutions. For this reason we define the complexity class UniqueEOPL $\subseteq$ EOPL, which adds three natural types of violation to the defining problem of EOPL. Together, these violations capture the idea of a promise problem that has a unique solution: if we are promised that none of these violations exist, then the EOPL problem must have a unique solution.

As such, there are two possible views of UniqueEOPL. The class UniqueEOPL is a class of total problems like any other subclass of TFNP. Problems in UniqueEOPL are *not* guaranteed to have unique solutions, since there may be violations. On the other hand, we also define the class PromiseUEOPL, in which it is promised that no violations occur. Problems in this class *are* guaranteed to have a unique solution.

The main results of this paper show containment results for UniqueEOPL, and we do not identify any problems that are in EOPL but not UniqueEOPL. This leaves as an open question the usefulness of EOPL: it provides an attractive definition of a combinatorial class at the intersection of PPAD and PLS, but as yet it has not been distinguished from UniqueEOPL. We think that it is unlikely that EOPL = UniqueEOPL, since UniqueEOPL places strong properties on the uniqueness of solutions. The main open question then, in our view, is to examine the relationship between EOPL and CLS. Are these two characterisations of PPAD $\cap$ PLS the same, or are there problems in CLS that are not in EOPL? A starting point would be to give EOPL containment results for the problems currently known to be in CLS but not in UniqueEOPL, but we leave that for future work.

UniqueEOPL containment results. The main technical results of the paper are that USO, P-LCP, and a variant of the Contraction problem all lie in UniqueEOPL. We are interested in both the promise and non-promise versions of these problems. To this end, we define the concept of a *promise-preserving* reduction, which is a polynomial-time reduction between two problems A and B, with the property that if A is promised to have a unique solution, then the instance of B that is produced by the reduction will also have a unique solution. All of the reductions that we produce

in this paper are promise-preserving, which means that whenever we show that a problem is in UniqueEOPL, we also get that the corresponding promise problem lies in PromiseUEOPL.

For the USO problem, our UniqueEOPL containment result (Theorem 24) substantially advances our knowledge about the problem. Prior to this work, the problem was only known to lie in TFNP, and Kalai [37, Problem 6] had posed the challenge to place it in some non-trivial subclass of TFNP. Our result places USO in UniqueEOPL, EOPL, CLS, PPAD (and hence PPA and PPP), and PLS, and so we answer Kalai's challenge by placing the problem in *all* of the standard subclasses of TFNP[1].

This result is in some sense surprising. Although every face of a USO has a unique sink, the orientation itself may contain cycles, and so there is no obvious way to define a potential function for the problem. Moreover, none of the well-known algorithms for solving USOs [30, 59] have the line-following nature needed to place a problem in PPAD.

The P-LCP problem is naturally a promise problem. To make the problem total, one asks to find a solution to the LCP or a short witness that the matrix is not a P-matrix. In this paper, we study a variant of the P-LCP problem with a violation inspired by the standard reduction from P-LCPs to USOs. Indeed, to show that this version of P-LCP is in UniqueEOPL, we observe that the standard reduction to USO corresponds to a promise-preserving reduction to our total version of the USO problem (Theorem 33). Since the reductions that place P-LCP in UniqueEOPL are promise-preserving, we get that promise P-LCP lies in PromiseUEOPL.

For finding fixpoints of contraction maps, we study the problem PL-Contraction, with contraction maps specified by LinearFIXP arithmetic circuits [17], i.e., *piecewise-linear* functions, that are contracting with respect to an ℓ_p norm. [13] studied contraction maps specified by arbitrary arithmetic circuits, which is not suitable for us, since the unique fixpoint of such a function may not be rational, and, given our interest in uniqueness we cannot resort to approximation. We note that contraction maps specified by LinearFIXP circuits are still interesting, since they are powerful enough to represent simple stochastic games [17].

We place this problem in UniqueEOPL via a promise-preserving reduction (Theorem 30). Our reduction can produce multiple types of violation. In addition to the standard violation of a pair of points at which f is not contracting, our reduction sometimes produces a different type of violation (cf. Definition 28), which while not giving an explicit violation of contraction, still gives a short certificate that f is not contracting.

Other problems in UniqueEOPL. Our results imply that several other problems lie in UniqueEOPL. The simple stochastic game (SSG) problem is known to reduce to PL-Contraction [17] and to P-LCP [29], and thus our two results give two separate proofs that the SSG problem lies in UniqueEOPL. It is known that discounted games can be reduced to SSGs [61], mean-payoff games can be reduced to discounted games [61], and parity games can be reduced to mean-payoff games [50]. So all of these problems lie in UniqueEOPL too. Finally, Gärtner et al. [26] noted that the ARRIVAL problem [16] lies in EOPL, and in fact their EndOfPotentialLine instance always contains exactly one line, and so ARRIVAL also lies in UniqueEOPL. We remark that none of these are promise-problems. Each of them can be formulated so that they *unconditionally* have a unique solution. Hence, these problems seem to be easier than the problems captured by UniqueEOPL, since problems that are complete for UniqueEOPL only have a unique solution conditioned on the promise that there are no violations.

A complete problem for UniqueEOPL. Finally we also give a UniqueEOPL-completeness result. Specifically, we show that *One-Permutation Discrete Contraction* (OPDC) is complete for

[1] However, we do not place the problem in the recently defined class PWPP [35].

UniqueEOPL (Theorem 16). OPDC is a problem that is inspired by both PL-Contraction and USO. Intuitively, it is a discrete version of PL-Contraction, and can also be viewed as a variant of USO in which only some of the faces have unique sinks.

OPDC actually plays a central role in the paper. We reduce PL-Contraction, USO, and P-LCP to it, and we then show that OPDC is in UniqueEOPL, as shown in Figure 1. The reduction from OPDC to the problem UniqueEOPL, the canonical complete problem for the class UniqueEOPL, is by far the most difficult and involved step.

Our hardness reduction produces an OPDC instance where the set of points in the instance is a hypercube. This makes significant progress towards showing a hardness result for PL-Contraction and USO. As we have mentioned, OPDC is a discrete variant of PL-Contraction, and when the set of points is a hypercube, the problem is also very similar to USO. The key difference is that OPDC insists that only certain "slices" should have a unique fixpoint, whereas Contraction and USO insist that *all* slices should have unique fixpoints. To show a hardness result for either of those two problems, one would need to produce an OPDC instance with that property.

1.2 Related work

CLS. Recent work by Hubáček and Yogev [34] proved lower bounds for CLS. They introduced a problem known as EndOfMeteredLine which they showed was in CLS, and for which they proved a query complexity lower bound of $\Omega(2^{n/2}/\sqrt{n})$ and hardness under the assumption that there were one-way permutations and indistinguishability obfuscators for problems in P/poly. As we will see, these lower bounds also apply to our new complexity class UniqueEOPL. Another recent result showed that the search version of the Colorful Carathéodory Theorem is in PPAD ∩ PLS, and left open whether the problem is also in CLS [45].

Until recently, it was not known whether there was a natural CLS-complete problem. In their original paper, Daskalakis and Papadimitriou suggested two natural candidates for CLS-complete problems, ContractionMap and P-LCP, which we study in this paper. Recently, two variants of ContractionMap have been shown to be CLS-complete. Whereas in the original definition of ContractionMap it is assumed that an ℓ_p or ℓ_∞ norm is fixed, and the contraction property is measured with respect to the metric induced by this fixed norm, in these two new complete variants, a metric [14] and meta-metric [19] are given as part of the input.

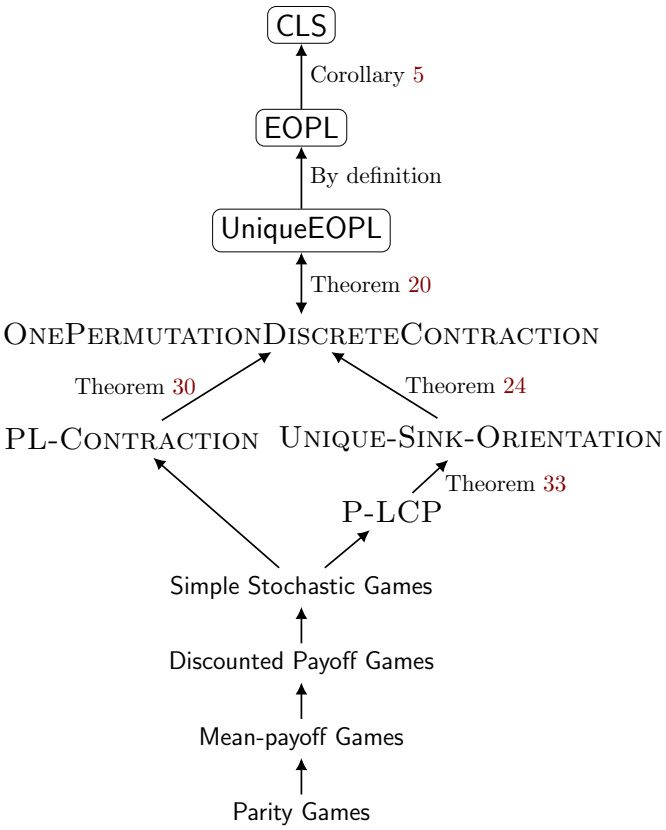

Figure 1: The relationship of UniqueEOPL to other classes and problems.

CLS also contains the problem of finding an approximate stationary point of a multivariate polynomial (the KKT problem) [13]. We do not place KKT in EOPL or UniqueEOPL in this paper,

which makes it one of the most prominent problems that is in CLS, but not in our two new classes.

P-LCP. Papadimitriou showed that P-LCP, the problem of solving the LCP or returning a violation of the P-matrix property, is in PPAD [49] using Lemke's algorithm. The relationship between Lemke's algorithm and PPAD has been studied by Adler and Verma [1]. Later, Daskalakis and Papadimitriou showed that P-LCP is in CLS [13], using the potential reduction method in [39]. Many algorithms for P-LCP have been studied, e.g., [38, 46, 47]. However, no polynomial-time algorithms are known for P-LCP, or for the promise version where one can assume that the input matrix is a P-matrix.

Unique Sink Orientations. USOs of cubes were first studied by Stickney and Watson [58] in the context of P-matrix LCPs. A USO arising from a P-matrix may be cyclic. Motivated by Linear Programming, *acyclic*, AUSOs have also been studied, both for cubes and general polytopes [27, 32]. Recently Gärtner and Thomas studied the computational complexity of recognizing USOs and AUSOs [28]. They found that the problem is coNP-complete for USOs and PSPACE-complete for AUSOs. While many papers have studied specific algorithms for solving (A)USOs [21, 22, 25, 30, 41, 53, 54, 59, 60], to the best of our knowledge, we are first to study the general problem of solving a USO from a complexity-theoretic point of view.

Contraction. The problem of computing a fixpoint of a continuous map $f : \mathcal{D} \mapsto \mathcal{D}$ with Lipschitz constant c has been extensively studied, in both continuous and discrete variants [6, 7, 15]. For arbitrary maps with $c > 1$, exponential bounds on the query complexity of computing fixpoints are known [5, 31]. There has been extensive work on algorithms for computing approximate fixpoints of contraction maps in a range of settings [4, 33, 48, 55–57].

2 Unique End of Potential Line

In this section we define two new complexity classes called EOPL and UniqueEOPL. These two classes are defined by merging the definitions of PPAD and PLS, so we will begin by recapping those classes.

The complexity class PPAD contains every problem that can be reduced to EndOfLine [49].

Definition 1 (EndOfLine)**.** *Given Boolean circuits $S, P : \{0,1\}^n \to \{0,1\}^n$ such that $P(0^n) = 0^n \neq S(0^n)$, find one of the following:*

(E1) A point $x \in \{0,1\}^n$ such that $P(S(x)) \neq x$.

(E2) A point $x \in \{0,1\}^n$ such that $x \neq 0^n$ and $S(P(x)) \neq x$.

Intuitively, the problem defines an exponentially large graph where all vertices have in-degree and out-degree at most one. Each bit-string in $\{0,1\}^n$ defines a vertex, while the functions S and P define *successor* and *predecessor* functions for each vertex. A directed edge exists from vertex x to y if and only if $S(x) = y$ and $P(y) = x$. Any vertex x for which $P(S(x)) \neq x$ has no outgoing edge, while every vertex y with $S(P(y)) \neq y$ has no incoming edge.

The condition that $P(0^n) = 0^n \neq S(0^n)$ specifies that the vertex 0^n has no incoming edge, and so it is the start of a line. To solve the problem, we must find either a solution of type (E1), which is a vertex x that is the end of a line, or a solution of type (E2), which is a vertex x other than 0^n that is the start of a line. Since we know that the graph has at least one source, there must at least exist a solution of type (E1), and so the problem is in TFNP.

The complexity class PLS contains every problem that can be reduced to SinkOfDag[2].

Definition 2 (SinkOfDag)**.** *Given a Boolean circuit* $S : \{0,1\}^n \to \{0,1\}^n$ *such that* $S(0^n) \neq 0^n$*, and a circuit* $V : \{0,1\}^n \to \{0,1,\ldots,2^m-1\}$ *find:*

(S1) A vertex $x \in \{0,1\}^n$ *such that* $S(x) \neq x$ *and either* $S(S(x)) = x$ *or* $V(S(x)) \leq V(x)$*.*

Once again, the problem specifies an exponentially large graph on the vertex set $\{0,1\}^n$, but this time the only guarantee is that each vertex has out-degree one. The circuit S gives a successor function. In this problem, not all bit-strings need to correspond to vertices in the graph. Specifically, if we have $S(x) = x$ for some bit-string $x \in \{0,1\}^n$, then x does not encode a vertex.

The second circuit V gives a *potential* to each vertex from the set $\{0,1,\ldots,2^m-1\}$. An edge exists in the graph if and only if the potential *increases* along that edge. Specifically, there is an edge from x to y if and only if $S(x) = y$ and $V(x) < V(y)$. This restriction means that the graph must be a DAG.

To solve the problem, we must find a sink of the DAG, i.e., a vertex that has no outgoing edge. Since we require that $S(0^n) \neq 0^n$, we know that the DAG has at least one vertex, and therefore it must also have at least one sink. This places the problem in TFNP.

End of potential line. We define a new problem called EndOfPotentialLine, which merges the two definitions of EndOfLine and SinkOfDag into a single problem.

Definition 3 (EndOfPotentialLine)**.** *Given Boolean circuits* $S, P : \{0,1\}^n \to \{0,1\}^n$ *such that* $P(0^n) = 0^n \neq S(0^n)$ *and a Boolean circuit* $V : \{0,1\}^n \to \{0,1,\ldots,2^m-1\}$ *such that* $V(0^n) = 0$ *find one of the following:*

(R1) A point $x \in \{0,1\}^n$ *such that* $S(P(x)) \neq x \neq 0^n$ *or* $P(S(x)) \neq x$*.*

(R2) A point $x \in \{0,1\}^n$ *such that* $x \neq S(x)$*,* $P(S(x)) = x$*, and* $V(S(x)) - V(x) \leq 0$*.*

This problem defines an exponentially large graph where each vertex has in-degree and out-degree at most one (as in EndOfLine) that is also a DAG (as in SinkOfDag). Hence, the graph consists entirely of directed paths, or *lines* as we shall call them in this paper. An edge exists from x to y if and only if $S(x) = y$, $P(y) = x$, and $V(x) < V(y)$. As in SinkOfDag, only some bit-strings encode vertices, and we adopt the same idea that if $S(x) = x$ for some bit-string x, then x does *not* encode a vertex.

So we have a single instance that is simultaneously an instance of EndOfLine and an instance of SinkOfDag. To solve the problem, it suffices to solve *either* of these problems. Solutions of type (R1) consist of vertices x that are either the end of a line, or the start of a line (excluding the case where $x = 0^n$). Solutions of type (R2) consist of any point x where the potential does not strictly increase on the edge between x and $S(x)$.

The complexity class EOPL. We define the complexity class EOPL to consist of all problems that can be reduced in polynomial time to EndOfPotentialLine. By definition the problem lies in $\mathsf{PPAD} \cap \mathsf{PLS}$, since one can simply ignore solutions of type (R2) to obtain an EndOfLine instance, and ignore solutions of type (R1) to obtain a SinkOfDag instance.

In fact we are able to show the stronger result that $\mathsf{EOPL} \subseteq \mathsf{CLS}$. To do this, we reduce EndOfPotentialLine to the problem EndOfMeteredLine, which was defined by Hubáček and

[2]While this is not the standard definition of PLS, it has been observed that the standard PLS-complete problem can easily be recast as a SinkOfDag instance [49].

Yogev, who also showed that the problem lies in CLS [34]. The main difference between the two problems is that EndOfMeteredLine requires that the potential increases by *exactly* one along each edge. The reduction from EndOfMeteredLine to EndOfPotentialLine is straightforward. The other direction is more involved, and requires us to insert new vertices into the instance. Specifically, if there is an edge between a vertex x and a vertex y, but $V(y) \neq V(x) + 1$, then we need to insert a new chain of vertices of length $V(y) - V(x) - 1$ between x and y, so that we can ensure that the potential always increases by exactly one along each edge. The following theorem is proved in Appendix A.1.

Theorem 4. EndOfMeteredLine *and* EndOfPotentialLine *are polynomial-time equivalent.*

As we have mentioned, Hubáček and Yogev have shown that EndOfMeteredLine lies in CLS [34], so we get the following corollary.

Corollary 5. EOPL $\subseteq$ CLS.

Problems with unique solutions. The problems that we study in this paper all share a specific set of properties that cause them to produce an interesting subclass of EndOfPotentialLine instances. Each of the problems that we study has a *promise*, and if the promise is satisfied the problem has a *unique* solution.

For example, in the Contraction problem, we are given a function $f : [0,1]^d \to [0,1]^d$ that is promised to be *contracting*, meaning that $d(f(x), f(y)) \leq c \cdot f(x,y)$ for some positive constant $c < 1$ and some distance metric d. We cannot efficiently check whether f is actually contracting, but if it is, then Banach's fixpoint theorem states that f has a unique fixpoint [2]. If f is not contracting, then there will exist *violations* that can be witnessed by short certificates. For Contraction, a violation is any pair of points x, y such that $d(f(x), f(y)) > c \cdot f(x,y)$.

We can use violations to formulate the problem as a non-promise problem that lies in TFNP. Specifically, if we ask for either a fixpoint or a violation of contraction, then the contraction problem is total, because if there is no fixpoint, then the contrapositive of Banach's theorem implies that there must exist a violation of contraction.

Unique End of Potential Line. When we place this type of problem in EOPL, we obtain an instance with extra properties. Specifically, if the original instance has no violations, meaning that the promise is satisfied, then the EndOfPotentialLine instance will contain a *single* line that starts at 0^n and ends at the unique solution of the original problem. This means that, if we ever find two distinct lines in our EndOfPotentialLine instance, then we immediately know that the instance fails to satisfy the promise.

We define the following problem, which is intended to capture these properties.

Definition 6 (UniqueEOPL)**.** *Given Boolean circuits $S, P : \{0,1\}^n \to \{0,1\}^n$ such that $P(0^n) = 0^n \neq S(0^n)$ and a Boolean circuit $V : \{0,1\}^n \to \{0,1,\ldots,2^m - 1\}$ such that $V(0^n) = 0$ find one of the following:*

(U1) A point $x \in \{0,1\}^n$ such that $P(S(x)) \neq x$.

(UV1) A point $x \in \{0,1\}^n$ such that $x \neq S(x)$, $P(S(x)) = x$, and $V(S(x)) - V(x) \leq 0$.

(UV2) A point $x \in \{0,1\}^n$ such that $S(P(x)) \neq x \neq 0^n$.

(UV3) Two points $x, y \in \{0,1\}^n$, such that $x \neq y$, $x \neq S(x)$, $y \neq S(y)$, and either $V(x) = V(y)$ or $V(x) < V(y) < V(S(x))$.

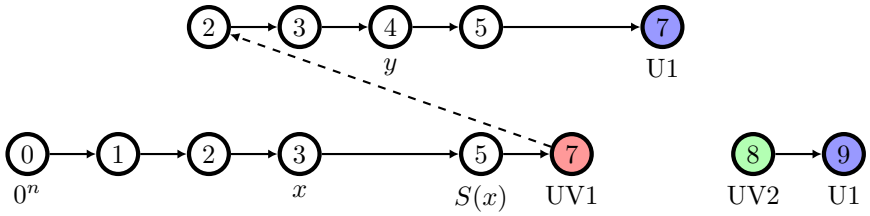

Figure 2: Example of solutions, including violations, for UniqueEOPL. This figure should be viewed in color. In this example we have 3 lines. The *main line* starts at 0^n and ends with a UV1 solution. There is a line of length one to the bottom right, which has a start that is a UV2 solution. This line does not intersect either of the other two lines in terms of the ranges of their potential values, so does not contribute to UV3 solutions. The main line and top line do intersect in terms of the ranges of their potential values and they contribute many UV3 solutions. We highlight one on the diagram with x, $S(x)$, and y, such that $V(x) < V(y) < V(S(x))$. There are two U1 solutions at the end of the top and bottom right lines. Finally, we note that if there were no violations, then there must be one line and a single U1 solution at the end of the main line.

We split the solutions into two types: proper solutions and violations. Solutions of type (U1) encode the end of any line, which are the proper solutions to the problem. There are three types of violation solution. Violations of type (UV1) are vertices at which the potential fails to strictly increase. Violations of type (UV2) ask for the start of any line, other than the vertex 0^n. Clearly, if there are two sources in the graph, then there are two lines.

Violations of type (UV3) give another witness that the instance contains more than one line. This is encoded by a pair of vertices x and y, with either $V(x) = V(y)$, or with the property that the potential of y lies between the potential of x and $S(x)$. Since we require the potential to strictly increase along every edge of a line, this means that y cannot lie on the same line as x, since all vertices before x in the line have potential strictly less than $V(x)$, while all vertices after $S(x)$ have potential strictly greater than $V(S(x))$.

We remark that (UV2) by itself already captures the property "there is a unique line", since if a second line cannot start, then it cannot exist. So why do we insist on the extra type of violation given by (UV3)? Violations of type (UV3) allow us to solve the problem immediately if we ever detect the existence of multiple lines. Note that this is not the case if we only have solutions of type (UV2), since we may find two vertices on two different lines, but both of them may be exponentially many steps away from the start of their respective lines.

By adding (UV3) solutions, we make the problem easier than EndOfPotentialLine (note that without UV3, the problem is actually the same as EndOfPotentialLine). This means that problems that can be reduced to UniqueEOPL have the very special property that, if at any point you detect the existence of multiple lines, either through the start of a second line, or through a violation in (UV3), then you *immediately* get a violation in the original problem without any extra effort. All of the problems that we study in this paper share this property.

The complexity class UniqueEOPL. We define the complexity class UniqueEOPL to be the class of problems that can be reduced in polynomial time to UniqueEOPL[3]. We note that UniqueEOPL $\subseteq$ EOPL is trivial, since the problem remains total even if we disallow solutions of type

[3] We remark that Hubáček and Yogev [34] mention that their lower bound results for CLS may also apply to such problems, but they did not investigate a corresponding complexity class.

UV3.

For each of our problems, it is also interesting to consider the complexity of the promise variant, in which it is guaranteed via a promise that no violations exist. We define PromiseUniqueEOPL to be the promise version of UniqueEOPL in which 0^n is the only start of a line (and hence there are no solutions that are type (UV2) or (UV3)). We define the complexity class PromiseUEOPL to be the class of promise problems that can be reduced in polynomial time to PromiseUniqueEOPL.

Promise-preserving reductions. The problem UniqueEOPL has the interesting property that, if it is promised that there are no violation solutions, then there must be a unique solution. All of the problems that we study in this paper share this property: their solutions can naturally be partitioned into proper solutions, which are the solutions to well-formed instances, and violation solutions, which give a succinct proof that the instance is not well-formed. When we reduce these problems to UniqueEOPL, the resulting instance will have a unique line whenever the original problem has no violation solutions.

We formalise this by defining the concept of a *promise-preserving* reduction. This is a reduction from problem A to problem B, both of which have proper solutions and violation solutions. The reduction is promise-preserving if, when it is promised that A has no violations, then the resulting instance of B also has no violations.

Definition 7 (Promise-preserving reduction)**.** *Let (f, g) be a polynomial-time reduction from problem A to problem B, where f maps each instance I of problem A to an instance I' of problem B, and g maps each solution of I' back to to a solution of I. Suppose that problems A and B both have functions* $\text{proper}(I)$ *and* $\text{violation}(I)$ *that, for each instance I, give a set of proper solutions to I and a set of violation solutions to I, respectively, where* $\text{proper}(I) \cup \text{violation}(I)$ *is required to contain every solution of I. The reduction (f, g) is promise-preserving if $x \in \text{violation}(f(I))$ implies that $g(x, f(I)) \in \text{violation}(I)$ for all instances I.*

Hence, if we reduce a problem to UniqueEOPL via a chain of promise-preserving reductions, and we know that there are no violations in the original problem, then there is a unique line ending at the unique proper solution in the UniqueEOPL instance.

Note that this is more restrictive than a general reduction. We could in principle produce a reduction that took an instance of A, where it is promised that there are no violations, and produce an instance of B that sometimes contains violations. By using promise-preserving reductions, we are showing that our problems have the natural properties that one would expect for a problem in UniqueEOPL. Specifically, that the promise version has a unique solution, and that this can be found by following the unique line in the UniqueEOPL instance.

One added bonus is that, if we show that a problem is in UniqueEOPL via a chain of promise-preserving reductions, then we automatically get that the promise version of that problem, where it is promised that there are no violations, lies in PromiseUEOPL. Moreover, if we show that a problem is UniqueEOPL-complete via a promise-preserving reduction, then this also implies that the promise version of that problem is PromiseUEOPL-complete. In this paper we will show that the OPDC problem is UniqueEOPL-complete, and this means that the promise version of OPDC is PromiseUEOPL-complete.

3 One-Permutation Discrete Contraction

The *One-Permutation Discrete Contraction* (OPDC) problem will play a crucial role in our results. We will show that the problem lies in UniqueEOPL, and we will then reduce both PL-Contraction

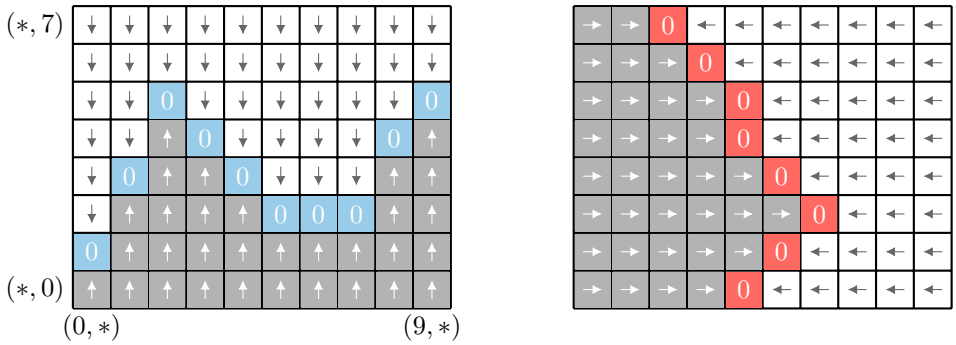

Figure 3: Left: A direction function for the up/down dimension. Right: A direction function for the left/right dimension. This figure should be viewed in color.

and Unique-Sink-Orientation to OPDC, thereby showing that those problems also lie in UniqueEOPL. We will also show that UniqueEOPL can be reduced to OPDC, making this problem the first example of a non-trivial UniqueEOPL-complete problem.

Direction functions. The problem OPDC can be seen as a discrete variant of the continuous contraction problem. Recall that a contraction map is a function $f : [0,1]^d \to [0,1]^d$ that is contracting under a metric m, i.e., $m(f(x), f(y)) \leq c \cdot m(x,y)$ for all $x, y \in [0,1]^d$ and some constant c satisfying $0 < c < 1$.

We will discretize the space by overlaying a grid of points on the $[0,1]^d$ cube. Let $[k]$ denote the set $\{0, 1, \ldots, k\}$. Given a tuple of grid widths $(k_1, k_2, \ldots, k_d)$, we define the set

$$P(k_1, k_2, \ldots, k_d) = [k_1] \times [k_2] \times \cdots \times [k_d].$$

We will refer to $P(k_1, k_2, \ldots, k_d)$ simply as P when the grid widths are clear from the context. Note that each point $p \in P$ is a tuple $(p_1, p_2, \ldots, p_d)$, where p_i is an integer between 0 and k_i, and this point maps onto the point $(p_1/k_1, p_2/k_2, \ldots, p_d/k_d) \in [0,1]^d$.

Instead of a single function f, in the discrete version of the problem we will use a family of *direction functions* over the grid P. For each dimension $i \leq d$, we have function $D_i : P \to \{\mathsf{up}, \mathsf{down}, \mathsf{zero}\}$. Intuitively, the natural reduction from a contraction map f to a family of direction functions would, for each point $p \in P$ and each dimension $i \leq d$ set:

- $D_i(p) = \mathsf{up}$ whenever $f(p)_i > p_i$,
- $D_i(p) = \mathsf{down}$ whenever $f(p)_i < p_i$, and
- $D_i(p) = \mathsf{zero}$ whenever $f(p)_i = p_i$.

In other words, the function D_i simply outputs whether $f(p)$ moves up, down, or not at all in dimension i. So a point $p \in P$ with $D_i(p) = \mathsf{zero}$ for all i would correspond to the fixpoint of f. Note, however, that the grid may not actually contain the fixpoint of f, and so there may be no point p satisfying $D_i(p) = \mathsf{zero}$ for all i.

A two-dimensional example. To illustrate this definition, consider the two-dimensional instance given in Figure 3, which we will use as a running example. It shows two direction functions: the figure on the left shows a direction function for the up-down dimension, which we will call dimension

1 and illustrate using the color blue. The figure on the right shows a direction function for the left-right dimension, which we will call dimension 2 and illustrate using the color red. Each square in the figures represents a point in the discretized space, and the value of the direction function is shown inside the box. Note that there is exactly one point p where $D_1(p) = D_2(p) = \mathsf{zero}$, which is the fixpoint that we seek.

Slices. We will frequently refer to subsets of P in which some of the dimensions have been fixed. A *slice* will be represented as a tuple $(s_1, s_2, \ldots, s_d)$, where each s_i is either

- a number in $[0, 1]$, which indicates that dimension i should be fixed to s_i, or
- the special symbol $*$, which indicates that dimension i is free to vary.

We define $\mathsf{Slice}_d = ([0, 1] \cup \{*\})^d$ to be the set of all possible slices in dimension d. Given a slice $s \in \mathsf{Slice}_d$, we define $P_s \subseteq P$ to be the set of points in that slice, i.e., P_s contains every point $p \in P$ such that $p_i = s_i$ whenever $s_i \neq *$. We say that a slice $s' \in \mathsf{Slice}_d$ is a sub-slice of a slice $s \in \mathsf{Slice}_d$ if $s_j \neq * \implies s'_j = s_j$ for all $j \in [d]$.

An *i-slice* is a slice s for which $s_j = *$ for all $j \leq i$, and $s_j \neq *$ for all $j > i$. In other words, all dimensions up to and including dimension i are allowed to vary, while all other dimensions are fixed. Note that this definition uses the specific ordering of the coordinates: different orderings would give different i-slices. This will eventually lead to the "one permutation" property of OPDC.

In our two-dimensional example, there are three types of i-slices. For each pair x, y there is a 0-slice (y, x), which contains only the exact point corresponding to x and y. For each x, there is a 1-slice $(*, x)$, which restricts the left/right dimension to the value x. There is one 2-slice: the slice $(*, *)$ that contains every point.

Discrete contraction maps. We can now define a one-permutation discrete contraction map. We say that a point $p \in P_s$ in some slice s is a *fixpoint* of s if $D_i(p) = \mathsf{zero}$ for all dimensions i where $s_i = *$. The following definition captures the promise version of the problem, and we will later give a non-promise version by formulating appropriate violations.

Definition 8 (One-Permutation Discrete Contraction Map)**.** *Let P be a grid of points over $[0, 1]^d$ and let $\mathcal{D} = (D_i)_{i=1,\ldots,d}$ be a family of direction functions over P. We say that $\mathcal{D}$ and P form a one-permutation discrete contraction map if, for every i-slice s, the following conditions hold.*

1. *There is a unique fixpoint of s.*
2. *Let $s' \in \mathsf{Slice}_d$ be a sub-slice of s where coordinate i has been fixed to a value, and all other coordinates are unchanged (meaning that s' an $(i-1)$-slice). If q is the unique fixpoint of s, and p is the unique fixpoint of s', then*
   - *if $p_i < q_i$, then $D_i(p) = \mathsf{up}$, and*
   - *if $p_i > q_i$, then $D_i(p) = \mathsf{down}$.*

The first condition specifies that each i-slice must have a unique fixed point. Since the slice $(*, *, \ldots, *)$ is an i-slice (specifically, it is the only d-slice), this implies that the full problem also has a unique fixpoint.

The second condition is a more technical condition. It tells us that if we have found the unique fixpoint p of a $(i-1)$-slice s', and if this point is not the unique fixpoint of the i-slice s that contains s', then the direction function $D_i(p)$ tells us which way to walk to find the unique fixpoint of s. This is a crucial property that we will use in our reduction from OPDC to UniqueEOPL.

In our two-dimensional example, the first condition requires that every slice $(*, x)$ has a unique fixpoint, and this corresponds to saying that for every fixed slice of the left/right dimension, there is a unique blue point that is zero. The second condition requires that, if we are at some blue zero, then the red direction function at that point tells us the direction of the overall fixpoint. It can be seen that our example satisfies both of these requirements.

Note that both properties only consider i-slices. In the continuous contraction map problem with an L_p norm distance metric, *every* slice has a unique fixpoint, and so one may expect a discrete version of contraction to share this property. The problem is that the second property is very difficult to prove. Indeed, when we reduce PL-Contraction to OPDC in Section 5.2, we must carefully choose the grid size to ensure that both the first and second properties hold. In fact, our choice of grid size for dimension i will depend on the grid size of dimension $i+1$, which is why our definition only considers i-slices.

The name *One-Permutation* Discrete Contraction was chosen to emphasize this fact. The i-slices correspond to restricting dimensions in order, starting with dimension d. Since the order of the dimensions is arbitrary, we could have chosen any permutation of the dimensions, but we must choose *one* of these permutations to define the problem.

The OPDC problem. The OPDC problem (defined formally in Definition 9) is as follows: given a discrete contraction map $\mathcal{D} = (D_i(p))_{i=1,\dots,d}$, find the unique point p such that $D_i(p) = \mathsf{zero}$ for all i. Note that we cannot efficiently verify whether $\mathcal{D}$ is actually a one-permutation discrete contraction map.

So, the OPDC problem is a promise problem, and we will formulate a total variant of it that uses a set of violations to cover the cases where $\mathcal{D}$ fails to be a discrete contraction map.

Definition 9 (OPDC)**.** *Given a tuple $(k_1, k_2, \dots, k_d)$ and circuits $(D_i(p))_{i=1,\dots,d}$, where each circuit $D_i : P(k_1, k_2, \dots, k_d) \to \{\mathsf{up}, \mathsf{down}, \mathsf{zero}\}$, find one of the following*

(O1) A point $p \in P$ such that $D_i(p) = \mathsf{zero}$ for all i.

(OV1) An i-slice s and two points $p, q \in P_s$ with $p \neq q$ such that $D_j(p) = D_j(q) = \mathsf{zero}$ for all $j \leq i$.

(OV2) An i-slice s and two points $p, q \in P_s$ such that

- *$D_j(p) = D_j(q) = \mathsf{zero}$ for all $j < i$,*
- *$p_i = q_i + 1$, and*
- *$D_i(p) = \mathsf{down}$ and $D_i(q) = \mathsf{up}$.*

(OV3) An i-slice s and a point $p \in P_s$ such that

- *$D_j(p) = \mathsf{zero}$ for all $j < i$, and either*
- *$p_i = 0$ and $D_i(p) = \mathsf{down}$, or*
- *$p_i = k_i$ and $D_i(p) = \mathsf{up}$.*

Solution type (O1) encodes a fixpoint, which is the proper solution of the discrete contraction map, while solution types (OV1) through (OV3) encode violations of the discrete contraction map property.

Solution type (OV1) witnesses a violation of the first property of a discrete contraction map, namely that each i-slice should have a unique fixpoint. A solution of type (OV1) gives two different points p and q in the same i-slice that are both fixpoints of that slice.

Solutions of type (OV2) witness violations of the first and second properties of a discrete contraction map. In these solutions we have two points p and q that are both fixpoints of their respective $(i-1)$-slices and are directly adjacent in an i-slice s. If there is a fixpoint r of the slice s, then this witnesses a violation of the second property of a discrete contraction map, which states that $D_i(p)$ and $D_i(q)$ should both point towards r, and clearly one of them does not. On the other hand, if slice s has no fixpoint, then p and q also witness this fact, since the fixpoint should be in-between p and q, which is not possible.

Solutions of type (OV3) consist of a point p that is a fixpoint of its $(i-1)$-slice but $D_i(p)$ points outside the boundary of the grid. These are clear violations of the second property, since $D_i(p)$ should point towards the fixpoint of the i-slice containing p, but that fixpoint cannot be outside the grid.

It is perhaps not immediately obvious that OPDC is a total problem. Ultimately we will prove this fact in the next section by providing a promise-preserving reduction from OPDC to UniqueEOPL. This will give us a proof of totality, and will also prove that, if the discrete contraction map has no violations, then it does indeed have a unique solution.

3.1 One-Permutation Discrete Contraction is in UniqueEOPL

In this section, we will show that One-Permutation Discrete Contraction lies in UniqueEOPL under promise-preserving reductions.

UFEOPL. Our reduction will make use of an intermediate problem that we call *unique forward EOPL*, which is a version of UniqueEOPL in which we only have a successor circuit S, meaning that no predecessor circuit P is given.

Definition 10 (UniqueForwardEOPL)**.** *Given a Boolean circuit $S : \{0,1\}^n \to \{0,1\}^n$ such that $S(0^n) \neq 0^n$ and a Boolean circuit $V : \{0,1\}^n \to \{0,1,\ldots,2^m-1\}$ such that $V(0^n) = 0$ find one of the following:*

(UF1) A point $x \in \{0,1\}^n$ such that $S(x) \neq x$ and either $S(S(x)) = S(x)$ or $V(S(x)) \leq V(x)$.

(UFV1) Two points $x, y \in \{0,1\}^n$, such that $x \neq y$, $x \neq S(x)$, $y \neq S(y)$, and either $V(x) = V(y)$ or $V(x) < V(y) < V(S(x))$.

(UFV2) Two points $x, y \in \{0,1\}^n$, such that $x \neq y$, x is a solution of type (UF1), $y \neq S(y)$, and $V(x) < V(y)$.

Without the predecessor circuit, this problem bears more resemblance to SinkOfDag than to EndOfPotentialLine. As in SinkOfDag, a bit-string x encodes a vertex if and only if $S(x) \neq x$, and an edge exists between vertices x and y if and only if $S(x) = y$ and $V(x) < V(y)$. The proper solution type (UF1) asks us to find a vertex that is a sink of the DAG, just as before.

The difference lies in the violations. Violations of type (UFV1) are the same as violation type (UV3) of UniqueEOPL. It asks for two vertices x and y that either have the same potential, or for which the potential of y lies strictly between the potential of x and the potential of $S(x)$.

Violations of type (UFV2) encode a break in the line. In UEOPL, we encoded breaks in the line by asking for a second source. In UFEOPL we lack a predecessor circuit, and so we have no way of verifying whether a vertex is a source. Instead, violations of type (UFV2) ask for two vertices x and y such that x is the end of a line, y is a vertex, and the potential of y is strictly larger than the potential of x. Any break in the line will produce such a pair.

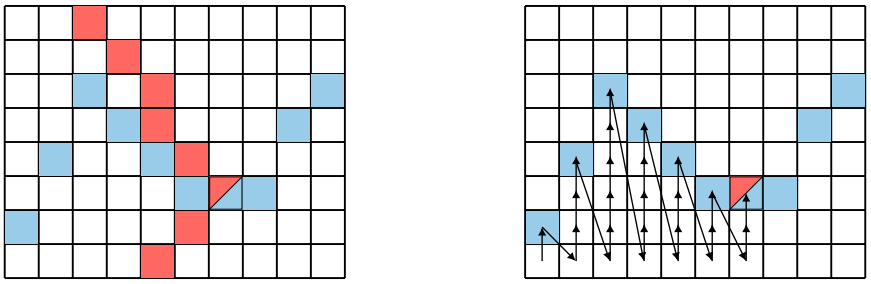

Figure 4: Left: The red and blue surfaces. Right: the path that we follow. This figure should be viewed in color.

Obviously every violation of type (UFV2) is also a solution of type (UF1), and so this type of violation may seem pointless at first glance. The reason that we need these violations is to prove that our reductions are promise-preserving. That is, if we reduce an instance of UEOPL that has a break in the line to UFEOPL, then we must ensure that the resulting UFEOPL instance contains a violation. Without violations of type (UFV2) there may only be solutions of type (UF1), and so our reduction would not be promise-preserving.

The UniqueForwardEOPL problem will play a crucial role in our reduction. We will reduce OPDC to it, and we will then reduce it to UniqueEOPL.

An illustration of the reduction. Before we discuss the formal definition of the construction, we first give some intuition by describing the reduction for the two-dimensional example shown in Figure 3. The reduction uses the notion of a *surface*. On the left side in Figure 4, we have overlaid the surfaces of the two direction functions from Figure 3. The surface of a direction function D_i is exactly the set of points $p \in P$ such that $D_i(p) = \mathsf{zero}$. The fixpoint p that we seek has $D_i(p) = \mathsf{zero}$ for all dimensions i, and so it lies at the intersection of these surfaces.

To reach the overall fixpoint, we walk along a path starting from the bottom-left corner, which is shown on the right-hand side of Figure 4. The path begins by walking upwards until it finds the blue surface. Once it has found the blue surface, then there are two possibilities: either we have found the overall fixpoint, in which case the line ends, or we have not found the overall fixpoint, and the red direction function tells us that the direction of the overall fixpoint is to the right.

If we have not found the overall fixpoint, then we move one step to the right, go back to the bottom of the diagram, and start walking upwards again. We keep repeating this until we find the overall fixpoint. This procedure gives us the line shown on the right-hand side of Figure 4.

The potential. How do we define a potential for this line? Observe that the dimension-two coordinates of the points on the line are weakly monotone, meaning that the line never moves to the left. Furthermore, for any dimension-two slice (meaning any slice in which the left/right coordinate is fixed), the dimension-one coordinate is monotonically increasing. So, if $p = (p_1, p_2)$ denotes any point on the line, if k denotes the maximum coordinate in either dimension, then the function

$$V(p_1, p_2) = k \cdot p_2 + p_1$$

is a function that monotonically increases along the line, which we can use as a potential function.

Uniqueness. To provide a promise-preserving reduction to UFEOPL, we must argue that the line is unique whenever the OPDC instance has no violations. Here we must carefully define what exactly a vertex on the line actually is, to ensure that no other line can exist. Specifically, we must be careful that only points that are to the left of the fixpoint are actually on the line, and that no "false" line exists to the right of the fixpoint.

Here we rely on the following fact: if the line visits a point with coordinate x in dimension 2, then it must have visited the point p on the blue surface in the slice defined by $x - 1$. Moreover, for that point p we must have $D_2(p) = \mathsf{up}$, which means that it is to the left of the overall fixpoint.

Using this fact, each vertex on our line will be a pair (p, q), where p is the current point that we are visiting, and q is either

- the symbol $-$, indicating that we are still in the first column of points, and we have never visited a point on the blue surface, or
- a point q that is on the blue surface, that satisfies $q_2 = p_2 - 1$ and $D_2(q) = \mathsf{up}$.

Hence the point q is always the last point that we visited on the blue surface, which provides a witness that we have not yet walked past the overall fixpoint.

When we finish walking up a column of points, and find the point on the blue surface, we overwrite q with the new point that we have found. This step is the reason why only a successor circuit can be given for the line, since the value that is overwritten cannot easily be computed by a predecessor circuit.

Violations. Our two-dimensional OPDC example does not contain any violations, but our reduction can still handle all possible violations in the OPDC instance. At a high level, there are two possible ways in which the reduction can go wrong if there are violations.

1. It is possible, that as we walk upwards in some column, we do not find a fixpoint, and our line will get stuck. This creates an end of line solution of type (UF1), which must be mapped back to an OPDC violation. In our two-dimensional example, this case corresponds to a column of points in which there is no point on the blue surface. However, if there is no point on the blue surface, then we will either

   - find two adjacent points p and q in that column with $D_1(p) = \mathsf{up}$, $D_1(q) = \mathsf{down}$ and $p_1 = q_1 + 1$, which is a solution of type (OV2), or
   - find a point p at the top of the column with $D_1(p) = \mathsf{up}$, or a point q at the bottom of the column with $D_1(q) = \mathsf{down}$. Both of these are solutions of type (OV3).

   There is also the similar case where we walk all the way to the right without finding an overall fixpoint, in which case we will find a point p on the right-hand boundary that satisfies $D_1(p) = \mathsf{zero}$ and $D_2(p) = \mathsf{up}$, which is a solution of type (OV3).

2. The other possibility is that there may be more than one point on the blue surface in some of the columns. This will inevitably lead to multiple lines, since if q and q' are both points on the blue surface in some column, and p is some point in the column to the right of p and q, then (p, q) and (p, q') will both be valid vertices on two different lines.

   These can show up as violations of type (UFV1), which we map back to solutions of type (OV1). Specifically, the points p and q, which are given as part of the two vertices, are both fixpoints of the same slice, which is exactly what (OV1) asks for.

Using these facts, we can show that if the OPDC instance contains no violations, then the resulting UFEOPL instance will also contain no violations. Hence, the reduction is promise-preserving.

The full reduction. Our reduction from OPDC to UniqueForwardEOPL generalizes the approach given above to d dimensions. We say that a point $p \in P$ is on the *i-surface* if $D_j(p) = \mathsf{zero}$ for all $j \leq i$. In our two-dimensional example we followed a line of points on the one-surface, in order to find a point on the two-surface. In between any two points on the one-surface, we followed a line of points on the zero-surface (every point is trivially on the zero-surface).

Our line will visit a sequence of points on the $(d-1)$-surface in order to find the point on the d-surface, which is the fixpoint. Between any two points on the $(d-1)$-surface the line visits a sequence of points on the $(d-2)$-surface, between any two points on the $(d-2)$-surface the line visits a sequence of points on the $(d-3)$-surface, and so on.

The line will follow the same pattern that we laid out in two dimensions. Every time we find a point on the i-surface, we remember it, increment our position in dimension i by 1, and reset our coordinates back to 0 for all dimensions $j < i$. Hence, a vertex will be a tuple $(p_0, p_1, \ldots, p_d)$, where each p_i is either

- the symbol $-$, indicating that we have not yet encountered a point on the i-surface, or
- the most recent point on the i-surface that we have visited.

This is a generalization of the witnessing scheme that we saw in two-dimensions.

The potential is likewise generalized so that the potential of a point p is proportional to $\sum_{i=1}^{d} k^i p_i$, where again k is some constant that is larger than the grid size. This means that progress in dimension i dominates progress in dimension j whenever $j < i$, which allows the potential to monotonically increase along the line.

We are also able to deal with all possible violations, using the ideas that we have described in the two-dimensional case. Our reduction is captured by the following Lemma, which is proved in Appendix B.1

Lemma 11. *There is a polynomial-time promise-preserving reduction from* OPDC *to* UniqueForwardEOPL.

From UniqueForwardEOPL to UniqueForwardEOPL+1. The next step of the reduction is to slightly modify the UniqueForwardEOPL instance, so that the potential increases by exactly one in each step. Specifically we define the following problem

Definition 12 (UniqueForwardEOPL+1)**.** *Given a Boolean circuits $S : \{0,1\}^n \to \{0,1\}^n$ such that $S(0^n) \neq 0^n$ and a Boolean circuit $V : \{0,1\}^n \to \{0,1,\ldots,2^m-1\}$ such that $V(0^n) = 0$ find one of the following:*

(UFP1) A point $x \in \{0,1\}^n$ such that $S(x) \neq x$ and either $S(S(x)) = S(x)$ or $V(S(x)) \neq V(x)+1$.

(UFPV1) Two points $x, y \in \{0,1\}^n$, such that $x \neq y$, $x \neq S(x)$, $y \neq S(y)$, and $V(x) = V(y)$.

(UFPV2) Two points $x, y \in \{0,1\}^n$, such that $x \neq y$, x is a solution of type (UFP1), $y \neq S(y)$, and $V(x) < V(y)$.

There are two differences between this problem and UniqueForwardEOPL. Firstly, an edge exists between x and y if and only if $S(x) = y$, and $V(y) = V(x)+1$, and this is reflected in the modified definition of solution type (UFP1). Secondly, solution type (UFPV1) has been modified to only cover the case where we have two vertices x and y that have the same potential. The case where $V(x) < V(y) < V(S(x))$ is not covered, since in this setting this would imply $V(S(x)) > V(x)+1$, which already gives us a solution of type (UFP1). Violations of type (UFPV2) are unchanged.

We can reduce from UniqueForwardEOPL to UniqueForwardEOPL+1, using the same techniques that we used in the reduction from EndOfPotentialLine to EndOfMeteredLine in Theorem 4. Specifically, the "dummy line" technique used in that theorem can also be applied to UniqueForwardEOPL instances, since it does not depend on the existence of a predecessor circuit. Note also that the technique does not change the number or type of solutions, so the reduction is promise-preserving. Hence, we get the following lemma.

Lemma 13. *There is a polynomial-time promise-preserving reduction from* UniqueForwardEOPL *to* UniqueForwardEOPL+1*.*

UniqueForwardEOPL+1 to UniqueEOPL. The final step of the proof is to reduce UniqueForwardEOPL+1 to UniqueEOPL. For this, we are able to build upon existing work. The following problem was introduced by Bitansky et al [3].

Definition 14 (SinkOfVerifiableLine [3])**.** *The input to the problem consists of a starting vertex $x_s \in \{0,1\}^n$, a target integer $T \leq 2^n$, and two boolean circuits $S : \{0,1\}^n \to \{0,1\}^n$, $W : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$. It is promised that, for every vertex $x \in \{0,1\}^n$, and every integer $i \leq T$, we have $W(x,i) = 1$ if and only if $x = S^{i-1}(x_s)$. The goal is to find the vertex $x_f \in \{0,1\}^n$ such that $W(x_f, T) = 1$.*

SinkOfVerifiableLine is intuitively very similar to UniqueForwardEOPL. In this problem, a single line is encoded, where as usual the vertices are encoded as bit-strings, and the circuit S gives the successor of each vertex. The difference in this problem is that the circuit W gives a way of verifying how far along the line a given vertex is. Specifically, $W(x,i) = 1$ if and only if x is the ith vertex on the line. Note that this is inherently a promise-problem, since if $W(x,i) = 1$ for some i, we have no way of knowing whether x is *actually* i steps along the line, without walking all of those steps ourselves.

It was shown by Hubáček and Yogev [34] that SinkOfVerifiableLine can be reduced in polynomial time to EndOfMeteredLine, and hence also to EndOfPotentialLine (via Theorem 4). Moreover, the resulting EndOfPotentialLine instance has a unique line, so this reduction also reduces SinkOfVerifiableLine to UniqueEOPL. It is easy to reduce the promise-version of UniqueForwardEOPL+1 to SinkOfVerifiableLine, since we can construct the circuit W so that $W(x,i) = 1$ if and only if $V(x) = i$.

However, the existing work only deals with the promise problem. Our contribution is to deal with violations. We show that, if one creates a SinkOfVerifiableLine instance from a UniqueForwardEOPL+1 instance, in the way described above, and applies the reduction of Hubáček and Yogev to produce a UniqueEOPL instance, then any violation can be mapped back to a solution in the original UniqueForwardEOPL+1 instance. Hence, we show the following lemma, which we will prove in Appendix B.2.

Lemma 15. *There is a polynomial-time promise-preserving reduction from* UniqueForwardEOPL *to* UniqueEOPL*.*

This completes the chain of promise-preserving reductions from OPDC to UniqueEOPL. Hence, we have shown the following theorem.

Theorem 16. OPDC *is in* UniqueEOPL *under polynomial-time promise-preserving reductions.*

It is worth pointing out that during this reduction, we have reduced SinkOfVerifiableLine to UniqueEOPL. Hence the query complexity and cryptographic lower bounds of hardness results shown by Bitansky et al. for SinkOfVerifiableLine [3], and shown to hold for EndOfMeteredLine and thus CLS by Hubáček and Yogev [34], also apply to UniqueEOPL.

4 One-Permutation Discrete Contraction is UniqueEOPL-hard

In this section we will show that One-Permutation Discrete Contraction is UniqueEOPL-complete, by giving a hardness result. Specifically, we give a reduction from UniqueEOPL to OPDC.

Modifying the line. The first step of the reduction is to slightly alter the UniqueEOPL instance. Specifically, we would like to ensure the following two properties.

1. Every edge increases the potential by exactly one. That is, $V(S(x)) = V(x) + 1$ for every vertex x.
2. The line has length exactly 2^n for some given integer n (where n here is not necessarily the same as the number of bits used to describe each vertex). More specifically, we ensure that if x is the end of any line then we have $V(x) = 2^n - 1$. The start of the line given in the problem has potential 0, this ensures that the length of that line is exactly 2^n, although other lines may be shorter.

We have already developed a technique for ensuring the first property in the reduction from EOPL to EOML in Theorem 4, which can be reused here. Specifically, we introduce a chain of dummy vertices between any pair of vertices x and y with $S(x) = y$ and $V(y) > V(x) + 1$. The second property can be ensured by choosing n so that 2^n is larger than the longest possible line in the instance. Then, at every vertex x that is the end of a line, we introduce a chain of dummy vertices

$$(x, 0) \to (x, 1) \to \cdots \to (x, 2^n - V(x) - 1),$$

where $V(x, i) = V(x) + i$. The vertex $e = (x, 2^n - V(x) - 1)$ will be the new end of the line, and note that $V(e) = 2^n - 1$ as required. From this, we get the following lemma.

Lemma 17. *Given a* UniqueEOPL *instance $L = (S, P, V)$, there is a polynomial-time promise-preserving reduction that produces a* UniqueEOPL *instance $L' = (S', P', V')$, where*

- *For every x and y with $y = S(x)$ and $x = P(y)$ we have $V(y) = V(x) + 1$, and*
- *There exists an integer n such that, x is a solution of type (U1) if and only if we have $V(x) = 2^n - 1$.*

For the remainder of this section, we will assume that we have a UniqueEOPL instance $L = (S, P, V)$ that satisfies the two extra conditions given by Lemma 17. We will use m to denote the bit-length of a vertex in L.

The set of points. We create an OPDC instance over a boolean hypercube with $m \cdot n$ dimensions, so our set of points is $P = \{0, 1\}^{mn}$. We will interpret each point $p \in P$ as a tuple $(v_1, v_2, \ldots, v_n)$, where each v_i is a bit-string of length m, meaning that each v_i can represent a vertex in L.

Note that we have specifically chosen P to be a Boolean hypercube. Although a conceptually simpler reduction might be obtained if we had instead chosen the space $\{0, 1, \ldots, 2^m - 1\}^n$, we use the cube for two reasons. Firstly, it shows that OPDC is hard, even when restricted to a Boolean hypercube. Secondly OPDC on a Boolean hypercube is very similar to the USO problem: USO can be thought of as "all-permutation" discrete contraction on a Boolean hypercube. We hope that our hardness result for OPDC on a cube might be the first step towards showing hardness for Unique-Sink-Orientation.

To understand the reduction, it helps to consider the case where there is a unique line in L. We know that this line has length exactly 2^n. The reduction repeatedly splits this line into two equal parts.

- Let L_1 denote the first half of L, which contains all vertices v with potential $0 \leq V(v) \leq 2^{n-1} - 1$.
- Let L_2 denote the second half of L, which contains all vertices v with potential $2^{n-1} \leq V(v) \leq 2^n - 1$.

Observe that L_1 and L_2 both contain exactly 2^{n-1} vertices.

The idea is to embed L_1 and L_2 into different sub-cubes of the $\{0,1\}^{mn}$ point space. As mentioned, each point $p \in P$ represents a tuple $(v_1, v_2, \ldots, v_n)$, and the line that we embed will be determined by the *last* element of this tuple. Let v be the vertex satisfying $V(v) = 2^{n-1}$, meaning that v is the first vertex of L_2.

- We embed L_2 into the sub-cube $(*, *, \ldots, *, v)$.
- We embed a copy of L_1 into each sub-cube $(*, *, \ldots, *, u)$ with $u \neq v$.

Note that this means that we embed a single copy of L_2, but many copies of L_1. Specifically, there are 2^m possibilities for the final element of the tuple. One of these corresponds to the sub-cube containing L_2, while $2^m - 1$ of them contain a copy of L_1.

The construction is recursive. So we split L_2 into two lines $L_{2,1}$ and $L_{2,2}$, each containing half of the vertices of L_2. If w is the vertex satisfying $V(w) = 2^{n-1} + 2^{n-2}$, which is the first vertex of $L_{2,2}$, then we embed a copy of $L_{2,2}$ into the sub-cube $(*, *, \ldots, *, w, v)$, where v is the same vertex that we used above, and we embed a copy of $L_{2,1}$ into each sub-cube $(*, *, \ldots, *, u, v)$, where $u \neq w$. Likewise L_1 is split into two, and embedded into the sub-cubes of $(*, *, \ldots, *, u)$ whenever $u \neq v$.

Given a vertex $(v_1, v_2, \ldots, v_n)$, we can view the bit-string v_n as choosing either L_1 or L_2, based on whether $V(v_n) = 2^{n-1}$. Once that decision has been made, we can then view v_{n-1} as choosing one half of the remaining line. Since the original line L has length 2^n, and we repeat this process n times, this means that at the end of the process we will be left with a line containing a single vertex. So in this way, a point $(v_1, v_2, \ldots, v_n)$ is a representation of some vertex in L, specifically the vertex that is left after we repeatedly split the line according to the choices made by v_n through v_1.

We can compute the vertex represented by any tuple in polynomial time. Moreover, given a slice $(*, *, \ldots, *, v_i, v_{i+1}, \ldots, v_n)$ that fixes elements i through n of the tuple, we can produce, in polynomial time, a UniqueEOPL instance corresponding to the line that is embedded in that slice. This is formalised in the following lemma, which is proved in Appendix C.1.

Lemma 18. *There are polynomial-time algorithms for computing the following two functions.*

- *The function* $\text{decode}(v_1, v_2, \ldots, v_n)$ *which takes a point in* P *and returns the corresponding vertex of* L*.*
- *The function* $\text{subline}(v_i, v_{i+1}, \ldots, v_n, L)$*, which takes bit-strings* v_i *through* v_n*, representing the slice,* $(*, *, \ldots, *, v_i, v_{i+1}, \ldots, v_n)$*, and the instance* $L = (S, P, V)$*, and returns a new* UniqueEOPL *instance* $L' = (S', P', V')$ *which represents the instance that is to be embedded into this slice.*

Although we have described this construction in terms of a single line, the two polynomial time algorithms given by Lemma 18 are capable of working with instances that contain multiple lines. In the case where there are multiple lines, there may be two or more bit-strings x and y with $V(x) = V(y) = 2^{n-1}$. In that case, we will embed second-half instances into $(*, *, \ldots, x)$ and $(*, *, \ldots, y)$. This is not a problem for the functions decode and subline, although this may lead to violations in the resulting OPDC instance that we will need to deal with.

The direction functions. The direction functions will carry out the embedding of the lines. Since our space is $\{0,1\}^{mn}$, we will need to define $m \cdot n$ direction functions D_1 through D_{mn}.

The direction functions $D_{m(n-1)+1}$ through D_{mn} correspond to the bits used to define v_n in a point $(v_1, v_2, \ldots, v_n)$. These direction functions are used to implement the transition between the first and second half of the line. For each point $p = (v_1, v_2, \ldots, v_n)$ we define these functions using the following algorithm.

1. In the case where $V(v_n) \neq 2^{n-1}$, meaning that decode(p) is a vertex in the first half of the line, then there are two possibilities.

   (a) If $V(\text{decode}(p)) = 2^{n-1} - 1$, meaning that p is the last vertex on the first half of the line, then we orient the direction function of dimensions $(n-1) \cdot m + 1$ through nm towards the bit-string given by $S(\text{decode}(p))$. This captures the idea that once we reach the end of the first half of the line, we should then move to the second half, and we do this by moving towards the sub-cube $(*, *, \ldots, *, S(\text{decode}(p))$. So for each i in the range $m(n-1) + 1 \leq i \leq mn$ we define

   $$D_i(p) = \begin{cases} \mathsf{up} & \text{if } p_i = 0 \text{ and } S(\text{decode}(p))_i = 1, \\ \mathsf{down} & \text{if } p_i = 1 \text{ and } S(\text{decode}(p))_i = 0, \\ \mathsf{zero} & \text{otherwise.} \end{cases}$$

   (b) If $V(\text{decode}(p)) \neq 2^{n-1} - 1$, then we orient everything towards 0. Specifically, we set

   $$D_i(p) = \begin{cases} \mathsf{down} & \text{if } p_i = 1, \\ \mathsf{zero} & \text{if } p_i = 0. \end{cases}$$

   This is an arbitrary choice: our reduction would work with any valid direction rules in this case.

2. If $V(v_n) = 2^{n-1}$, then we are in the second half of the line. In this case we set $D_i(p) = \mathsf{zero}$ for all dimensions i in the range $(n-1) \cdot m + 1 \leq i \leq mn$. This captures the idea that, once we have entered the second half of the line, we should never leave it again.

We use the same idea recursively to define direction functions for all dimensions $i \leq m(n-1)$. This gives us a family of polynomial-time computable direction functions $\mathcal{D} = (D_i)_{i=1,\ldots,mn}$. Let $p = (v_1, v_2, \ldots, v_n)$ be a point. For the dimensions j in the range $m(i-1) + 1 \leq j \leq mi$ we use the following procedure to determine D_j. Let $L' = (S', P', V') = \text{subline}(v_{i+1}, v_{i+2}, \ldots, v_n, L)$.

1. In the case where $V'(v_i) \neq 2^{i-1}$, meaning that decode(p) is a vertex in the first half of L', then there are two possibilities.

   (a) If $V'(\text{decode}(p)) = 2^{i-1} - 1$, meaning that p is the last vertex on the first half L', then we orient the direction function towards the bit-string given by $S(\text{decode}(p))$. So we set

   $$D_j(p) = \begin{cases} \mathsf{up} & \text{if } p_j = 0 \text{ and } S(\text{decode}(p)) = 1, \\ \mathsf{down} & \text{if } p_j = 1 \text{ and } S(\text{decode}(p)) = 0, \\ \mathsf{zero} & \text{otherwise.} \end{cases}$$

(b) If $V(\text{decode}(p)) \neq 2^{n-1}-1$, then we orient everything towards 0 by setting

$$D_j(p) = \begin{cases} \mathsf{down} & \text{if } p_j = 1, \\ \mathsf{zero} & \text{if } p_j = 0. \end{cases}$$

2. If $V'(v_i) = 2^{i-1}$, then we are in the second half of the line. In this case we set $D_i(p) = \mathsf{zero}$.

The proof. We have now defined P and $\mathcal{D}$, so we have an OPDC instance. We must now argue that the reduction is correct. The intuitive idea is as follows. If we are at some point $p \in P$, and $\text{decode}(p) = v$ is a vertex that is not the end of a line, then there is some direction function D_i such that $D_i(p) \neq \mathsf{zero}$. We can see this directly for the case where $V(\text{decode}(p)) = 2^{n-1}-1$, since the direction functions on dimensions $m(n-1)+1$ through mn will be oriented towards $S(V(\text{decode}(p))$, and so p will not be a solution.

As we show in the proof, the same property holds for all other vertices in the middle of the line. The end of the line will be a solution, because it will be encoded by the point $p = (v_1, v_2, \ldots, v_n)$, where each v_i is the first vertex on the second half of the line embedded in the sub-cube. Our direction functions ensure that $D_j(p) = \mathsf{zero}$ for all j for this point.

Our reduction must also deal with violations. Violations of type (OV3) are impossible by construction. In violations of type (OV1) and (OV2) we have two points p and q that are in the same i-slice. Here we specifically use the fact that violations can only occur within i-slices. Note that an i-slice will fix the last $mn-i$ bits of the tuple $(v_1, v_2, \ldots, v_n)$, which means that there will be an index j such that all v_l with $l > j$ are fixed. This allows us to associate the slice with the line $L' = \text{subline}(v_{j+1}, v_{j+2}, \ldots, v_n)$, and we know that both p and q encode vertices of L'. In both cases, we are able to recover two vertices in L' that have the same potential, and these vertices also have the same potential in L. So we get a solution of type (UV3).

From this, we obtain the following lemma, which is proved formally in Appendix C.2.

Lemma 19. *There is a polynomial-time promise-preserving reduction from* UniqueEOPL *to* OPDC.

Thus, we have shown the following theorem.

Theorem 20. OPDC *is* UniqueEOPL*-complete under promise-preserving reductions, even when the set of points P is a hypercube.*

Since we have shown bidirectional promise-preserving reductions between OPDC and UniqueEOPL, we also get that the promise version of OPDC is complete for PromiseUEOPL.

5 UniqueEOPL containment results

5.1 Unique Sink Orientations

Unique sink orientations. Let $C = \{0,1\}^n$ be an n-dimensional hypercube. An *orientation* of C gives a direction to each edge of C. We formalise this as a function $\Psi : C \to \{0,1\}^n$, that assigns a bit-string to each vertex of C, with the interpretation that the i-th bit of the string gives an orientation of the edge in dimension i. More precisely, for each vertex $v \in C$ and each dimension i, let u be the vertex that is adjacent to v in dimension i.

- If $\Psi(v)_i = 0$ then the edge between v and u is oriented towards v.

- If $\Psi(v)_i = 1$ then the edge between v and u is oriented towards u.

Note that this definition does not insist that v and u agree on the orientation of the edge between them, meaning that $\Psi(v)_i$ and $\Psi(u)_i$ may orient the edge in opposite directions. However, this will be a violation in our set up, and a proper orientation should be thought of as always assigning a consistent direction to each edge.

A *face* is a subset of C in which some coordinates have been fixed. This can be defined using the same notation that we used for slices in OPDC. So a face $f = (f_1, f_2, \ldots, f_n)$, where each f_i is either 0, 1, or $*$, and the sub-cube defined by f contains every vertex $v \in C$ such that $v_i = f_i$ whenever $f_i \neq *$.

A vertex $v \in C$ is a *sink* if all of the edges of v are directed towards v, meaning that $\Psi(v)_i = 0$ for all i. Given a face f, a vertex v is a *sink of* f if it is a sink of the sub-cube defined by f, meaning that $\Psi(v)_i = 0$ whenever $f_i = *$.

A *unique sink orientation* (USO) is an orientation in which *every* face has a unique sink. Since $f = (*, *, \ldots, *)$ is a face, this also implies that the whole cube has a unique sink, and the USO problem is to find the unique sink.

Placing the problem in TFNP. The USO property is quite restrictive, and there are many orientations that are not USOs. Indeed, the overall cube may not have a sink at all, or it may have multiple sinks, and this may also be the case for the other faces of the cube. Fortunately, Szabó and Welzl have pointed out that if an orientation is not a USO, then there is a succinct witness of this fact [59].

Let $v, u \in C$ be two distinct vertices. We have that Ψ is a USO if and only if, for any distinct vertices $u, v \in C$, there exists some dimension i such that $v_i \neq u_i$ and $\Psi(v)_i \neq \Psi(u)_i$. Put another way, this means that if we restrict the orientation only to the sub-cube defined by any two vertices v and u, then the orientations of v and u must be different on that sub-cube. If one uses $\oplus$ to denote the XOR operation on binary strings, and $\cap$ to denote the bit-wise and-operation, then this condition can be written concisely as

$$(v \oplus u) \cap (\Psi(v) \oplus \Psi(u)) \neq 0^n.$$

Note that this condition also ensures that the orientation is consistent, since if v and u differ in only a single dimension, then the condition states that they must agree on the orientation of the edge between them.

We use this condition to formulate the USO problem as a problem in TFNP.

Definition 21 (Unique-Sink-Orientation)**.** *Given an orientation function* $\Psi : \{0,1\}^n \to \{0,1\}^n \cup \{-\}$ *find one of the following.*

(US1) A point $v \in \{0,1\}^n$ *such that* $\Psi(v)_i = 0$ *for all* i*.*

(USV1) A point $v \in \{0,1\}^n$ *such that* $\Psi(v) = -$*.*

(USV2) Two points $v, u \in \{0,1\}^n$ *such that* $v \neq u$ *and* $(v \oplus u) \cap (\Psi(v) \oplus \Psi(u)) = 0^n$*.*

Note that this formulation of the problem allows the orientation function to decline to give an orientation for some vertices, and this is indicated by setting $\Psi(v) = -$. Any such vertex is a violation of type (USV1). While this adds nothing interesting to the USO problem, we will use this in Section 5.3 when we reduce P-LCP *to* USO, since in some cases the reduction may not be able to produce an orientation at a particular vertex.

Assuming that there are no violations of type (USV1), it is easy to see that the problem is total. This is because every USO has a sink, giving a solution of type (US1), while every orientation that is not a USO has a violation of type (USV2).

We remark that, although violations of type (USV1) are useful for expositional purposes, they are not actually necessary, since they can all be mapped to global sinks instead. We prove the correctness of this in the following lemma.

Lemma 22. *There is a polynomial-time promise-preserving reduction from* Unique-Sink-Orientation *to a variant of* Unique-Sink-Orientation *in which there are no violations of type (USV1).*

Proof. Let Ψ be an instance of Unique-Sink-Orientation. We create an instance Ψ' in the following way. For each point $v \in \{0,1\}^n$

- If $\Psi(v) \neq -$, then $\Psi'(v) = \Psi(v)$.
- Otherwise, $\Psi'(v) = 0^n$.

Observe that Ψ' cannot have a solution of type (USV1) by construction. We argue that this reduction is correct.

- If $v \in \{0,1\}^n$ is a solution of type (US1) for Ψ', then either $\Psi(v) = 0^n$, meaning that v is a solution of type (US1) for Ψ, or $\Psi(v) = -$, meaning that v is a solution of type (USV1) for Ψ.
- If $v, u \in \{0,1\}^n$ are a solution of type (USV2) for Ψ', and either $\Psi(v) = -$ or $\Psi(u) = -$, then we have a solution of type (USV1) for Ψ. If $\Psi(v) \neq -$ and $\Psi(u) \neq -$, then we have a solution of type (USV2) for Ψ.

To see that the reduction is promise-preserving, observe that solutions of type (US1) for Ψ are only ever mapped to solutions of type (US1) for Ψ'. □

Placing the problem in UniqueEOPL. We show that the problem lies in UniqueEOPL by providing a promise-preserving reduction from Unique-Sink-Orientation to OPDC. The reduction is actually not too difficult, because when the point set for the OPDC instance is a hypercube, the OPDC problem can be viewed as a less restrictive variant of USO. Specifically, USO demands that *every* face has a unique sink, while OPDC only requires that the i-slices should have unique sinks.

The reduction creates an OPDC instance on the same set of points, meaning that $P = C$. The direction functions simply follow the orientation given by Ψ. Specifically, for each $v \in P$ and each dimension i we define

$$D_i(v) = \begin{cases} \mathsf{zero} & \text{if } \Psi(v)_i = 0, \\ \mathsf{up} & \text{if } \Psi(v)_i = 1 \text{ and } v_i = 0, \\ \mathsf{down} & \text{if } \Psi(v)_i = 1 \text{ and } v_i = 1. \end{cases}$$

If $\Psi(v) = -$, then we instead set $D_i(v) = \mathsf{zero}$ for all i.

To prove that this is correct, we show that every solution of the OPDC instance can be mapped back to a solution of the USO instance. Any fixpoint of the OPDC instance satisfies $D_i(v) = \mathsf{zero}$ for all i, which can only occur if v is a sink, or if $\Psi(v) = -$. The violation solutions of OPDC can be used to generate a pair of vertices that constitute a (USV2) violation. We can prove the following lemma.

Lemma 23. *There is a polynomial-time promise-preserving reduction from* Unique-Sink-Orientation *to* OPDC.

Proof. Every solution of the OPDC instance can be mapped back to a solution of the USO instance. We prove this by enumerating all possible types of solution.

1. In solutions of type (O1) we are given a point $p \in P$ such that $D_i(p) = \mathsf{zero}$ for all i. If $\Psi(p) \neq -$, then this means that p is a sink, and so it is a solution of type (US1). If $\Psi(p) = -$, then this means that p is a solution of type (USV1).

2. In solutions of type (OV1), we have an i-slice s and two points $p, q \in P_s$ with $p \neq q$ such that $D_j(p) = D_j(q) = \mathsf{zero}$ for all $j \leq i$. If $\Psi(p) = -$ or $\Psi(q) = -$, then we have a solution of type (USV1). Otherwise, this means that p and q are both sinks of the face corresponding to s, and specifically this means that they have the same out-map on this face. So this gives us a solution of type (USV2).

3. In solutions of type (OV2) we have an i-slice s and two points $p, q \in P_s$ such that

   - $D_j(p) = D_j(q) = \mathsf{zero}$ for all $j < i$,
   - $p_i = q_i + 1$, and
   - $D_i(p) = \mathsf{down}$ and $D_i(q) = \mathsf{up}$.

   Note that in this case we must have $\Psi(p) \neq -$ and $\Psi(q) \neq -$. If we restrict the cube to the face defined by s, note that $\Psi(p)_j = \Psi(q)_j = 0$ for all dimensions $j \neq i$, and $\Psi(p)_i = \Psi(q)_i = 1$. Hence p and q have the same out-map on the face defined by s, which gives us a solution of type (USV2).

4. Solutions of type (OV3) are impossible for the instance produced by our reduction. In these solutions we have a point p with $p_i = 0$ and $D_i(p) = \mathsf{down}$, or a point q with $q_j = 1$ and $D_j(q) = \mathsf{up}$. Since our reduction never does this, solutions of type (OV3) cannot occur.

To see that the reduction is promise-preserving, it suffices to note that solutions of type (US1) are only ever mapped onto solutions of type (O1). Thus, if the USO instance has no violations, then the resulting OPDC instance also has no violations. This completes the proof of Lemma 23. □

Thus we have shown the following theorem.

Theorem 24. Unique-Sink-Orientation *is in* UniqueEOPL *under promise-preserving reductions.*

This proves the following new facts about the complexity of finding the sink of a USO.

Corollary 25. Unique-Sink-Orientation *is in* PPAD, PLS, *and* CLS.

5.2 Piecewise Linear Contraction Maps

In this section, we show that PL-Contraction, the problem of finding a fixpoint of a *piecewise linear* contraction map, lies in UniqueEOPL. Specifically, we study contraction maps where the function f is given as a LinearFIXP *circuit*, which is an arithmetic circuit comprised of $\max, \min, +, -$, and $\times\zeta$ (multiplication by a constant) gates [17]. Hence, a LinearFIXP circuit defines a *piecewise linear* function.

Violations. Not every function f is contracting, and the most obvious way to prove that f is not contracting is to give a pair of points x and y that satisfy $\|f(x) - f(y)\|_p > c \cdot \|x - y\|_p$, which directly witness the fact that f is not contracting.

However, when we discretize PL-Contraction in order to reduce it to OPDC, there are certain situations in which we have a convincing proof that f is not contracting, but no apparent way to actually produce a violation of contraction. In fact, the discretization itself is non-trivial, so we will explain that first, and then define the type of violations that we will use.

The reduction. We are given a function $f : [0,1]^n \to [0,1]^n$, that is purported to be contracting with contraction factor c in the ℓ_p norm. We will produce an OPDC instance by constructing the point set P, and a family of direction functions $\mathcal{D}$.

The most complex step of the reduction is to produce an appropriate set of points P for the OPDC instance. This means we need to choose integers k_1, k_2, through k_d in order to define the point set $P(k_1, k_2, \ldots, k_d)$, where we recall that this defines a grid of integers, where each dimension i can take values between 0 and k_i. We will describe the method for picking k_1 through k_d after we have specified the rest of the reduction.

The direction functions will simply follow the directions given by f. Specifically, for every point $p \in P(k_1, k_2, \ldots, k_d)$, let p' be the corresponding point in $[0,1]^n$, meaning that $p'_i = p_i/k_i$ for all i. For every dimension i we define the direction function D_i so that

- if $f(p')_i > p'_i$ then $D_i(p) = \mathsf{up}$,
- if $f(p')_i < p'_i$ then $D_i(p) = \mathsf{down}$, and
- if $f(p')_i = p'_i$ then $D_i(p) = \mathsf{zero}$.

In other words, the function D_i simply checks whether $f(p')$ moves up, down, or not at all in dimension i. This completes the specification of the family of direction functions $\mathcal{D}$.

We must carefully choose k_1 through k_d to ensure that the fixpoint of f is contained within the grid. In fact, we need a stronger property: for every i-slice of the grid, if f has a fixpoint in that i-slice, then it should also appear in the grid. Recall that $p \in P$ is a fixpoint of some slice s if $D_i(p) = \mathsf{zero}$ for every i for which $s_i = *$. We can extend this definition to the continuous function f as follows: a point $x \in [0,1]^d$ is a fixpoint of s if $(x - f(x))_i = 0$ for all i for which $s_i = *$, where we now interpret s as specifying that $x_i = s_i/k_i$ whenever $s_i \neq *$. We are able to show the following lemma, whose proof appears in Appendix D.1.

Lemma 26. *There exists integers $(k_1, k_2, \ldots, k_d)$ such that for every i-slice s we have that if $x \in [0,1]^d$ is a fixpoint of s according to f, then there exists a point $p \in P(k_1, k_2, \ldots, k_d)$ such that*

- *p is a fixpoint of $\mathcal{D}$, and*
- *$p_i = k_i \cdot x_i$ for all i where $s_i = *$.*

Moreover, the number of bits needed to write down each k_i is polynomial in the number of bits needed to write down f.

This lemma states that we can pick the grid size to be fine enough so that all fixpoints of f in all i-slices are contained within the grid. The proof of this is actually quite involved, and relies crucially on the fact that we have access to a LinearFIXP representation of f. From this, we can compute upper bounds on the bit-length of any point that is a fixpoint of f. We also rely on the fact that we only need to consider i-slices, because our proof fixes the grid-widths one dimension at a time, starting with dimension d and working backwards.

The extra violation. The specification of our reduction is now complete, but we have still not fully defined the original problem, because we need to add an extra violation. The issue arises with solutions of type (OV2), where we have an i-slice s and two points p, q in s such that

- $D_j(p) = D_j(q) = \mathsf{zero}$ for all $j < i$,
- $p_i = q_i + 1$, and
- $D_i(p) = \mathsf{down}$ and $D_i(q) = \mathsf{up}$.

This means that p and q are both fixpoints of their respective $(i-1)$-slices, and are directly adjacent to each other in dimension i. We are able to show, in this situation, that if f is contracting, then f has a fixpoint for the slice s, and it must lie between p and q. We can show the following lemma, which is proved formally in Appendix D.2.

Lemma 27. *If f is contracting, and we have two points p and q that are a violation of type (OV2), then there exists a point $x \in [0,1]^n$ in the slice s that satisfies all of the following.*

- $(x - f(x))_j = 0$ *for all $j \leq i$, meaning that x is a fixpoint of the slice s, and*
- $q_i < k_i \cdot x_i < p_i$, *meaning that x lies between p and q in dimension i.*

So if we have an (OV2) violation, and if f is contracting, then Lemma 27 implies that there is a fixpoint x of the slice s that lies strictly between p and q in dimension i. However, Lemma 26 says that all fixpoints of s lie in the grid, and since p and q are directly adjacent adjacent in the grid in dimension i, there is no room for x, so it cannot exist. The only way that this contradiction can be resolved is if f is not actually contracting.

Hence, (OV2) violations give us a concise witness that f is not contracting. But the points p and q themselves may satisfy the contraction property. While we know that there must be a violation of contraction *somewhere*, we are not necessarily able to compute such a violation in polynomial time. To resolve this, we add the analogue of an (OV2) violation to the contraction problem.

Definition 28 (PL-Contraction)**.** *Given a* LinearFIXP *circuit computing $f : [0,1]^d \to [0,1]^d$, a constant $c \in (0,1)$, and a $p \in \mathbb{N} \cup \{\infty\}$, find one of the following.*

(CM1) A point $x \in [0,1]^d$ such that $f(x) = x$.

(CMV1) Two points $x, y \in [0,1]^d$ such that $\|f(x) - f(y)\|_p > c \cdot \|x - y\|_p$.

(CMV2) A point $x \in [0,1]^d$ such that $f(x) \notin [0,1]^d$.

(CMV3) An i-slice s and two points $x, y \in [0,1]^d$ in s such that

- $(f(x) - x)_j = (f(y) - y)_j = 0$ *for all $j < i$,*
- $k_i \cdot x_i = k_i \cdot y_i + 1$, *where k_i is the integer given by Lemma 26 for the* LinearFIXP *circuit that computes f, and*
- $f(x)_i < x_i$ *and* $f(y)_i > y_i$.

Solution type (CM1) asks us to find a fixpoint of the map f, and there are two types of violation. Violation type (CMV1) asks us to find two points x and y that prove that f is not contracting with respect to the ℓ_p norm. Violation type (CMV2) asks us to find a point that f does not map to $[0,1]^d$. Note that this second type of violation is necessary to make the problem total, because it is possible that f is a contraction map, but the unique fixpoint of f does not lie in $[0,1]^d$.

Violations of type (CMV3) are the direct translation of (OV2) violations to contraction. Note that if f actually is contracting, and has a fixpoint in $[0,1]^n$, then no violations can exist. For (CMV3) violations, this fact is a consequence of Lemmas 26 and 27.

Correctness of the reduction. To prove that the reduction is correct, we must show that all solutions of the OPDC instance given by P and $\mathcal{D}$ can be mapped back to solutions of the original instance. Solutions of type (O1) give us a point p such that $D_i(p) = \mathsf{zero}$ for all i, which by definition means that the point corresponding to p is a fixpoint of f. Violations of type (OV1) give us two points that are both fixpoints of the same slice s, which also means that they are both fixpoints of the slice s according to f, and it is not difficult to show that these two points violate contraction in an ℓ_p norm. Violations of type (OV3) are points that attempt to leave the $[0,1]^d$, and so give us a solution of type (CMV2). Violations of type (OV2) map directly to violations of type (CMV3), as we have discussed. So we have the following lemma, which is proved formally in Appendix D.3.

Lemma 29. *There is a polynomial-time promise-preserving reduction from* PL-Contraction *to* OPDC.

Theorem 30. PL-Contraction *is in* UniqueEOPL *under promise-preserving reductions.*

5.3 The P-Matrix Linear Complementarity Problem

In this section, we reduce P-LCP to Unique-Sink-Orientation. We begin by defining the problem.

Definition 31 (LCP $(M, \boldsymbol{q})$)**.** *Given a matrix $M \in \mathbb{R}^{d\times d}$ and vector $\boldsymbol{q} \in \mathbb{R}^{d\times 1}$, find a $\mathbf{y} \in \mathbb{R}^{d\times 1}$ s.t.:*

$$\mathbf{w} = M\mathbf{y} + \boldsymbol{q} \geq 0; \quad \mathbf{y} \geq 0; \quad y_i \cdot w_i = 0, \ \forall i \in [d]. \tag{1}$$

In general, deciding whether an LCP has a solution is NP-complete [9]. A matrix M is a P-matrix if and only if every principal minor of M is positive. If M is a P-matrix, then the LCP $(M, \boldsymbol{q})$ has a *unique* solution for all $\boldsymbol{q} \in \mathbb{R}^{d\times 1}$.

The problem of checking if a matrix is a P-matrix is coNP-complete [11], so we cannot expect to be able to verify that an LCP instance $(M, \boldsymbol{q})$ is actually defined by a P-matrix M. Instead, we use succinct witnesses that M is not a P-matrix as a violation solution, which allows us to define total variants of the P-LCP problem that lie in TFNP, as first done by Megiddo [42, 43]. This approach has previously been used to place the P-matrix problem in PPAD and CLS [13, 49].

It is well known that a matrix M is a P-matrix *if and only if* for all $\boldsymbol{q} \in \mathbb{R}^{d\times 1}$, the LCP $(M, \boldsymbol{q})$ has a unique solution [10]. However, this characterization of a P-matrix is not directly useful for defining succinct violations: while two distinct solutions would be a succinct violation, there is no corresponding succinct witness for the case of no solutions. Next we introduce the three well-known succinct witnesses for M not being a P-matrix that we will use.

First, we introduce some further required notation. Restating (1), the LCP problem $(M, \boldsymbol{q})$ seeks a pair of non-negative vectors $(\mathbf{y}, \mathbf{w})$ such that:

$$I\mathbf{w} - M\mathbf{y} = \boldsymbol{q} \quad \text{and } \forall i \in [d] \text{ we have } \mathbf{y}_i \cdot \mathbf{w}_i = 0. \tag{2}$$

If $\boldsymbol{q} \geq 0$, then $(\mathbf{y}, \mathbf{w}) = (0, \boldsymbol{q})$ is a *trivial solution*. We identify a solution $(\mathbf{y}, \mathbf{w})$ with the set of components of $\mathbf{y}$ that are positive: let $\alpha = \{i \mid \mathbf{y}_i > 0, \ i \in [d]\}$ denote such a set of "basic variables". Going the other way, to check if there is a solution that corresponds to a particular $\alpha \subseteq [d]$, we try to perform a *principal pivot transformation*, re-writing the LCP by writing certain variables $\mathbf{y}_i$ as $\mathbf{w}_i$, and checking if in this re-written LCP there exists the trivial solution $(\mathbf{y}', \mathbf{w}') = (0, \boldsymbol{q}')$. To that end, we construct a $d \times d$ matrix A_α, where the ith column of A_α is defined as follows. Let e_i

denote the ith unit column vector in dimension d, and let $M_{\cdot i}$ denote the ith column of the matrix M.

$$(A_\alpha)_{\cdot i} := \begin{cases} -M_{\cdot i} & \text{if } i \in \alpha, \\ e_i & \text{if } i \notin \alpha. \end{cases} \tag{3}$$

Then α corresponds to an LCP solution if A_α is non-singular and, the "new $\boldsymbol{q}$" i.e., $(A_\alpha^{-1}\boldsymbol{q})$, is non-negative. For a given $\alpha \subseteq [n]$, we define $\text{out}(\alpha) := -$ if $\det(A_\alpha) = 0$; note that this will not happen if M is really a P-matrix, but our general treatment here is made to deal with the non-promise problem, in which case a zero determinant will correspond to a violation[4]. If $\det(A_\alpha) \neq 0$, we define $\text{out}(\alpha)$ as a bit-string in $\{0,1\}^d$ as follows:

$$(\text{out}(\alpha))_i = \begin{cases} 1 & \text{if } (A_\alpha^{-1}\boldsymbol{q})_i < 0, \\ 0 & \text{if } (A_\alpha^{-1}\boldsymbol{q})_i \geq 0. \end{cases} \tag{4}$$

With this notation, a subset $\alpha \subseteq [d]$ corresponds to a solution of the LCP if $\text{out}(\alpha) = 0^d$. We will use $\text{out}(\alpha)$ both to define a succinct violation of the P-matrix property, and in our promise-preserving reduction from P-LCP to Unique-Sink-Orientation.

We are now ready to define the P-LCP problem. Let $\text{char}(v)$ for $v \subseteq [d]$ denote the characteristic vector of v, i.e., $(\text{char}(v))_i = 1$ if $i \in v$ and 0 otherwise. As in the section on USOs, when we write $\oplus$ and $\cap$, here we mean the bit-wise XOR and bit-wise and-operations on bit-strings.

Definition 32 (P-LCP)**.** *For a given LCP $(M, \boldsymbol{q})$ in dimension d, each of the following provides either a solution, or a polynomial-size witness that M is not a P-matrix:*

(P1) $\mathbf{y} \in \mathbb{R}^{d\times 1}$ *that satisfies (1).*

(PV1) A set $\alpha \subseteq [d]$ such that the corresponding principal minor is non-positive, i.e., $\det(M_{\alpha\alpha}) \leq 0$.

(PV2) Two distinct sets $\alpha, \beta \subset [d]$, with $(\text{char}(\alpha) \oplus \text{char}(\beta)) \cap (\text{out}(\alpha) \oplus \text{out}(\beta)) = 0^d$.

Solutions of type (P1) are solutions to the LCP problem. Violation (PV1) correspond to the standard definition of a P-matrix as having all positive principal minors. Megiddo [42, 43] used this violation to place the P-LCP problem in TFNP. The same violation was then used by Papadimitriou to put P-LCP in PPAD, because Lemke's algorithm is a PPAD-type complementary pivoting algorithm, which inspired PPAD, and it will return a non-positive principal minor if it fails to find an LCP solution.

The other violation, (PV2), follows from the work of Stickney and Watson [58], who showed that P-matrix LCPs give rise to USOs, and from the work of Szabó and Welzl [59, Lemma 2.3], who showed that this condition characterizes the "outmap" of USOs, as discussed in Section 5.1, hence the name of the function being "out".

It should be noted that using either (PV1) or (PV2) violations by themselves is already enough to make the problem total, since if M is not a P-matrix, then violations of both types must exist. However, our reduction to Unique-Sink-Orientation will sometimes produce violations of type (PV1), and sometimes produce violations of type (PV2), so we need both of them. Moreover, it is not immediately apparent how to convert violations of one type to the other in polynomial time, and it is conceivable that allowing different sets of violations changes the complexity of the problem. We leave it as further work to further explore this.

[4] We could also define $\text{out}(\alpha) := -$ if $\det(A_\alpha) < 0$, since then we also get a P-matrix violation, however we choose to still perform a principal pivot transformation in this case since, ceteris paribus, it seems reasonable to prefer an LCP solution to a P-matrix violation when we reduce P-LCP to USO.

Reduction from P-LCP to Unique-Sink-Orientation. We are now ready to present a reduction to USO, which is due to Stickney and Watson [58]. The reduction is simple, and we present it in full detail here. For the LCP instance $\mathcal{I} = (M, \boldsymbol{q})$ in dimension d, we produce an instance $\mathcal{U}$ of Unique-Sink-Orientation also in dimension d.

We first need to deal with the possibility that $\boldsymbol{q}$ is degenerate. A P-matrix LCP has a degenerate $\boldsymbol{q}$ if $A_\alpha^{-1} \cdot \boldsymbol{q}$ has a zero entry for some $\alpha \subseteq [d]$. To ensure that this does not present a problem for our reduction, we use a standard technique known as lexicographic perturbation [10, Section 4.2]: In the reduction that follows we assume that we are using such a degeneracy resolution scheme.

The reduction associates each vertex v of the resulting USO with a set $\alpha(v)$ of basic variables for the LCP, and then uses $\text{out}(\alpha)$ as the outmap at v. In detail, for a vertex $v \in \{0,1\}^d$ of $\mathcal{U}$, we define $\alpha(v) = \{i \mid v(i) = 1\}$, and set $\Psi(v) = \text{out}(\alpha(v))$. It immediately follows that:

- A solution of type (US1) in $\mathcal{U}$ is a solution of type (P1) in $\mathcal{I}$.
- A solution of type (USV1) in $\mathcal{U}$ is a solution of type (PV1) in $\mathcal{I}$.
- A solution of type (USV2) in $\mathcal{U}$ is a solution of type (PV2) in $\mathcal{I}$.

If M is actually a P-matrix then $\mathcal{U}$ will have exactly one solution of type (US1), and no violation solutions [58]. Thus our reduction is promise-preserving, and we obtain the following.

Theorem 33. *There is a polynomial-time promise-preserving reduction from* P-LCP *to* Unique-Sink-Orientation.

6 Future directions

A clear direction for future work is to show that further problems are UniqueEOPL-complete. We have several conjectures.

Conjecture 1. *USO is hard for* UniqueEOPL.

We think that, among our three motivating problems, USO is the most likely to be UniqueEOPL-complete. Our hardness proof for OPDC already goes some way towards proving this, since we showed that OPDC was hard even when the set of points is a hypercube. The key difference between OPDC on a hypercube and USO is that OPDC only requires that the faces corresponding to i-slices should have unique sinks, while USO requires that all faces should have unique sinks.

Conjecture 2. *Piecewise-Linear Contraction in an ℓ_p norm is hard for* UniqueEOPL.

Our OPDC hardness result also goes some way towards showing that Piecewise-Linear Contraction is hard, however there are more barriers to overcome here. In addition to the i-slice vs. all slice issue, we would also need to convert the discrete OPDC problem to the continuous contraction problem. Converting discrete problems to continuous fixpoint problems has been well-studied in the context of PPAD-hardness reductions [12, 44], but here the additional challenge is to carry out such a reduction while maintaining the contraction property.

Aside from hardness, we also think that the relationship between Contraction and USO should be explored further. Our formulation of the OPDC problem exposes significant similarities between the two problems, which until this point have not been recognised. Can we reduce USO to Contraction in polynomial time?

Conjecture 3. *P-LCP is hard for* UniqueEOPL.

Of all of our conjectures, this will be the most difficult to prove. Since P-LCP reduces to USO, the hardness of USO should be resolved before we attempt to show that P-LCP is hard. One possible avenue towards showing the hardness of P-LCP might be to reduce from Piecewise-Linear Contraction. Our UniqueEOPL containment proof for Piecewise-Linear Contraction makes explicit use of the fact that the problem can be formulated as an LCP, although in that case the resulting matrix is not a P-matrix. Can we modify the reduction to produce a P-matrix?

Conjecture 4. UniqueEOPL $\subset$ EOPL = CLS.

The question of EOPL vs CLS is unresolved, and we actually think it could go either way. One could show that EOPL = CLS by placing either of the two known CLS-complete Contraction variants into EOPL [14, 19]. If the two classes are actually distinct, then it is interesting to ask which of the problems in CLS are also in EOPL.

On the other hand, we believe that UniqueEOPL is a strict subset of EOPL. The evidence for this is that the extra violation in UniqueEOPL that does not appear in EndOfPotentialLine changes the problem significantly. This new violation will introduce many new solutions whenever there are multiple lines in the instance, and so it is unlikely, in our view, that one could reduce EndOfPotentialLine to UniqueEOPL. Of course, there is no hope to unconditionally prove that UniqueEOPL $\subset$ EOPL, but we can ask for further evidence to support the idea. For example, can oracle separations shed any light on the issue?

Finally, we remark that UniqueEOPL is the closest complexity class to FP, among all the standard sub-classes of TFNP. However, we still think that further subdivisions of UniqueEOPL will be needed. Specifically, we do not believe that simple stochastic games, or any of the problems that can be reduced to them, are UniqueEOPL-complete, since all of these problems have unique solutions unconditionally. Further research will be needed to classify these problems.

Acknowledgements

We thank Aviad Rubinstein, Kousha Etessami, and Rasmus Ibsen-Jensen for helpful comments on a preprint of this paper.

References

[1] Adler, I., and Verma, S. The linear complementarity problem, Lemke algorithm, perturbation, and the complexity class PPAD. Tech. rep., Manuscript, Depatrment of IEOR, University of California, Berkeley, CA 94720, 2011.

[2] Banach, S. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta Mathematicae 3*, 1 (1922), 133–181.

[3] Bitansky, N., Paneth, O., and Rosen, A. On the cryptographic hardness of finding a Nash equilibrium. In *Proc. of FOCS* (2015), pp. 1480–1498.

[4] Boonyasiriwat, C., Sikorski, K., and Xiong, C. A note on two fixed point problems. *J. Complexity 23*, 4-6 (2007), 952–961.

[5] Chen, X., and Deng, X. On algorithms for discrete and approximate Brouwer fixed points. In *Proc. of STOC* (2005), pp. 323–330.

[6] Chen, X., and Deng, X. Matching algorithmic bounds for finding a Brouwer fixed point. *J. ACM 55*, 3 (2008), 13:1–13:26.

[7] Chen, X., and Deng, X. On the complexity of 2d discrete fixed point problem. *Theor. Comput. Sci. 410*, 44 (2009), 4448–4456.

[8] Chen, X., Deng, X., and Teng, S.-H. Settling the complexity of computing two-player Nash equilibria. *J. ACM 56*, 3 (2009), 14.

[9] Chung, S.-J. NP-completeness of the Linear Complementarity Problem. *Journal of Optimization Theory and Applications 60*, 3 (1989), 393–399.

[10] Cottle, R. W., Pang, J.-S., and Stone, R. E. *The Linear Complementarity Problem.* SIAM, 2009.

[11] Coxson, G. E. The P-matrix problem is co-NP-complete. *Mathematical Programming 64*, 1 (1994), 173–178.

[12] Daskalakis, C., Goldberg, P. W., and Papadimitriou, C. H. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing 39*, 1 (2009), 195–259.

[13] Daskalakis, C., and Papadimitriou, C. Continuous Local Search. In *Proc. of SODA* (2011), pp. 790–804.

[14] Daskalakis, C., Tzamos, C., and Zampetakis, M. A Converse to Banach's Fixed Point Theorem and its CLS Completeness. In *Proc. of STOC* (2018).

[15] Deng, X., Qi, Q., Saberi, A., and Zhang, J. Discrete fixed points: Models, complexities, and applications. *Math. Oper. Res. 36*, 4 (2011), 636–652.

[16] Dohrau, J., Gärtner, B., Kohler, M., Matoušek, J., and Welzl, E. ARRIVAL: a zero-player graph game in $\mathsf{NP} \cap \mathsf{coNP}$. In *A journey through discrete mathematics*. Springer, Cham, 2017, pp. 367–374.

[17] Etessami, K., and Yannakakis, M. On the complexity of Nash equilibria and other fixed points. *SIAM J. Comput. 39*, 6 (2010), 2531–2597.

[18] Fabrikant, A., Papadimitriou, C., and Talwar, K. The complexity of pure Nash equilibria. In *Proc. of STOC* (2004), ACM, pp. 604–612.

[19] Fearnley, J., Gordon, S., Mehta, R., and Savani, R. CLS: New problems and completeness. *CoRR abs/1702.06017* (2017).

[20] Fearnley, J., Gordon, S., Mehta, R., and Savani, R. Unique End of Potential Line. In *Proc. of ICALP* (2019), pp. 56:1–56:15.

[21] Fearnley, J., and Savani, R. The complexity of all-switches strategy improvement. In *Proc. of SODA* (2016), pp. 130–139.

[22] Friedmann, O., Hansen, T. D., and Zwick, U. A subexponential lower bound for the random facet algorithm for parity games. In *Proc. of SODA* (2011), pp. 202–216.

[23] Gairing, M., and Savani, R. Computing stable outcomes in symmetric additively separable hedonic games. *Math. Oper. Res. 44*, 3 (2019), 1101–1121.

[24] Garg, S., Pandey, O., and Srinivasan, A. Revisiting the cryptographic hardness of finding a Nash equilibrium. In *Annual Cryptology Conference* (2016), Springer, pp. 579–604.

[25] Gärtner, B. The random-facet simplex algorithm on combinatorial cubes. *Random Struct. Algorithms 20*, 3 (2002), 353–381.

[26] Gärtner, B., Hansen, T. D., Hubáček, P., Král, K., Mosaad, H., and Slívová, V. ARRIVAL: next stop in CLS. In *Proc. of ICALP* (2018), pp. 60:1–60:13.

[27] Gärtner, B., and Schurr, I. Linear programming and unique sink orientations. In *Proc. of SODA* (2006), pp. 749–757.

[28] Gärtner, B., and Thomas, A. The complexity of recognizing unique sink orientations. In *Proc. of STACS* (2015), pp. 341–353.

[29] Hansen, T. D., and Ibsen-Jensen, R. The complexity of interior point methods for solving discounted turn-based stochastic games. In *Conference on Computability in Europe* (2013), pp. 252–262.

[30] Hansen, T. D., Paterson, M., and Zwick, U. Improved upper bounds for random-edge and random-jump on abstract cubes. In *Proc. of SODA* (2014), pp. 874–881.

[31] Hirsch, M. D., Papadimitriou, C. H., and Vavasis, S. A. Exponential lower bounds for finding Brouwer fix points. *J. Complexity 5*, 4 (1989), 379–416.

[32] Hoke, K. W. Completely unimodal numberings of a simple polytope. *Discrete Applied Mathematics 20*, 1 (1988), 69–81.

[33] Huang, Z., Khachiyan, L. G., and Sikorski, C. K. Approximating fixed points of weakly contracting mappings. *J. Complexity 15*, 2 (1999), 200–213.

[34] Hubáček, P., and Yogev, E. Hardness of continuous local search: Query complexity and cryptographic lower bounds. In *Proc. of SODA* (2017), pp. 1352–1371.

[35] Jeřábek, E. Integer factoring and modular square roots. *Journal of Computer and System Sciences 82*, 2 (2016), 380–394.

[36] Johnson, D. S., Papadimitriou, C. H., and Yannakakis, M. How easy is local search? *Journal of Computer and System Sciences 37*, 1 (1988), 79–100.

[37] Kalai, G. Three puzzles on mathematics, computation, and games. *CoRR abs/1801.02602* (2018).

[38] Kojima, M., Megiddo, N., Noma, T., and Yoshise, A. *A unified approach to interior point algorithms for linear complementarity problems*, vol. 538. Springer Science & Business Media, 1991.

[39] Kojima, M., Megiddo, N., and Ye, Y. An interior point potential reduction algorithm for the linear complementarity problem. *Mathematical Programming 54*, 1-3 (1992), 267–279.

[40] Lemke, C. E. Bimatrix equilibrium points and mathematical programming. *Management science 11*, 7 (1965), 681–689.

[41] Matoušek, J., and Szabó, T. Random edge can be exponential on abstract cubes. In *Proc. of FOCS* (2004), pp. 92–100.

[42] Megiddo, N. *A note on the complexity of P-matrix LCP and computing an equilibrium.* IBM Thomas J. Watson Research Division, 1988.

[43] Megiddo, N., and Papadimitriou, C. H. On total functions, existence theorems and computational complexity. *Theoretical Computer Science 81*, 2 (1991), 317–324.

[44] Mehta, R. Constant rank bimatrix games are PPAD-hard. In *Proc. of STOC* (2014), pp. 545–554.

[45] Meunier, F., Mulzer, W., Sarrabezolles, P., and Stein, Y. The rainbow at the end of the line: A PPAD formulation of the colorful Carathéodory theorem with applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2017), pp. 1342–1351.

[46] Morris Jr, W. D. Randomized pivot algorithms for P-matrix linear complementarity problems. *Mathematical programming 92*, 2 (2002), 285–296.

[47] Murty, K. G. Computational complexity of complementary pivot methods. In *Complementarity and fixed point problems.* Springer, 1978, pp. 61–73.

[48] Nemirovsky, A., and Yudin, D. B. *Problem Complexity and Method Efficiency in Optimization.* Wiley, New York, 1983.

[49] Papadimitriou, C. H. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences 48*, 3 (1994), 498–532.

[50] Puri, A. Theory of hybrid systems and discrete event systems.

[51] Rubinstein, A. Settling the complexity of computing approximate two-player Nash equilibria. In *Proc. of FOCS* (2016), pp. 258–265.

[52] Schäffer, A. A., and Yannakakis, M. Simple local search problems that are hard to solve. *SIAM journal on Computing 20*, 1 (1991), 56–87.

[53] Schurr, I., and Szabó, T. Finding the sink takes some time: An almost quadratic lower bound for finding the sink of unique sink oriented cubes. *Discrete & Computational Geometry 31*, 4 (2004), 627–642.

[54] Schurr, I., and Szabó, T. Jumping doesn't help in abstract cubes. In *Proc. of IPCO* (2005), pp. 225–235.

[55] Shellman, S., and Sikorski, K. A recursive algorithm for the infinity-norm fixed point problem. *Journal of Complexity 19*, 6 (2003), 799 – 834.

[56] Sikorski, K. *Optimal solution of Nonlinear Equations.* Oxford Press, New York, 200.

[57] Sikorski, K. Computational complexity of fixed points. *Journal of Fixed Point Theory and Applications 6*, 2 (2009), 249–283.

[58] Stickney, A., and Watson, L. Digraph models of bard-type algorithms for the linear complementarity problem. *Mathematics of Operations Research 3*, 4 (1978), 322–333.

[59] Szabó, T., and Welzl, E. Unique sink orientations of cubes. In *Proc. of FOCS* (2001), pp. 547–555.

[60] Thomas, A. Exponential lower bounds for history-based simplex pivot rules on abstract cubes. In *Proc. of ESA* (2017), pp. 69:1–69:14.

[61] Zwick, U., and Paterson, M. The complexity of mean payoff games on graphs. *Theoretical Computer Science 158*, 1-2 (1996), 343–359.

A Proofs for Section 2

A.1 Proof of Theorem 4

First we recall the definition of EndOfMeteredLine, which was first defined in [34].

Definition 34 (EndOfMeteredLine [34])**.** *Given circuits $S, P : \{0,1\}^n \to \{0,1\}^n$, and $V : \{0,1\}^n \to \{0,\dots,2^n\}$ such that $P(0^n) = 0^n \neq S(0^n)$ and $V(0^n) = 1$, find a string $\boldsymbol{x} \in \{0,1\}^n$ satisfying one of the following*

(T1) either $S(P(\boldsymbol{x})) \neq \boldsymbol{x} \neq 0^n$ or $P(S(\boldsymbol{x})) \neq \boldsymbol{x}$,

(T2) $\boldsymbol{x} \neq 0^n, V(\boldsymbol{x}) = 1$,

(T3) either $V(\boldsymbol{x}) > 0$ and $V(S(\boldsymbol{x})) - V(\boldsymbol{x}) \neq 1$, or $V(\boldsymbol{x}) > 1$ and $V(\boldsymbol{x}) - V(P(\boldsymbol{x})) \neq 1$.

EndOfMeteredLine is actually quite similar to EndOfPotentialLine. The main difference is that in EndOfMeteredLine, each edge must increase the potential by exactly 1, which is enforced by solution type T3. In the following lemma we show that every EndOfPotentialLine instance can be modified so that this is the case.

Lemma 35. *Every* EndOfPotentialLine *instance $L = (S, P, V)$ can be reduced in polynomial time to another* EndOfPotentialLine *instance $L' = (S', P', V')$ so that for every vertex x and y with $y = S'(x)$ and $x = P'(y)$ we have $V'(y) = V'(x) + 1$.*

Proof. Suppose that (x, y) is an edge in L, and that $V(y) > V(x) + 1$. The idea is to introduce a chain of dummy vertices between x and y of length $V(y) - V(x)$ that each increase the potential by exactly one. Specifically, each vertex in L' will consist of a pair (x, i), where $x \in \{0,1\}^n$ is a bit-string from the instance L, and i is a number between 0 and 2^n. The edge (x, y) is replaced by the following sequence:

$$(x, 0) \to (x, 1) \to \cdots \to (x, V(y) - V(x) - 1) \to (y, 0).$$

If we set $V'(x, i) = V(x) + i$, then observe that each edge in this sequence increases the potential by exactly one.

More formally, the EndOfPotentialLine instance L' is defined according to the following rules.

- If $x \in \{0,1\}^n$ is not a vertex in L, then for all i, (x, i) is not a vertex in L'.
- If $x \in \{0,1\}^n$ is a vertex in L but $S(x)$ is not, meaning that x is a sink, then $(x, 0)$ is a vertex and a sink in L' and each (x, i) with $i > 0$ will not be a vertex in L'.
- If $x, y \in \{0,1\}^n$ are both vertices in L, and $V(x) < V(y)$, then (x, i) is a vertex in L' if and only if $i \leq V(y) - V(x) - 1$. The successors and predecessors of these vertices follow the chain that we described above.
- If $x, y \in \{0,1\}^n$ are both vertices in L, but $V(x) \geq V(y)$, then $(x, 0)$ is a vertex with successor $(y, 0)$, and each (x, i) with $i \neq 0$ is not a vertex.
- For all vertices (x, i) in L', we have $V'(x, i) = V(x) + i$.

These rules can easily be encoded using polynomial sized circuits S', P', and V'.

To see that this is correct, observe that the solutions of the instance have not changed. A vertex x is a solution of type (R1) in L if and only if $(x, 0)$ is a solution of type (R1) in L'. A vertex x and $y = S(x)$ are a solution of type (R2) if and only if $(x, 0)$ and $(y, 0) = S'(x, 0)$ are a solution of type (R2) in L'. Vertices of the form (x, i) with $i > 0$ cannot be solutions of type (R1) because they always have a valid successor and predecessor, and they cannot be solutions of type (R2) since their outgoing edge always increases the potential by exactly one. □

We can now prove Theorem 4. Reducing EndOfMeteredLine to EndOfPotentialLine is straightforward: we can simply use the circuits S, P, with no modifications, and set $V'(x) = V(x) - 1$, to account for the fact that EndOfMeteredLine sets $V(0^n) = 1$, while EndOfPotentialLine sets $V(0^n) = 0$. Solutions of type (R1) map directly to solutions of type (T1), while solutions of type (R2) map directly to solutions of type (T3).

In the other direction, given an EndOfPotentialLine instance L, we first apply Lemma 35 to obtain $L' = (S', P', V')$, and create $V''(x) = V'(x) + 1$. We reinterpret (S', P', V'') as an EndOfMeteredLine instance. Solutions of type (T1) map directly to solutions of type (R1), and solutions of type (T3) map directly to solutions of type (R2) because we applied Lemma 35.

Solutions of type (T2) have no direct analogue, but if we are given a solution x of type (T2), meaning that $x \neq 0^n$ and $V''(x) = 1$, then we are able to find a solution of one of the other two types. Specifically, if $P'(x)$ is not a vertex, then we have found a solution of type (R1). If it is a vertex, then, since $V(x) = V''(x) - 1 = 0$, the edge $(P'(x), x)$ must have non-increasing potential, giving us a solution of type (R2).

B Proofs for Section 3

B.1 Proof of Lemma 11

Throughout this proof, we will fix $\mathcal{D} = (D_i)_{i=1,\dots,d}$ to be the direction functions, and $P = P(k_1, k_2, \dots, k_d)$ to be the set of points used in the OPDC instance. We will produce a UniqueForwardEOPL instance $L = (S, V)$.

The circuit S. A vertex of the line is a tuple $(p_0, p_1, p_2, \dots, p_d)$, where each $p_i \in P \cup \{-\}$ is either a point or a special symbol, $-$, that is used to indicate an unused element. We use $\mathsf{vert} = (P \cup \{-\})^{d+1}$ to denote the set of possible vertices. Only some of the tuples are valid encodings for a vertex. To be valid, a vertex $(p_0, p_1, p_2, \dots, p_d)$ must obey the following rules:

1. If $p_i \neq -$, then $D_j(p_i) = \mathsf{zero}$ for all $j \leq i$. This means that if p_i is a point, then it must be a point on the i-surface.
2. If $p_i \neq -$, then we must have $D_{i+1}(p_i) \neq \mathsf{down}$.
3. If $p_i \neq -$ and $p_j = -$ and $i < j$, then we must have $(p_i)_{j+1} = 0$.
4. If $p_i \neq -$ and $p_j \neq -$ and $i < j$, then we must have $(p_i)_{j+1} = (p_j)_{j+1} + 1$.

We define the function $\text{IsVertex} : \mathsf{vert} \to \{\texttt{true}, \texttt{false}\}$ that determines whether a given $v \in \mathsf{vert}$ is a valid encoding of a vertex, by following the rules laid out above. This can clearly be computed in polynomial time.

The initial vertex, which will be mapped to the bit-string 0^n, will be $(p_{\text{init}}, -, \dots, -)$, where $p_{\text{init}} = (0, 0, \dots, 0)$ is the all zeros point in P. We also select some bit-string $\texttt{end} \in \{0,1\}^n$, which

will satisfy $S(\texttt{end}) = \texttt{end}$ and $V(\texttt{end}) = 0$. Whenever we need to make a vertex v a sink, we will set $S(v) = \texttt{end}$, which will cause v to become a solution of type (UF1). Note that since $V(\texttt{end}) = 0$, this cannot not create a solution of type (UFV2).

Given a vertex encoding $v = (p_0, p_1, p_2, \ldots, p_d) \in \mathsf{vert}$, the circuit S carries out the following operations. If IsVertex(v) is `false`, then $S(v) = v$, indicating that v is indeed not a vertex. Otherwise, we use the following set of rules to determine the successor of v. Let i be the smallest index such that $p_i \neq -$.

1. If $i = d$ then our vertex has the form $v = (-, \ldots, -, p_d)$, and p_d is on the d-surface, meaning that it is a solution to the discrete contraction map. So we set $S(v) = \texttt{end}$ to ensure that this is a solution.

2. If $D_{i+1}(p_i) = \mathsf{zero}$, then we define $S(v) = v'$ where

$$v'_j = \begin{cases} - & \text{if } j < i+1, \\ p_i & \text{if } j = i+1, \\ p_j & \text{if } j > i+1. \end{cases}$$

   This operation overwrites the point in position $i+1$ with p_i, and sets position i to $-$. All other components of v are unchanged.

3. If $D_{i+1}(p_i) \neq \mathsf{zero}$ and $i > 0$ then let q be the point such that

$$(q)_j = \begin{cases} 0 & \text{if } j < i+1, \\ (p_i)_{i+1} + 1 & \text{if } j = i+1, \\ (p_i)_j & \text{if } j > i+1. \end{cases}$$

   (a) If q is a point in P, then we define $S(v) = (q, p_1, p_2, \ldots, p_d)$.

   (b) Otherwise, we must have that $(p_i)_{i+1} = k_{i+1}$, meaning that p_i is the last point of the grid. This means that we have a solution of type (OV3), since the fact that IsVertex$(v) = \texttt{true}$ implies that $D_{i+1}(p_i) = \mathsf{up}$. So we set $S(v) = \texttt{end}$.

4. If $D_{i+1}(p_i) \neq \mathsf{zero}$ and $i = 0$ then let q be the point such that $(q)_j = (p_0)_j$ for all $j > 1$, and $(q)_1 = (p_0)_1 + 1$.

   (a) If q is in the point set P, then we define $S(v) = (q, p_1, p_2, \ldots, p_d)$.

   (b) If q is not in P, then we again have a solution of type (OV3), since $(p_0)_1 = k_1$, and $D_1(p_0) = \mathsf{up}$ from the fact that IsVertex$(v) = \texttt{true}$. So we set $S(v) = \texttt{end}$.

Note that in the case where IsVertex$(x) = \texttt{true}$, the algorithm never sets $S(x) = x$, and so we have that $S(x) \neq x$ if and only if IsVertex$(x) = \texttt{true}$, which is a fact that we will use in our proofs.

The potential function. To define the potential function, we first define an intuitive potential that uses a tuple of values ordered lexicographically, and then translate this into a circuit V that produces integers. We call the tuple of values the lexicographic potential associated with a vertex v, and denote it using LexPot(v). To define the lexicographic potential, we need to introduce an auxiliary function Potential : $(P \cup \{-\}) \times \{0, \ldots, d+1\} \to \mathbb{Z}$ given by

$$\text{Potential}(p, i) = \begin{cases} (p)_{i+1} + 1 & \text{if } p \neq -, \\ 0 & \text{otherwise.} \end{cases}$$

The lexicographic potential of $v = (p_0, p_1, \ldots, p_d) \in \mathsf{vert}$ is the following:

$$\text{LexPot}(v) = (\text{Potential}(p_0, 0), \text{Potential}(p_1, 1), \ldots, \text{Potential}(p_{d-1}, d-1)) .$$

Note that the $\text{LexPot}(v) \in \mathbb{Z}^d$, since the definition ignores p_d.

Let $\prec_d$ be the ordering on tuples from $\mathbb{Z}^d$ where they are compared lexicographically from right to left, so that $(0,0) \prec_2 (1,0) \prec_2 (0,1) \prec_2 (1,1)$, for example. Our potential function will be defined by the tuples given by LexPot and the order $\prec_d$. We omit the subscript from $\prec$ whenever it is clear from the context.

Given a vertex $v = (p_0, p_1, \ldots, p_d) \in \mathsf{vert}$, let $\text{LexPot}(v) = (l_0, \ldots, l_{d-1})$. To translate from lexicographically ordered tuples to integers in a way that preserves the ordering, we pick some integer k such that $k > k_i$ for all i, meaning that k is larger than the grid-width used in every dimension, which implies that $l_j < k$ for all j. We now take a weighted sum of the coordinates of $\text{LexPot}(v)$ where the weight for coordinate i is k^i, so that the ith coordinate dominates coordinates 0 through $i-1$. The final potential value $V : \mathsf{vert} \to \mathbb{Z}$ is then given by

$$V(v) = \sum_{i=0}^{d-1} k^i l_i,$$

which can be easily computed in polynomial time. This completes the definition of the UniqueForwardEOPL problem (S, V).

Lemma 36. *Every solution of the* UniqueForwardEOPL *instance* (S, V) *can be used to find a solution of the* OPDC *instance given by* $\mathcal{D}$ *and* P*. Furthermore, solutions of type (O1) are only ever mapped onto solutions of type (UF1).*

Proof. We begin by considering solutions of type (UF1). Let $x = (p_0, p_1, \ldots, p_d)$ and let $y = S(x)$, and suppose that x is a (UF1) solution. This means that $S(x) \neq x$ and either $S(y) = y$ or $V(y) \leq V(x)$. We first suppose that $S(y) = y$, and note that in this case we must have $\text{IsVertex}(x) = \texttt{true}$ while $\text{IsVertex}(y) = \texttt{false}$. We have the following cases based on the rule used to determine $S(x)$.

1. If $S(x)$ is determined by the first rule in the definition of S, then this means that $p_d \neq -$. Since $\text{IsVertex}(x) = \texttt{true}$, this means that $D_i(p_d) = \mathsf{zero}$ for all i, which means that p_d is a solution of type (O1).

2. If $S(x)$ is determined by the second rule in the definition of S, then there are two cases. Let i be the smallest index such that $p_i \neq -$.

   (a) If $D_{i+2}(p_i) = \mathsf{down}$ then we have a solution of type (OV2) or (OV3).

      i. If $p_{i+2} \neq -$ then p_i and p_{i+2} give us a solution of type (OV2). Specifically, this holds because $D_{i+2}(p_{i+2}) = \mathsf{up}$ while $D_{i+2}(p_i) = \mathsf{down}$, and $(p_i)_{i+2} = (p_{i+2})_{i+2} + 1$ holds because $\text{IsVertex}(x) = \texttt{true}$. Moreover, $D_j(p_i) = D_j(p_{i+2}) = \mathsf{zero}$, for all $j < i+2$, where in particular the fact that $D_{i+1}(p_i) = \mathsf{zero}$ is given by the fact that we are in the second case of the definition of S.

      ii. If $p_{i+2} = -$ then this means that $(p_i)_{i+2} = 0$. Since $D_{i+2}(p_i) = \mathsf{down}$ this gives us a solution of type (OV3).

(b) If $D_{i+2}(p_i) \neq \mathsf{down}$, then we argue that this case is impossible. Specifically, we will show that IsVertex$(y) = \texttt{true}$, meaning that $S(y) \neq y$. To do this, we will prove that the four conditions of IsVertex hold for y. Note that y differs from x only in positions i and $i+1$, and that position i of y is $-$. So we only need to consider the conditions imposed by IsVertex when the point p_i is placed in position $i+1$.

i. The first condition of IsVertex is that p_i should be on the $(i+1)$-surface, which is true because the second rule of S explicitly checks that $D_{i+1}(p_i) = \mathsf{zero}$, while the fact that IsVertex$(x) = \texttt{true}$ guarantees that $D_j(p_i) = \mathsf{zero}$ for all $j < i+1$.

ii. The second condition requires that $D_{i+2}(p_i) \neq \mathsf{down}$, which is true by assumption.

iii. Every constraint imposed by the third and fourth conditions also holds for p_i in x, and so the fact that IsVertex$(x) = \texttt{true}$ implies that these conditions hold for y.

3. If $S(x)$ is determined by the third rule defining S, then we have two cases. Since the third rule was used, we know that $y = (q, p_1, p_2, \ldots, p_d)$, with the definition of q being given in the third rule.

(a) If q is not in P, then we have a solution of type (OV3), as described in the algorithm for S.

(b) If $q \in P$ and $D_1(q) = \mathsf{down}$, then we have a solution of type (OV3), since $q_1 = 0$ by definition.

(c) If $q \in P$ and $D_1(q) \neq \mathsf{down}$ and $q \in P$, then we argue that the case is impossible, and we prove this by showing that IsVertex$(y) = \texttt{true}$. Note that y differs from x only in the position occupied by q, and so this is the only point for which we need to prove the conditions, since all the other points satisfy the conditions by the fact that IsVertex$(x) = \texttt{true}$.

i. The first requirement of IsVertex(y) holds trivially, since the only new requirement is that q is on the 0-surface, and every point is on the 0-surface by definition.

ii. The second requirement is that $D_1(q) \neq \mathsf{down}$, which is true by assumption.

iii. The third and fourth conditions place constraints on certain coordinates of q. For coordinates $j < i$, the third condition requires that $q_j = 0$, which is true by definition, while the fourth condition is inapplicable. For coordinates $j \geq i$, the constraints imposed by the third and fourth conditions hold because $q_j = (p_i)_j$ in these coordinates, and p_i also satisfies these constraints.

4. If $S(x)$ is determined by the fourth rule, then we have two cases. Let $y = (q, p_1, p_2, \ldots, p_d)$ be the value of y produced by the fourth rule, where the definition of q is given in that rule.

(a) If q is not in P, then we have a solution of type (OV3), as described in the algorithm for S.

(b) If $q \in P$ and $D_1(q) = \mathsf{down}$ then we have a solution of type (OV2). Specifically, the points p_0 and q provide the violation since $(q)_1 = (p_0)_1 + 1$, while $D_1(p_0) = \mathsf{up}$ and $D_1(q) = \mathsf{down}$. The fact that both q and p_0 belong to the same 1-slice is guaranteed by the definition of q.

(c) If $q \in P$ and $D_1(q) \neq \mathsf{down}$ then we again argue that IsVertex$(y) = \texttt{true}$, making this case impossible. The reasoning is the same as the reasoning used in case 3b.

We now proceed to the case where we have a solution $x = (p_0, p_1, \ldots, p_d)$ of type (UF1) and the vertex $y = S(x)$ satisfies $S(y) \neq y$. In this case, we must have $V(y) \leq V(x)$. We argue that this is impossible, and again we will do a case analysis based on the rule used to determine the output of the circuit S.

1. If $S(x)$ is determined by the first rule then we have $\text{IsVertex}(y) = \texttt{false}$, which is not possible in this case.

2. If $S(x)$ is determined by the second rule, then we can prove that $V(y) > V(x)$. This is because y differs from x only in positions i and $i+1$, and because $(p_i)_{i+1} = (p_{i+1})_{i+1} + 1$ by the fourth rule of $\text{IsVertex}(x)$. Hence

   $$k^i \cdot \text{Potential}(-, i) + k^{i+1} \cdot \text{Potential}(p_i, i+1) > k^i \cdot \text{Potential}(p_i, i) + k^{i+1} \cdot \text{Potential}(p_{i+1}, i+1),$$

   where we are using the fact that $k^{i+1} > k^i \cdot \text{Potential}(p_i, i)$. Thus, $V(y) > V(x)$.

3. If $S(x)$ is determined by the third rule, then note that we must have $q \in P$. We again argue that $V(y) > V(x)$. Specifically, observe that $V(y) = V(x) + \text{Potential}(q, 0) = V(x) + 1$.

4. If $S(x)$ is determined by the fourth rule, then note that we must have $q \in P$, and we also have $V(y) > V(x)$. Specifically

   $$\begin{aligned} V(y) &= V(x) - \text{Potential}(p_0, 0) + \text{Potential}(q, 0) \\ &= V(x) - (p_0)_1 + q_1 \\ &= V(x) + 1. \end{aligned}$$

We now move to the case where we have a solution of type (UFV1). In this case we have two vertices $x = (p_0, p_1, \ldots, p_d)$ and $y = (q_0, q_1, \ldots, q_d)$ for which $\text{IsVertex}(x) = \text{IsVertex}(y) = \texttt{true}$, and for which $V(x) \leq V(y) < V(S(x))$. We will once again perform a case analysis over the possible cases of S.

1. $S(x)$ cannot be determined by the first case in the definition of S, because that case only applies when $\text{IsVertex}(y) = \texttt{false}$.

2. If $S(x)$ is determined by the second case in the definition of S, then we observe that we have $\text{LexPot}(x) = (0, \ldots, 0, v_i, v_{i+1}, \ldots, v_d)$ and $\text{LexPot}(S(x)) = (0, \ldots, 0, 0, v_{i+1} + 1, v_{i+2} \ldots, v_d)$, i.e., the ith element of $\text{LexPot}(x)$ is replaced by 0, and the $(i+1)$th element is replaced by $v_{i+1} + 1$. Note also that elements $i+2$ through d of $\text{LexPot}(S(x))$ agree with the corresponding elements of $\text{LexPot}(x)$.

   Since we have $V(x) \leq V(y) < V(S(x))$ this means that $\text{LexPot}(x) \prec \text{LexPot}(y) \prec \text{LexPot}(S(x))$. Hence, $\text{LexPot}(y)$ must have the form $(v'_0, v'_1, \ldots, v'_i, v_{i+1}, v_{i+2}, \ldots, v_d)$, meaning that it agrees with $\text{LexPot}(x)$ and $\text{LexPot}(S(x))$ on elements $i+1$ through d. Furthermore, we must have $v'_i \geq v_i$.

   Let j be the largest index satisfying $j \leq i$ and $v'_j \neq v_j$. Note that such a j must exist, since otherwise we would have $x = y$, which would contradict the fact that $x \neq y$ in any solution of type UFV1. We claim that p_j and q_j form a solution of type (OV1).

   Let s be the j-slice that satisfies $s_l = *$ for all $l \leq j$ and $s_l = (p_j)_l$ for all $l > j$. The point p_j lies in the slice s by definition. We claim that q_j also lies in this slice, which follows from the

fact that $v_l = v'_l$ for all $l > j$, combined with the third and fourth properties of IsVertex(x) and IsVertex(y). Note also that $(p_j)_j \neq (q_j)_j$, and hence $p_j \neq q_j$.

Finally, the first property of IsVertex(x) and IsVertex(y) imply that $D_l(p_j) = \mathsf{zero}$ and $D_l(q_j) = \mathsf{zero}$ for all $l \leq j$. So p_j and q_j are two distinct fixpoints of the j-slice s, meaning that we have a solution of type (OV1).

3. We claim that $S(x)$ cannot be determined by the third rule in the definition of S. In this case we have LexPot$(x) = (0, v_1, v_2, \ldots, v_d)$, since the case is only applicable when $p_0 = -$. Observe that LexPot$(S(x)) = (1, v_1, v_2, \ldots, v_d)$, and that there is no possible tuple t that satisfies LexPot$(x) \prec t \prec$ LexPot$(S(x))$. This means that we must have $V(y) = V(x)$, but this is only possible if $y = x$, due to the constraints placed by the third and fourth properties of IsVertex. Hence this case is not possible, since $y = x$ is specifically ruled out in a solution of type UVF1.

4. For similar reasons, we claim that $S(x)$ cannot be determined by the fourth rule. In this case we have LexPot$(x) = (v_0, v_1, v_2, \ldots, v_d)$, and LexPot$(S(x)) = (v_0 + 1, v_1, v_2, \ldots, v_d)$, which again means that there cannot be a tuple t satisfying LexPot$(x) \prec t \prec$ LexPot$(S(x))$. Hence we would have $V(x) = V(y)$ and $x = y$ as in the previous case, which is impossible.

Finally, violations of type (UFV2) can be easily dealt with, because each violation of type (UFV2) contains a solution of type (UF1), and we have already argued that solutions of type (UF1) can be mapped back to solutions or violations of the original OPDC problem.

To complete the proof, we must argue that the reduction is promise-preserving. Specifically, we must show that if the original OPDC instance has no violations, then the resulting UFEOPL instance also has no violations. We have already shown that any violation of type (UFV1) can be mapped back to a violation of the OPDC instance, and therefore if the OPDC instance has no violations, then the UFEOPL instance cannot have violations of type (UFV1).

For violations of type (UFV2), let x and y be the two vertices given in the violation, and recall that x is a solution of type (UF1). We have already shown that if $S(x)$ is determined by anything other than the first rule of the algorithm for S, then we have a violation in the OPDC instance.

So, we must show that if $S(x)$ is determined by the first rule, then there must exist a violation somewhere in the OPDC instance. To find this violation, one can follow the line starting at y to find a second distinct solution x' of the UFEOPL instance. We are immediatly done if this solution is of type (UFV1), or if it is of type (UF1) and $S(x')$ is determined by anything other than the first rule of the algorithm for S, since these yield violations of the OPDC instance. Otherwise, x and x' map back to two distinct solutions of type (O1) of the OPDC instance. But these solutions are also a violation of type (OV1).

□

This completes the proof of Lemma 11.

B.2 Proof of Lemma 15

The reduction of Hubáček and Yogev [34] relies on the *pebbling game* technique that was first applied by Bitansky et al [3]. Let $L = (S, W, x_s, T)$ be an SinkOfVerifiableLine instance. The pebbling game is played by placing *pebbles* on the vertices of this instance according to the following rules.

- A pebble may be placed or removed from the starting vertex x_s at any time.

- A pebble may be placed or removed on a vertex $x \neq x_s$ if and only if there is a pebble on a vertex y with $S(y) = x$.

Given n pebbles, how far can we move along the line by following these rules? The answer is that we can place a pebble on vertex $2^n - 1$ by applying the following *optimal strategy*. The strategy is recursive. In the base case, we can place a pebble on vertex $2^1 - 1 = 1$ by placing our single pebble on the successor of x_s. In the recursive step, where we have n pebbles, we use the following approach:

1. Follow the optimal strategy for $n - 1$ pebbles, in order to place a pebble on vertex $2^{n-1} - 1$.
2. Place pebble n on the vertex 2^{n-1}.
3. Follow the optimal strategy for $n - 1$ pebbles *backwards* in order to reclaim the first $n - 1$ pebbles.
4. Follow the optimal strategy for $n - 1$ pebbles forwards, but starting from the vertex 2^{n-1}. This ends by placing a pebble on vertex $2^{n-1} + 2^{n-1} - 1 = 2^n - 1$.

Step 3 above relies on the fact that the pebbling game is *reversible*, meaning that we can execute any sequence of moves backwards as well as forwards. At the beginning of Step 4, we have a single pebble on vertex 2^{n-1}, and we follow the optimal strategy again, but using 2^{n-1} as the starting point.

The reduction from UniqueForwardEOPL to UniqueEOPL. To reduce UniqueForwardEOPL to UniqueEOPL, we play the optimal strategy for the pebbling game. Note that, since every step of the pebbling game is reversible, this gives us a predecessor circuit. We will closely follow the reduction given by Bitansky et al [3] from SinkOfVerifiableLine to EndOfLine. Specifically, we will reduce an instance $L = (S, V)$ of UniqueForwardEOPL to an instance $L' = (S', P', V')$ of UniqueEOPL.

A vertex in L' will be a tuple of pairs $((v_1, a_1), (v_2, a_2), \ldots, (v_n, a_n))$ describing the state of the pebbling game. Each v_i is a bit-string, while each a_i is either

- the special symbol $-$, implying that pebble i is not used and that the bit-string v_i should be disregarded, or
- an integer such that $a_i = V(v_i)$, meaning that pebble i is placed on the vertex v_i.

Bitansky et al [3] have produced circuits S' and P' that implement the optimal strategy of the pebbling game for pebbles encoded in this way. The only slight difference is that they reduce from SinkOfVerifiableLine, but we can apply their construction by creating the circuit W so that $W(v, a) = 1$ if and only if $V(v) = a$. We refer the reader to their work for the full definition of these two circuits.

Hubáček and Yogev [34] built upon this reduction by showing that it is possible to give a potential function V' for the resulting instance. Specifically, their potential encodes how much progress we have made along the optimal strategy, which it turns out, can be computed purely from the current configuration of the pebbles. Their construction also guarantees that the potential at each vertex always increases by 1, meaning that we have $V(S(x)) = V(x) + 1$ whenever $S(x)$ and x are both vertices. We refer the reader to their work for the full definition of the circuit V'.

Violations. So far, we have a reduction from the promise version of UniqueForwardEOPL to UniqueEOPL, which entirely utilizes prior work. Specifically, every solution of type (U1) in L' will map back to a solution of type (UFP1) in L.

Our contribution is to handle the violations, thereby giving a promise-preserving reduction from the non-promise version of UniqueForwardEOPL to the non-promise version of UniqueEOPL. We begin by showing, in the following lemma, that the reduction is correct, and afterwards we will argue that the reduction is promise-preserving.

Lemma 37. *Every violation in L' can be mapped back to a solution of L.*

Proof. There are three types of violation in L'.

1. Violations of type (UV1), which are edges where the potential decreases, are not possible, since the reduction of Hubáček and Yogev ensures that $V'(S'(x)) = V'(x) + 1$ whenever x and $S'(x)$ are both vertices.

2. In violations of type (UV2) we have a vertex $((v_1, a_1), (v_2, a_2), \ldots, (v_n, a_n))$ that is the start of a second line. This means that, for some reason, we are not able to execute the optimal strategy backwards from this vertex. There are two possibilities

   (a) The optimal strategy needs to place a pebble on the successor of some vertex v_i, but it cannot because v_i is the end of a line. This means that either $S(v_i) = v_i$ or that $V(S(v_i)) \neq V(v_i) + 1$, and in either case we have a solution of type (UFP1) for L.

   (b) The optimal strategy needs to remove the pebble v_i, but it cannot, because it does not have a pebble on a vertex u with $S(u) = u$. By construction, there will be some pebble v_j with $a_j = a_i - 1$, but in this case we have $S(v_j) \neq v_i$. This means that we have two lines, and specifically we have that v_i and $S(v_j)$ are two vertices with the same potential, since $V(v_j) = a_j$ and $V(S(v_j)) = V(v_j) + 1$. This gives us a solution of type (UFPV1).

3. In violations of type (UV3) we have two distinct vertices $x = ((v_1, a_1), (v_2, a_2), \ldots, (v_n, a_n))$ and $y = ((u_1, b_1), (u_2, b_2), \ldots, (u_n, b_n))$ with $V'(x) \leq V'(y) < V'(S'(x))$. Since the reduction ensures that $V'(S'(x)) = V'(x) + 1$ this means that x and y have the same potential. The reduction of Hubáček and Yogev ensures that, if two vertices have the same potential, then they refer to the same step of the optimal strategy, meaning that $a_i = b_i$ for all i. This means that any pair of vertices v_i and u_i with $a_i \neq -$ is a pair of vertices with $V(v_i) = V(u_i)$, and so a solution of type (UFPV1). To see that such a pair must exist, it suffices to note that the only vertex with $a_i = -$ for all i is the start of the line, and there cannot be two distinct vertices with this property.

□

The lemma above proves that the reduction is correct, but it does not directly prove that it is promise-preserving. Specifically, in case 2a of the proof we show that some violations of (UV2) are mapped back to solutions of type (UFP1). This, however, is not a problem, because we can argue that case 2a of the proof can only occur if there is more than one line in L'.

Specifically, if we are at some vertex $x = ((v_1, a_1), (v_2, a_2), \ldots, (v_n, a_n))$ and $P(x)$ needs to place a pebble on $S(v_i)$, then v_i cannot be the furthest point in the pebble configuration, meaning that there is some v_j with $a_j \neq -$ and $a_j > a_i$. This can be verified by inspecting the recursive definition of the optimal strategy. But note that if v_i is the end of a line, and v_j is a vertex with $V(v_j) > V(v_i)$, then L' must contain more than one line.

Hence we can conclude that the reduction is promise-preserving. That is, assuming that there are no violations in L, we have shown that any violation in L' must map back to a violation in L, and so there cannot be any violations in L'. This completes the proof of Lemma 15.

C Proofs for Section 4

C.1 Proof of Lemma 18

The proof requires us to define two polynomial-time algorithms.

Splitting lines around a vertex. We begin by defining the subline function. We will use two sub-functions that split an instance in two based on a particular vertex. Let $L = (S, P, V)$ be a UniqueEOPL instance, and v be some vertex of L with $V(v) = 2^{n-1}$.

1. We define the function FirstHalf(v, L) to return a UniqueEOPL instance (S', P', V') by removing every vertex with potential greater than or equal to $V(v)$. Specifically, the S' and P' circuits will check whether $V(x) \geq 2^{n-1}$ for each vertex x, and if so, then they will set $S'(x) = P'(x) = x$, which ensures that x is not on the line. For any vertex x with $V(x) < 2^{n-1}$, we set $S'(x) = S(x)$, $P'(x) = P(x)$ and $V'(x) = V(x)$. Note that S', P', and V' can all be produced in polynomial time if we are given (S, P, V).

2. We define the function SecondHalf(v, L) to return a UniqueEOPL instance (S', P', V') by removing every vertex with potential strictly less than $V(v)$. For the circuits S' and P', this is done in the same way as the previous case, but this time the circuits will check whether $V(x) < 2^{n-1}$. The function V' is defined so that $V'(x) = V(x) - 2^{n-1}$, thereby ensuring that the potentials are in the range $[0, 2^{n-1})$. Finally, the string 0^n is remapped to represent the vertex v, which is the start of the second half of the line. In this case, we are able to compute S', P', and V' in polynomial time if we are given (S, P, V) and the bit-string v.

We remark that, although we view these functions as intuitively splitting a line, the definitions still work if L happens to contain multiple lines. Each line in the instance will be split based on the potential of v.

The subline function. The subline function is defined recursively, based on the number of bit-strings that are given to the function. In the base case we are given a line L and zero bit-strings, in which case subline$(L) = L$.

For the recursive step, assume that we are given bit-strings v_i through v_n. Let $L' = (S', P', V') =$ subline$(v_{i+1}, v_{i+2}, \ldots, v_n, L)$.

- If $V'(v_i) \neq 2^{i-1}$ then we set subline$(v_i, v_{i+1}, \ldots, v_n, L) =$ FirstHalf(L').
- If $V'(v_i) = 2^{i-1}$ then we set subline$(v_i, v_{i+1}, \ldots, v_n, L) =$ SecondHalf(L').

Note that since FirstHalf and SecondHalf can be computed out in polynomial time, this means that subline can also be computed in polynomial time.

An important property of the reduction is that the output of subline$(v_i, v_{i+1}, \ldots, v_n)$ is a UniqueEOPL instance in which the longest possible line has length 2^{i-1}. This can be proved by induction. For the base case note that subline(L) allows potentials between 0 and $2^n - 1$. Each step of the recursion cuts this in half, meaning that subline$(v_i, v_{i+1}, \ldots, v_n, L)$ allows potentials

between 0 and $2^i - 1$. Since each edge in a line must always strictly increase the potential, this means that the longest possible line in subline$(v_i, v_{i+1}, \ldots, v_n, L)$ has length 2^{i-1}. This holds even if the instance has multiple lines.

The decode function. Given a point $(v_1, v_2, \ldots, v_n)$, let $L' = \text{subline}(v_1, v_2, \ldots, v_n, L)$. As we have argued above, we know that L' is an instance in which the longest possible line has length $2^{1-1} = 1$. Hence, the starting vertex of L', which is by definition v_1, is the end of a line. So we set $\text{decode}(v_1, v_2, \ldots, v_n) = v_1$. Since subline can be computed in polynomial time, this means that decode can also be computed in polynomial time. This completes the proof of Lemma 18.

C.2 Proof of Lemma 19

To prove this lemma, we must map every solution of the OPDC instance given by P and $\mathcal{D}$ to a solution of the UniqueEOPL instance $L = (S, P, V)$. We do this by enumerating the solution types for OPDC.

1. In solutions of type (O1) we have a point $p = (v_1, v_2, \ldots, v_n)$ such that $D_i(p) = \mathsf{zero}$ for all i. From this we can conclude, that decode(p) is the end of a line, since if decode(p) had a valid successor, then by definition at least one dimension i would satisfy $D_i(p) \neq \mathsf{zero}$. Hence decode$(p)$ is a solution of type (U1).

2. In solutions of type (OV1) we have two fixed points of a single i-slice. More specifically, we have an i-slice s and two points $p = (v_1, v_2, \ldots, v_n)$ and $q = (u_1, u_2, \ldots, u_n)$ in the slice s with $p \neq q$ such that $D_j(p) = D_j(q) = \mathsf{zero}$ for all $j \leq i$. Let v_j be the bit-string that uses dimension i, and let $L' = (S', P', V') = \text{subline}(v_{j+1}, v_{j+2}, \ldots, v_n, L)$. Since p and q both lie in s, we know that $v_l = u_l$ for all $l > j$. Observe that, since $D_l(p) = D_l(q) = \mathsf{zero}$ for all $l \leq j$, we have that v_1 through v_{j-1} encode a vertex that is at the end of the sub-line embedded into $(*, *, \ldots, v_j, v_{j+1}, \ldots, v_n)$, and u_1 through u_{j-1} encode a vertex that is at the end of the sub-line embedded into $(*, *, \ldots, u_j, v_{j+1}, \ldots, v_n)$. There are multiple cases to consider.

   (a) If $v_j = u_j$, then we have that $\text{subline}(v_j, v_{j+1}, \ldots, v_n, L) = \text{subline}(u_j, u_{j+1}, \ldots, u_n, L)$. Since $p \neq q$, there must be some pair of vertices v_l and u_l such that $v_l \neq u_l$, and note that since p and q are both at the end of their respective sub-lines, we have $V'(v_l) = V'(u_l)$, which also implies that $V(v_l) = V(u_l)$, which is a solution of type (UV3).

   (b) In this case we have $v_j \neq u_j$, and $D_l(p) = \mathsf{zero}$ for all l in the range $m(j-1) + 1 \leq l \leq mj$, but some index z in the range $m(j-1) + 1 \leq z \leq mj$ for which $D_l(q) \neq \mathsf{zero}$. This means that subline$(v_j, v_{j+1}, \ldots, v_n, L)$ is a second half instance of L', while subline$(u_j, u_{j+1}, \ldots, u_n, L)$ is a first half instance. Since q represents a vertex at the end of the corresponding line, we have that $S(\text{decode}(q))$ is the first vertex on the next half of L', meaning that $V'(S(\text{decode}(q))) = 2^{j-1}$. Moreover since p is on the second-half of the line, we have that $V'(v_j) = 2^{j-1}$, so we have $V(S(\text{decode}(q))) = V(v_j)$. This is a solution of type (UV3) so long as $S(\text{decode}(q)) \neq v_j$.

   To prove that this is the case, recall that the direction functions for q in the dimensions corresponding to u_j always point towards $S(\text{decode}(q))$. Since $u_j \neq v_j$, and since u_j and v_j lie in the same slice s, we know that they disagree on some dimension $l \leq i$. But we also have that $D_l(q) = \mathsf{zero}$ for all $l \leq i$, which can only occur if $S(\text{decode}(q))$ disagrees with v_j in some dimension. Hence we have shown that $S(\text{decode}(q)) \neq v_j$.

(c) If $v_j \neq u_j$, and $D_l(q) = \mathsf{zero}$, for all l in the range $m(i-1)+1 \leq l \leq mi$, but there exists an index z in the range $m(j-1)+1 \leq z \leq mj$ for which $D_l(p) \neq \mathsf{zero}$, then this is entirely symmetric to the previous case.

(d) In the last case, we have $v_j \neq u_j$, an index l_1 in the range $m(j-1)+1 \leq l_1 \leq mj$ with $D_{l_1}(p) \neq \mathsf{zero}$, and an index l_2 in the range $m(j-1)+1 \leq l_2 \leq mj$ with $D_{l_2}(p) \neq \mathsf{zero}$. Thus $\text{subline}(v_j, v_{j+1}, \ldots, v_n, L) = \text{subline}(u_j, u_{j+1}, \ldots, u_n, L)$, meaning that both points lie at the end of the first half of L'. Thus the direction functions at p point towards $S(\text{decode}(p))$, while the direction functions at q point towards $S(\text{decode}(q))$. Moreover we have $V'(S(\text{decode}(p))) = V'(S(\text{decode}(q))$, so to obtain a solution of type (UV3), we need to prove that $S(\text{decode}(p))) \neq S(\text{decode}(q)$.

This follows from the fact that $v_j \neq v_u$, and v_j and v_u's membership in the slice s. This means that they disagree on some dimension $l < i$, but since $D_a(p) = D_a(q) = \mathsf{zero}$ for all $a < i$, we must have $S(\text{decode}(p))) \neq S(\text{decode}(q)$.

3. For solutions of type (OV2), we have two points $p = (v_1, v_2, \ldots, v_n)$ and $q = (u_1, u_2, \ldots, u_n)$ that lie in an i-slice s with the following properties.

   - $D_j(p) = D_j(q) = \mathsf{zero}$ for all $j < i$,
   - $p_i = q_i + 1$, and
   - $D_i(p) = \mathsf{down}$ and $D_i(q) = \mathsf{up}$.

   As with case 2, let v_j be the bit-string that uses dimension i, and let $L' = \text{subline}(v_{j+1}, v_{j+2}, \ldots, v_n) = \text{subline}(u_{j+1}, u_{j+2}, \ldots, u_n)$. Since $D_i(p) \neq \mathsf{zero}$ and $D_i(q) \neq \mathsf{zero}$, we must have that p and q are both points on the first half of L'. Furthermore, $\text{decode}(p)$ and $\text{decode}(q)$ must both be at the end of the first half, since $D_l(p) = D_l(q) = \mathsf{zero}$ for all $l < i$. Hence, $V'(\text{decode}(p)) = V'(\text{decode}(q)) = 2^{j-1} - 1$, which also implies that $V(\text{decode}(p)) = V(\text{decode}(q))$. So to obtain a solution of type (UV3), we just need to prove that $\text{decode}(p) \neq \text{decode}(q)$.

   To prove this, we observe that the direction function in dimension i should be oriented towards $S(\text{decode}(p))$ for the point p, and $S(\text{decode}(q))$ for the point q. However, since $D_i(p)$ and $D_i(q)$ both point away from their points, and since $p_i \neq q_i$, this must mean that $S(\text{decode}(p)) \neq S(\text{decode}(q))$, which also implies that $\text{decode}(p) \neq \text{decode}(q)$.

4. We claim that solutions of type (OV3) are impossible. A solution of type (OV3) requires that we have a point p with either $p_i = 0$ and $D_i(p_i) = \mathsf{down}$, or $p_i = 1$ and $D_i(p_i) = \mathsf{up}$. Our construction never sets $D_i(p_i) = \mathsf{down}$ when $p_i = 0$, and it never sets $D_i(p_i) = \mathsf{up}$ when $p_i = 1$. So solutions of type (OV3) cannot occur.

To see that the reduction is promise-preserving, it suffices to note that we only ever map solutions of type (U1) onto solutions of type (O1). Thus, if the original instance has only solutions of type (U1), then the resulting OPDC instance will only have solutions of type (O1).

D Proofs for Section 5

D.1 Proof of Lemma 26

We denote the bit-length of an integer $n \in \mathbb{Z}$ as $b(n) \triangleq \lceil \log_2 n \rceil$. We extend the definition to the rationals by defining the bit-length for $x \in \mathbb{Q}$ as the minimum number of bits needed to represent

the numerator and denominator of some representation of x as a ratio of integers:

$$b(x) \triangleq \min_{\substack{p,q\in\mathbb{Z} \\ x=p/q}} (b(p) + b(q)) .$$

We extend this notion of bit-length to matrices by defining the bit-length $b(M)$ of a matrix $M \in \mathbb{Q}^{m\times n}$ by $b(M) \triangleq \max_{i,j} b(M_{ij})$ and to vectors by defining $b(v) \triangleq \max_i b(v_i)$ for $v \in \mathbb{Q}^n$.

LinearFIXP circuits and LCPs. The goal of this proof is to find a tuple $(k_1, k_2, \ldots, k_d)$ such that the lemma holds. We utilize a lemma from [44] which asserts that every LinearFIXP circuit can be transformed in polynomial time into an LCP with bounded bit-length, such that solutions to the LCP capture exactly the fixpoints of the circuit.

For any LinearFIXP circuit $C : [0,1]^d \to [0,1]^d$ with gates $g_1, \ldots, g_m$ and constants $\zeta_1, \ldots, \zeta_q$, we define the size of C by

$$\text{size}(C) \triangleq d + m + \sum_{i=1}^{q} b(\zeta_i).$$

Lemma 38 ([44])**.** *Let $C : [0,1]^d \to [0,1]^d$ be a* LinearFIXP *circuit. We can produce in polynomial time an LCP defined by an $n \times n$ matrix M_C and n-dimensional vector $\boldsymbol{q}_C$ for some n with $d \le n \le$ size(C) such that there is a bijection between solutions of the LCP $(M_C, \boldsymbol{q}_C)$ and fixpoints of C. Moreover, to obtain a fixpoint x of C from a solution $\mathbf{y}$ to the LCP $(M_C, \boldsymbol{q}_C)$, we can just set $x = (\mathbf{y}_1, \ldots, \mathbf{y}_d)$. Furthermore, $b(M_C)$ and $b(\boldsymbol{q}_C)$ are both at most $O(n \times$ size$(C))$.*

Crucially the construction interacts nicely with fixing inputs; if C' denotes a circuit where one of the inputs of C is fixed to be some number x, we can bound the bit-length of $M_{C'}$ and $\boldsymbol{q}_{C'}$ in terms of the bit-lengths of M_C, $\boldsymbol{q}_C$, and x.

Observation 39. $b(M_{C'}) \le b(M_C)$.

Observation 40. $b(\boldsymbol{q}_{C'}) \le \max\{b(\boldsymbol{q}_C), b(M_C) + b(x)\} + 1$.

In other words, the bit-length of $M_{C'}$ does not depend on x, and is in fact at most the bit-length of M_C, and the bit-length of $\boldsymbol{q}_{C'}$ is bounded by the worse of the bit-lengths of $\boldsymbol{q}_C$ or the sum of the bit-lengths of M_C and x plus an additional bit.

Bounding the bit-length of a solution of an LCP. We now prove two technical lemmas about the bit-length of any solution to an LCP. We begin with the following lemma regarding the bit-length of a matrix inverse.

Lemma 41. *Let $A \in \mathbb{Z}^{n\times n}$ be a square matrix of full rank, and let $B = \max_{ij} |A_{ij}|$ be the largest absolute value of an entry of A. Then $b(A^{-1}) \le 2n \log B + n \log n$.*

Proof. We have $A^{-1} = \frac{1}{\det(A)} \text{adj}(A)$. Each entry of adj$(A)$ is a cofactor of A, each of which is a (possibly negated) determinant of an $(n-1) \times (n-1)$ submatrix of A. It is a well-known corollary of Hadamard's inequality that $|\det(A)| \le B^n n^{n/2}$. The same bound obtains for each submatrix. We can write entry A^{-1}_{ij} as p/q where $p = \text{adj}(A)_{ij}$ and $q = \det(A)$ are both integers. We have $b(p), b(q) \le n \log B + (n/2) \log n$. The lemma follows immediately. □

We now use this to prove the following bound on the bit-length of a solution of an LCP.

Lemma 42. *Let $M \in \mathbb{Q}^{n\times n}$ and $\boldsymbol{q} \in \mathbb{Q}^n$. Let $\mathbf{y}$ be a solution to the LCP $(M, \boldsymbol{q})$. Then*

$$b(\mathbf{y}) \leq (5n+2)\log n + (4n+1)b(M) + n + b(\boldsymbol{q}).$$

Proof. We first note that if an LCP has a solution, then it has a vertex solution. Let $(\mathbf{y}, \mathbf{w})$ be such a vertex solution of the LCP and, as in Section 5.3, let $\alpha = \{i \mid \mathbf{y}_i > 0\}$, and let $A = A_\alpha$ be defined according to (3). We have that A is guaranteed to be invertible, and we have that $A\boldsymbol{x} = \boldsymbol{q}$, with $\mathbf{y}_i = \boldsymbol{x}_i$ for $i \in \alpha$ and $\mathbf{y}_i = 0$ for $i \notin \alpha$, so we have $b(\mathbf{y}) \leq b(\boldsymbol{x})$. Also note that we have $b(A) \leq b(M)$, since the entries in columns that take the value of e_i have constant bit-length.

We must transform A into an integer matrix in order to apply Lemma 41. Let ℓ denote the least common multiple of the denominators of the entries in A. Note that $\ell \leq n^2 2^{b(A)}$ and hence $b(\ell) \leq b(A) + 2\log(n)$. Our matrix equation above can be rewritten as $\ell A\boldsymbol{x} = \ell\boldsymbol{q}$, where (ℓA) is an integer matrix. Hence we have $\boldsymbol{x} = (\ell A)^{-1}(\ell\boldsymbol{q})$.

If B denotes the maximum entry of ℓA, then $B \leq \ell 2^{b(A)}$. So by Lemma 41 every entry of $(\ell A)^{-1}$ can be represented with integer numerators and denominators bounded by

$$B^n n^{n/2} \leq (\ell 2^{b(A)})^n n^{n/2} \leq (n^2 2^{2b(A)})^n n^{n/2}.$$

From this bound we obtain

$$\begin{aligned} b((\ell A)^{-1}) &\leq 2\log\left((n^2 2^{2b(A)})^n n^{n/2}\right) \\ &= 2\left(n\left(2\log n + 2b(A)\right) + (n/2)\log n\right) \\ &\leq 5n\log n + 4nb(A). \end{aligned}$$

Each entry of $\mathbf{y}$ consists of the sum of n entries of $(\ell A)^{-1}$ each of which is multiplied by an entry of $\boldsymbol{q}$, followed at the end with a multiplication by ℓ. We get the following bound on the bit-length of $\mathbf{y}$.

$$\begin{aligned} b(\mathbf{y}) &\leq b((\ell A)^{-1}) + b(\boldsymbol{q}) + n + b(\ell) \\ &\leq 5n\log n + 4nb(A) + n + b(A) + 2\log n \\ &\leq (5n+2)\log n + (4n+1)b(A) + n + b(\boldsymbol{q}) \end{aligned}$$

□

Fixing the grid size. We shall fix the grid size iteratively, starting with k_d, and working downwards. At each step, we will have a space that is partially continuous and partially discrete. Specifically, after the iteration in which we fix k_i, the dimensions $j < i$ will allow any number in $[0, 1]$, while the dimensions $j \geq i$ will only allow the points with denominator k_j. If I_k denotes the set of all rationals with denominator k, then we will denote this as

$$P(k_i, k_{i+1}, \ldots, k_d) = [0,1]^{i-1} \times I_{k_i} \times I_{k_{i+1}} \times \cdots \times I_{k_d}.$$

Moreover, after fixing k_i, we will have that the property required by Lemma 26 holds for all j-slices s with $j \geq i$. Specifically, that if x is a fixpoint of s according to f, then there exists a $p \in P(k_i, k_{i+1}, \ldots, k_d)$ that is a fixpoint of s according to $\mathcal{D}$.

We start by bounding the bit-length of a solution to the PL-Contraction problem computed one coordinate at a time. Given a PL-circuit C, we use Lemma 38 to produce an LCP defined by $M \in \mathbb{Q}^{n\times n}$ and $\boldsymbol{q} \in \mathbb{Q}^n$. Now let $x_1, \ldots, x_n$ be formal variables representing the inputs to our circuit C. We want to determine parameters $\kappa_1, \ldots, \kappa_d$ such that if we fix variables $x_{i+1}, ..., x_d$ to

values with bit-lengths $b(x_{i+1}), \ldots, b(x_d)$ where $b(x_j) \leq \kappa_j$ for each $j \in \{i+1, \ldots, d\}$, then any fixpoints of C with respect to the free variables $x_1, \ldots, x_i$ will have bit-lengths $b(x_1), \ldots, b(x_i)$ with $b(x_j) \leq \kappa_j$ for $j \in [i]$.

We set

$$\kappa_i \triangleq (d - i + 1)((5n + 2) \log n + n + (4n + 2)b(M) + 1) + b(\boldsymbol{q}).$$

Note that $\kappa_1 \geq \kappa_2 \geq \cdots \geq \kappa_d$.

To prove that these bounds suffice, we use induction on i, starting from $i = d$. First, we observe that by Lemma 42, we have $b(\mathbf{y}) \leq (5n + 2) \log n + (4n + 1)b(M) + b(\boldsymbol{q}) = \kappa_d - 1 \leq \kappa_d$ for any solution $\mathbf{y}$ to the LCP $(M, \boldsymbol{q})$. Moreover each fixpoint x of C corresponds to a solution $\mathbf{y}$ to the LCP by Lemma 38, so we have $b(x_1), \ldots, b(x_d) \leq \kappa_d$ which implies the weaker claim that all fixpoints can be found by choosing $b(x_i) \leq \kappa_i$ for all $i \in [d]$, since $\kappa_i \geq \kappa_d$ for all $i \in [d]$.

Now we handle the inductive case. For $i = 1, \ldots, d$, let $M^{(i)}, \boldsymbol{q}^{(i)}$ be the pair defining the LCP after x_{i+1} through x_d are fixed to values with bit-lengths bounded by $\kappa_{i+1}, \ldots, \kappa_d$, respectively. This pair will of course depend on the values $x_{i+1}, \ldots, x_d$, but since the bit-length of $M^{(i)}$ and $\boldsymbol{q}^{(i)}$ depend only on the bit-lengths of the fixed values, we can ignore the values of $x_{i+1}, \ldots, x_d$ as long as we have bounds on their bit-lengths and as long as we restrict our attention to the bit-lengths of $M^{(i)}$ and $\boldsymbol{q}^{(i)}$ and not the values themselves.

Using Lemma 42 we know that the solution to the LCP has $b(x_i) \leq (5n + 2) \log n + (4n + 1)b(M^{(i)}) + b(\boldsymbol{q}^{(i)})$. Moreover, we obtained $M^{(i)}$ by repeatedly fixing inputs to C, so the repeated application of Observation 39 implies that $b(M^{(i)}) \leq b(M^{(i+1)}) \leq b(M)$. Moreover, Observation 40 gives us

$$b(\boldsymbol{q}^{(i)}) \leq \max\left\{b(\boldsymbol{q}^{(i+1)}), b(M^{(i+1)}) + b(x_{i+1})\right\} + 1.$$

By a simple argument, we can also show that $\kappa_{i+1} \geq b(\boldsymbol{q}^{(i+1)})$ so that the above bound can be simplified to

$$b(\boldsymbol{q}^{(i)}) \leq b(M^{(i+1)}) + b(x_{i+1}) + 1$$

at which point we can use our observation and the bound $b(x_{i+1}) \leq \kappa_{i+1}$ to obtain

$$b(\boldsymbol{q}^{(i)}) \leq b(M) + \kappa_{i+1} + 1.$$

Now we use the bound from Lemma 42 again to obtain

$$\begin{aligned} b(x_i) &\leq (5n + 2) \log n + (4n + 1)b(M^{(i)}) + b(\boldsymbol{q}^{(i)}) \\ &\leq (5n + 2) \log n + (4n + 1)b(M) + b(M) + \kappa_{i+1} + 1 \end{aligned}$$

and we now use the definition of κ_{i+1} to conclude

$$\begin{aligned} b(x_i) &\leq (5n + 2) \log n + (4n + 1)b(M) + b(M) \\ &\quad + (d - (i + 1) + 1)((5n + 2) \log n + n + (4n + 2)b(M) + 1) + b(\boldsymbol{q}) + 1 \\ &= (d - i + 1)((5n + 2) \log n + (4n + 2)b(M) + 1) + b(\boldsymbol{q}) \\ &= \kappa_i. \end{aligned}$$

We conclude that all solutions to the LCP $(M^{(i)}, \boldsymbol{q}^{(i)})$ have free coordinates with bit-length at most κ_i, and similarly for the fixpoints of C. Thus, every fixpoint of C with respect to the free variables can be found by choosing $x_1, \ldots, x_i$ to have bit-lengths at most $\kappa_1, \ldots, \kappa_i$, respectively, when the bit-lengths of $x_{i+1}, \ldots, x_d$ are chosen to have bit-lengths at most $\kappa_{i+1}, \ldots, \kappa_d$.

To conclude the proof of Lemma 26, we need to choose $k_1, \ldots, k_d$ so that every i-slice with fixed coordinates in $P(k_1, \ldots, k_d)$ has a fixpoint also on $P(k_1, \ldots, k_d)$ when we map points $x \in P(k_1, \ldots, k_d)$ to $[0,1]^d$ by

$$x \mapsto (x_1/k_1, \ldots, x_d/k_d).$$

We set $k_i \triangleq 2^{\kappa_i}$. Now a point $x \in P(k_1, \ldots, k_d)$ corresponds to a point $y \in [0,1]^d$ with $b(y_i) \leq 2\kappa_i$ for all $i \in [d]$ and any i-slice where the fixed coordinates satisfy the bit-length bounds will have fixpoints for the remaining variables with all coordinates satisfying the bit-length bounds.

Finally, we observe that for each $i \in [d]$, $b(k_i) = \kappa_i \leq \kappa_1 = O(\mathrm{poly}(\mathrm{size}(C)))$, so each of the k_i can be represented using polynomially many bits in the size of the circuit C. This completes the proof of Lemma 26.

D.2 Proof of Lemma 27

Before we begin, we first prove two auxiliary lemmas concerning contraction maps.

Slice Restrictions of Contraction Maps Next we define some further notation for slice restrictions of a contraction map. We define the set of fixed coordinates of a slice $s \in \mathsf{Slice}_d$, to be $\mathrm{fixed}(s) = \{i \in [d] \mid s_i \neq *\}$, and we define the set of free coordinates to be $\mathrm{free}(s) = [d] \setminus \mathrm{fixed}(s)$. Thus, an i-slice is a slice $s \in \mathsf{Slice}_d$ for which $\mathrm{free}(s) = [i]$. We say that a slice is a *k-dimensional slice* if $|\mathrm{free}(s)| = k$.

We can define the *slice restriction* of a function $f : [0,1]^d \to [0,1]^d$ with respect to a slice $s \in \mathsf{Slice}_d$, denoted $f_{|s}$, to be the function obtained by fixing the coordinates $\mathrm{fixed}(s)$ according to s, and keeping the coordinates of $\mathrm{free}(s)$ as arguments. To simplify usage of $f_{|s}$ we formally treat $f_{|s}$ as a function with d arguments, where the coordinates in $\mathrm{fixed}(s)$ are ignored. Thus, we define $f_{|s} : [0,1]^d \to [0,1]^d$ by

$$f_{|s}(x) = f(y) \quad \text{where } y_i = \begin{cases} s_i & \text{if } i \in \mathrm{fixed}(s) \\ x_i & \text{if } i \in \mathrm{free}(s). \end{cases}$$

Let $\mathrm{free}(s) = \{i_1, \ldots, i_k\}$. We also introduce a variant of $f_{|s}$ when we want to consider the slice restriction as a lower-dimensional function, $\tilde{f}_{|s} : [0,1]^d \to [0,1]^{|\mathrm{free}(s)|}$ defined by

$$\tilde{f}_{|s}(x) = \left(\left(f_{|s}(x)\right)_{i_1}, \ldots, \left(f_{|s}(x)\right)_{i_k} \right).$$

We can also define slice restrictions for vectors in the natural way:

$$\left(x_{|s}\right)_i = \begin{cases} s_i & \text{if } s_i \neq * \\ x_i & \text{otherwise.} \end{cases}$$

Finally, we use $\tilde{x}_{|s}$ to denote projection of x onto the coordinates in $\mathrm{free}(s)$:

$$\tilde{x}_{|s} = (x_{i_1}, \ldots, x_{i_k}).$$

We now extend the definition of a contraction map to a slice restriction of a function in the obvious way. We say that $\tilde{f}_{|s}$ is a contraction map with respect to a norm $\|\cdot\|$ with Lipschitz constant c if for any $x, y \in [0,1]^d$ we have

$$\left\| \tilde{f}_{|s}(x) - \tilde{f}_{|s}(y) \right\| \leq c \left\| \tilde{x}_{|s} - \tilde{y}_{|s} \right\|.$$

Slice restrictions will prove immensely useful through the following observations:

Lemma 43. *Let $f : [0,1]^d \to [0,1]^d$ be a contraction map with respect to $\|\cdot\|_p$ with Lipschitz constant $c \in (0,1)$. Then for any slice $s \in \mathsf{Slice}_d$, $\tilde{f}_{|s}$ is also a contraction map with respect to $\|\cdot\|_p$ with Lipschitz constant c.*

Proof. For any two vectors $x, y \in [0,1]^d$ we have

$$\begin{aligned}\left\|\tilde{f}_{|s}(x) - \tilde{f}_{|s}(y)\right\|_p &\leq \left\|f(x_{|s}) - f(y_{|s})\right\|_p \\ &\leq c\left\|x_{|s} - y_{|s}\right\|_p \\ &= c\left\|\tilde{x}_{|s} - \tilde{y}_{|s}\right\|_p \ .\end{aligned}$$

□

Since slice restrictions of contraction maps are themselves contraction maps in the sense defined above, they have unique fixpoints, up to the coordinates of the argument which are fixed by the slice and thus ignored. We nevertheless refer to the *unique fixpoint of a slice restriction of a contraction map*, which is the unique point $x \in [0,1]^d$ such that

$$\tilde{f}_{|s}(x) = \tilde{x}_{|s} \quad \text{and} \quad x = x_{|s}\,.$$

Lemma 44. *Let $f : [0,1]^d \to [0,1]^d$ be a contraction map with respect to $\|\cdot\|_p$ with Lipschitz constant $c \in (0,1)$. Let $s, s' \in \mathsf{Slice}_d$ be such that $\mathrm{fixed}(s') = \mathrm{fixed}(s) \cup \{i\}$ and $s_j = s'_j$ for all $j \in \mathrm{fixed}(s)$. Let $x, y \in [0,1]^d$ be the unique fixpoints of $\tilde{f}_{|s}$ and $\tilde{f}_{|s'}$, respectively. Then $(x_i - y_i)(f(y)_i - y_i) > 0$.*

Proof. We prove this by contradiction. Without loss of generality, assume towards a contradiction that $x_i \leq y_i$ and that $f(y)_i > y_i$. Then we have

$$\begin{aligned}\|f(y) - f(x)\|_p^p &= \|(f(y)_1, \ldots, f(y)_d) - (f(x)_1, \ldots, f(x)_d)\|_p^p \\ &= \sum_{j \in \mathrm{fixed}(s)} |s_j - s_j|^p + \sum_{j \in \mathrm{free}(s')} |y_j - x_j|^p + |f(y)_i - x_i|^p \\ &> \sum_{j \in \mathrm{fixed}(s)} |s_j - s_j|^p + \sum_{j \in \mathrm{free}(s')} |y_j - x_j|^p + |y_i - x_i|^p \\ &= \|y - x\|_p^p \ ,\end{aligned}$$

which contradicts the fact that f is a contraction map. The lemma follows. □

The proof of Lemma 27 The statement of the lemma says that we can assume that f is contracting with respect to some ℓ_p norm, and that we have an i-slice s and two points p, q in s satisfying the following conditions.

- $D_j(p) = D_j(q) = \mathsf{zero}$ for all $j < i$,
- $p_i = q_i + 1$, and
- $D_i(p) = \mathsf{down}$ and $D_i(q) = \mathsf{up}$.

To translate p and q from the grid to the $[0,1]^n$ space, we must divide each component by the grid length in that dimension. Specifically, we define the point $a \in [0,1]^n$ so that $a_i = p_i/k_i$ for all i, and the point $b \in [0,1]^n$ such that $b_i = q_i/k_i$ for all i.

Lemma 43 states that if f is contracting with respect to an ℓ_p norm, then the restriction of f to the slice s is also contracting in that ℓ_p norm. Hence, f must have a fixpoint in the slice s. Let $x \in [0,1]^n$ denote this fixpoint.

By definition we have that $(f(x) - x)_j = 0$ for all $j \leq i$, and we also have $(f(a) - a)_j = 0$ for all $j < i$ from the fact that p is a fixpoint of its $(i-1)$-slice. So we can apply Lemma 44 to x and a, and from this we get that $(x_i - a_i) \cdot (f(a)_i - a_i) > 0$. Since $D_i(p) = \mathsf{down}$, we have that $f(a)_i < a_i$, and hence $(f(a)_i - a_i) < 0$. Therefore, we can conclude that $(x_i - a_i) < 0$, which implies that $x_i < a_i$.

By the same reasoning we can apply Lemma 44 to x and b, and this gives us that $(x_i - b_i) \cdot (f(b)_i - b_i) > 0$. This time we have $D_i(q) = \mathsf{up}$, which implies that $f(b)_i > b_i$, and hence $(f(b)_i - b_i) > 0$. So we can conclude that $(x_i - b_i) > 0$, meaning that $x_i > b_i$.

Hence we have shown that $b_i < x_i < a_i$, and so the point x satisfies the conditions of the lemma. This completes the proof of Lemma 27.

D.3 Proof of Lemma 29

We must map every possible solution of the OPDC instance back to a solution of the PL-Contraction instance. We will enumerate all solution types for OPDC.

- In solutions of type (O1), we have a point $p \in P$ such that $D_i(p) = \mathsf{zero}$ for all i. This means that the point x, where $x_i = p_i/k_i$ for all i, satisfies $f(x) - x = 0$. Hence x is a solution of type (CM1).

- In solutions of type (OV1) we have i-slice s and two points $p, q \in P_s$ with $p \neq q$ such that $D_j(p) = D_j(q) = \mathsf{zero}$ for all $j \leq i$. Let a and b be the two corresponding points in $[0,1]^n$, meaning that $a_i = p_i/k_i$ for all i, and $b_i = q_i/k_i$ for all i.

  We first first consider the contraction property in the $[0,1]^i$ space defined by the slice s. Let $\|a - b\|_p^i$ denote the distance between a and b according to the ℓ_p norm restricted to the $[0,1]^i$ space, and likewise let $\|f(a) - f(b)\|_p^i$ be the distance between $f(a)$ and $f(b)$ in that space. Since $D_j(p) = D_j(q) = \mathsf{zero}$ for all $j \leq i$, this means that we have $\|f(a) - f(b)\|_p^i = \|a - b\|_p^i$, which is a clear violation of contraction in the space $[0,1]^i$.

  To see that a and b also violate contraction in $[0,1]^d$, observe that $\|a - b\|_p^i = \|a - b\|_p$, because a and b lie in the same i-slice, meaning that $a_j = b_j$ for all $j > i$. Furthermore, we have $\|f(a) - f(b)\|_p \geq \|f(a) - f(b)\|_p^i$, since adding in extra dimensions can only increase the distance in an ℓ_p norm. Hence we have $\|f(a) - f(b)\|_p \geq \|a - b\|_p$, which is a violation of contraction, giving us a solution of type (CMV1).

- Violations of type (OV2) map directly to violations of type (CMV3), as discussed in the main body.

- Violations of type (OV3) give us a point p such that $p_i = 0$ and $D_i(p) = \mathsf{down}$, or $p_i = k_i$ and $D_i(p) = \mathsf{up}$. In both cases this means that $f(p) \notin [0,1]^d$, and so we have a violation of type (CMV2).

To see that the reduction is promise-preserving, it suffices to note that solutions of type (CM1) are only ever mapped on to solutions of type (O1). Hence, if the input problem is a contraction map, the resulting OPDC instance only has solutions of type (CM1).