Unique Sink Orientations of Grids is in Unique End of Potential Line

Michaela Borzechowski[*1] and Wolfgang Mulzer[†2]

1   **Institut für Informatik, Freie Universität Berlin**
    `michaela.borzechowski@fu-berlin.de`
2   **Institut für Informatik, Freie Universität Berlin**
    `mulzer@inf.fu-berlin.de`

──── **Abstract** ────

The complexity classes *Unique End of Potential Line* (UEOPL) and its promise version PromiseUEOPL were introduced in 2018 by Fearnly et al. [4]. PromiseUEOPL captures search problems where the instances are promised to have a unique solution. UEOPL captures total search versions of these promise problems. The promise problems can be made total by defining *violations* that are returned as a short certificate of an unfulfilled promise.

Grid-USO is the problem of finding the sink in a grid with a unique sink orientation. It was introduced by Gärtner et al. [7]. We describe a promise preserving reduction from Grid-USO to Unique Forward EOPL, a UEOPL-complete problem. Thus, we show that Grid-USO is in UEOPL and its promise version is in PromiseUEOPL.

1   Introduction

Many tasks in computer science are naturally formulated as *search problems*, where the goal is to *find* a "solution" for a given instance. *Promise problems*, where it is guaranteed that we see only instances that have a certain property, are also an intuitive approach to formulate certain tasks. Nonetheless, standard complexity theory works with decision problems, and it may happen that the computational complexity of the decision problem and the search problem are not equivalent. In particular, this is the case for problems for which it is guaranteed that a solution always exists. The complexity of such *total* search problems has been studied since at least 1991, when Megiddo and Papadimitriou defined the class *Total Function NP* (TFNP) [9]. Here, we consider a subclass of TFNP, namely the complexity class *Unique End of Potential Line* (UEOPL). It contains some interesting problems from computational geometry for which no polynomial time algorithm is known but which are unlikely to be NP-hard, for example α-Ham-Sandwich [3] and Arrival [6]. Currently UEOPL contains one complete problem: One-Permutation-Discrete-Contraction. The problem Grid-USO is an abstraction of the simplex algorithm executed on a *general P-Matrix linear complementarity problem*. Since UEOPL was introduced only recently, in 2018, by Fearnly et al. [4], the knowledge about this class is still very limited. We show that the problem Grid-USO lies in UEOPL, making progress towards elucidating the nature of this class and the problems in it. In particular, with more problems that are known to lie in UEOPL, it becomes more likely that additional complete problems are found.

2   Unique End of Potential Line

A UNIQUE FORWARD EOPL instance is defined by two circuits $S : \{0,1\}^d \to \{0,1\}^d$ and $c : \{0,1\}^d \to \{0,1\}^m$. A circuit is a compact polynomial sized representation of information which in a map would be exponentially large (for a more formal definition, see [1, Definition 6.1]). The circuits represent a directed graph G. The vertices of G are bit strings $v \in \{0,1\}^d$ with $S(v) \neq v$, and there is a directed edge from a node v to a node w if and only if $S(v) = w$ and $c(w) > c(v)$ (where the result of c is interpreted as an m-bit positive integer). Thus, each node in G has out-degree of at most 1, and the circuit S computes the candidate *successor* of a node. The circuit c assigns a positive *cost* (also called *potential*) from $\{0, \ldots, 2^m - 1\}$ to every node, and all edges go in the direction of strictly increasing cost. Furthermore, we define that the bit string 0^d is a node of G, and that $c(0^d) = 0$. These properties ensure that G is a collection of directed paths, which we call *lines*, and that 0^d is the start vertex of a line. Our computational task is as follows: if the nodes of G form a single line (that necessarily starts in 0^d), then we should find the unique end node of this line — the *sink*. Otherwise, we should find a *violation certificate* that shows that G does not consist of a single line. UNIQUE FORWARD EOPL is a total search problem. There always exists a valid sink or a violation. Note that there might exist a valid sink and a violation simultaneously. The promise version of UNIQUE FORWARD EOPL is: under the promise that no violations exist for the given instance, find the unique end of the line. The formal definition of the total search problem UNIQUE FORWARD EOPL is as follows:
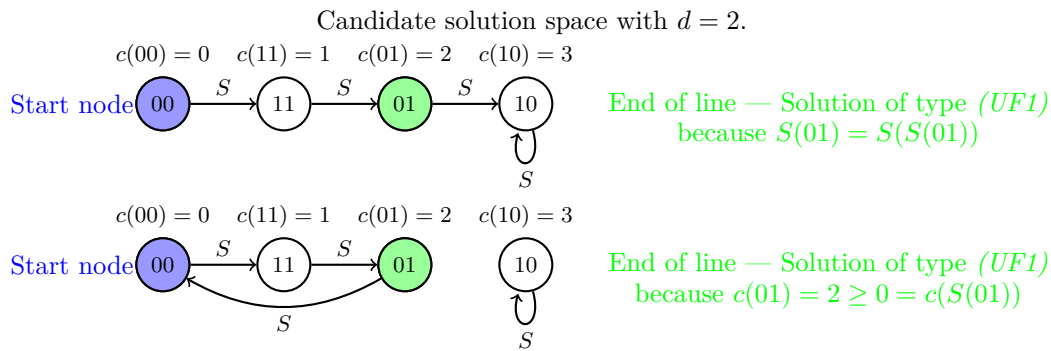
▶ **Definition 2.1.** ([5, Definition 10]) Let $d, m \in \mathbb{N}^+$ with $m \geq d$. Given Boolean circuits $S : \{0,1\}^d \to \{0,1\}^d$ and $c : \{0,1\}^d \to \{0, 1, \ldots, 2^m - 1\}$ which must have the property that $S(0^d) \neq 0^d$ and $c(0^d) = 0$, find one of the following:

(UF1) A bit string $v \in \{0,1\}^d$ with $S(v) \neq v$ and either $S(S(v)) = S(v)$ or $c(S(v)) \leq c(v)$. Then, $S(v)$ is not a valid node and v is a sink node in G and thus a valid solution.

(UFV1) Two bit strings $v, w \in \{0,1\}^d$ with $v \neq w$, $S(v) \neq v$, $S(w) \neq w$ and either *(a)* $c(v) = c(w)$ or *(b)* $c(v) < c(w) < c(S(v))$. Then, v and w are two different nodes that violate the promise of the strictly increasing potential.

(UFV2) Two nodes $v, w \in \{0,1\}^d$ such that v is a solution of type *(UF1)*, $v \neq w$, $S(w) \neq w$ and $c(v) < c(w)$. This encodes a break in the line. The node v is the end of one line, but there exists a different line with a node w that has higher cost than v.
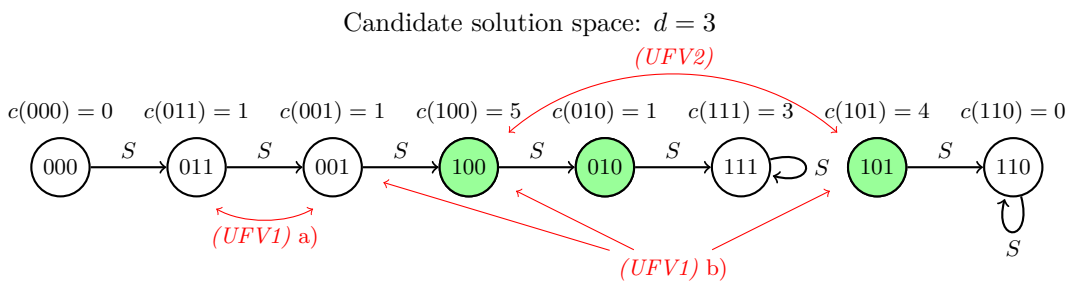
Two examples of an instances with no violations are shown in Figure 1 and an instance with all types of violations can be seen in Figure 2.

▶ **Definition 2.2.** ([5, Definition 7]) Let R be a search problem, and $\mathcal{I}^R \subseteq \{0,1\}^*$ be the set of all instances for R. For an instance $I \in \mathcal{I}^R$, let $\mathcal{S}^R(I)$ be the set of candidate solutions of I. A search problem R can be reduced by a *promise preserving Karp reduction in polynomial time* to a search problem R' if there exist two polynomial-time functions $f : \mathcal{I}^R \to \mathcal{I}^{R'}$ and $g : \mathcal{I}^R \times \mathcal{S}^{R'}(f(I)) \to \mathcal{S}^R(I)$ such that if s' is a violation of $f(I)$, then $g(I, s')$ is a violation of I and if s' is a valid solution of $f(I)$, then $g(I, s')$ is a valid solution or a violation of I. We are given an instance of R from which we construct with f an instance of problem R'. If we then solve R', we can re-translate the solution from R' to a solution of R with g. Promise preserving reductions are transitive.

▶ **Definition 2.3.** ([4]) The search problem complexity class UEOPL contains all problems that can be reduced in polynomial time to UNIQUE FORWARD EOPL. Thus, the complexity

Candidate solution space with $d = 2$.

$c(00) = 0$ $c(11) = 1$ $c(01) = 2$ $c(10) = 3$

Start node 00 $\xrightarrow{S}$ 11 $\xrightarrow{S}$ 01 $\xrightarrow{S}$ 10 $\circlearrowright S$

End of line — Solution of type *(UF1)* because $S(01) = S(S(01))$

$c(00) = 0$ $c(11) = 1$ $c(01) = 2$ $c(10) = 3$

Start node 00 $\xrightarrow{S}$ 11 $\xrightarrow{S}$ 01 10 $\circlearrowright S$

End of line — Solution of type *(UF1)* because $c(01) = 2 \geq 0 = c(S(01))$

Figure 1 UNIQUE FORWARD EOPL instances that form a valid line without violations.

Candidate solution space: $d = 3$

(UFV2)

$c(000) = 0$ $c(011) = 1$ $c(001) = 1$ $c(100) = 5$ $c(010) = 1$ $c(111) = 3$ $c(101) = 4$ $c(110) = 0$

000 $\xrightarrow{S}$ 011 $\xrightarrow{S}$ 001 $\xrightarrow{S}$ 100 $\xrightarrow{S}$ 010 $\xrightarrow{S}$ 111 $\circlearrowright S$ 101 $\xrightarrow{S}$ 110 $\circlearrowright S$

(UFV1) a)

(UFV1) b)

Figure 2 A UNIQUE FORWARD EOPL instance line with all types of violations.

class UEOPL captures all total search problems where the space of candidate solutions has the structure of a unique line with increasing cost. The relationship of UEOPL to other classes is shown in Figure 3.

PromiseUEOPL is the promise version of the search problem class UEOPL. When containment of a search problem R in UEOPL is shown via a promise preserving reduction, the promise version of R is contained in PromiseUEOPL.

3   Unique Sink Orientations of Grids

▶ **Definition 3.1.** ([7, p. 206]) Let $n, d \in \mathbb{N}^+$. Let $\mathcal{M} = \{1, \ldots, n\}$ be an ordered set of integers called *directions* and $K = (\kappa_1, \ldots, \kappa_d)$ be a partition of $\mathcal{M}$ with κ_i being ordered and $|\kappa_i| \geq 2$ for all *dimensions* $i = 1, \ldots, d$. The *d-dimensional grid* Γ is the undirected graph derived from $\Gamma = (\mathcal{M}, K)$ with a set of vertices $V := \{v \subseteq \mathcal{M} \mid i = 1, \ldots, d, |v \cap \kappa_i| = 1\}$ and a set of edges $E := \{\{p, q\} \mid p, q \in V, |p \oplus q| = 2\}$, where $\oplus$ denotes the *symmetric difference*.

▶ **Definition 3.2.** ([7, p. 211]) The *outmap function* $\sigma : V \to Powerset(\mathcal{M})$ defines an *orientation* of the edges of a grid Γ. For each point $p \in V$, the set $\sigma(p)$ contains all directions to which p has outgoing edges. The edges of p for all other directions are incoming. In particular, we have $\sigma(p) \cap p = \emptyset$. An outmap σ is called *unique sink orientation* of Γ if all nonempty induced subgrids of Γ have a unique sink.
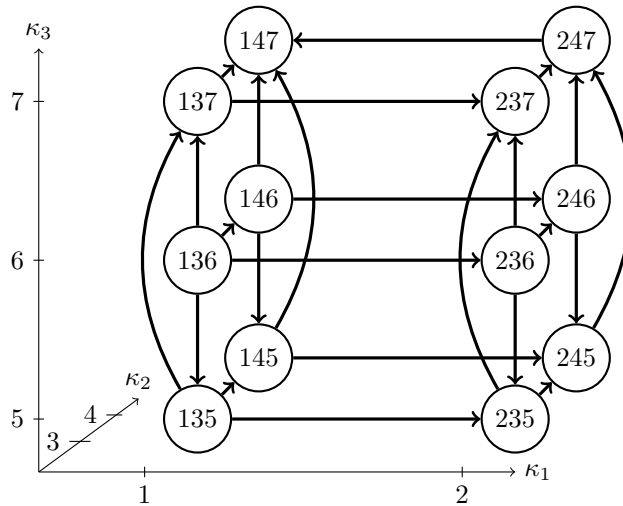
▶ **Definition 3.3.** ([7, Definition 2.13]) The *refined index* r_σ of an outmap σ with $r_\sigma : V \to \{0, \ldots, |\kappa_1| - 1\} \times \cdots \times \{0, \ldots, |\kappa_d| - 1\}$ is defined as: $r_\sigma(p) := (|\sigma(p) \cap \kappa_1|, \ldots, |\sigma(p) \cap \kappa_d|)$.
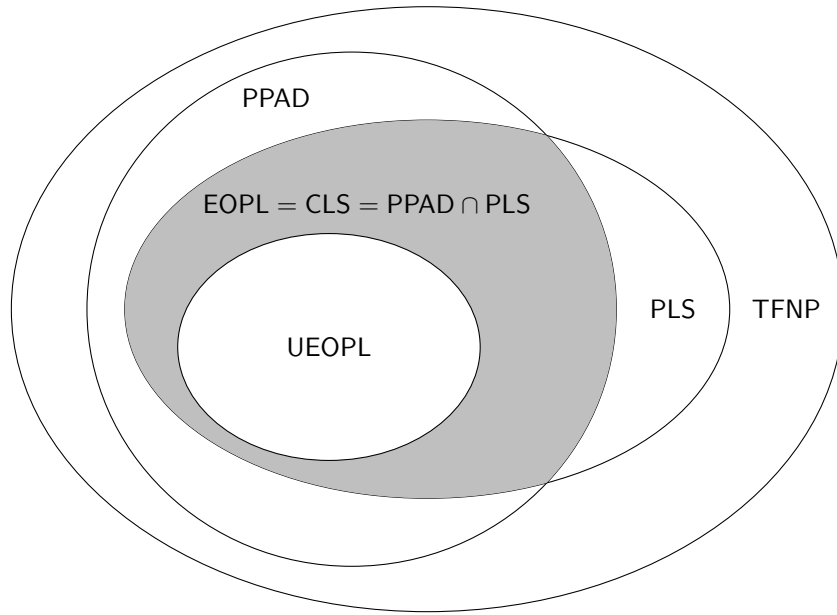
■ **Figure 3** Relation of UEOPL to other search problem complexity classes according to [8].

The refined index assigns to each point a d-tuple containing at index i the number of outgoing edges in dimension i.

▶ **Theorem 3.4.** *([7, Theorem 2.14]) If σ is a unique sink orientation, then r_σ is a bijection.*

■ **Figure 4** Example grid Γ with $\mathcal{M} = \{1, \ldots, 7\}$, $\kappa_1 = \{1, 2\}$, $\kappa_2 = \{3, 4\}$ and $\kappa_3 = \{5, 6, 7\}$. The orientation of the edges is a unique sink orientation. The unique sink is the point (147). The outmap of point (246) is $\sigma(246) = \{5, 7\}$ and its refined index is $r_\sigma(246) = (0, 0, 2)$.

▶ **Definition 3.5.** ([2, Definition 6.1.23]) The search problem GRID-USO is defined as follows: Given a d-dimensional grid, represented implicitly by $\Gamma = (\mathcal{M}, K)$, and a circuit computing an outmap function $\sigma \colon V \to Powerset(\mathcal{M})$, find one of the following:

(GU1) A point $p \in V$ with $\sigma(p) = \emptyset$. The point p is a sink.

(GUV1) A point $p \in V$ with $p \cap \sigma(p) \neq \emptyset$. The point p has a directed edge to itself, thus it is a certificate of σ not being a valid unique sink orientation.

(GUV2) An induced subgrid $\Gamma' = (\mathcal{M}', K')$ with $\sigma'(p) := \sigma(p) \cap \mathcal{M}'$ and two points $p, q \in V'$ with $p \neq q$ and $r_{\sigma'}(p) = r_{\sigma'}(q)$. The points p, q and the subgrid Γ' are a polynomial time verifiable certificate that $r_{\sigma'}$ is not a bijection and thus, σ not a unique sink orientation.

GRID-USO is a total search problem. Under the promise that the outmap σ is a unique sink orientation, the unique sink will be found and returned as solution *(GU1)*. If σ is *not* a unique sink orientation, there exists at least one of the violations *(GUV1)* or *(GUV2)*. An example instance can be seen in Figure 4. Unique sink orientations on grids were introduced by Gärtner et al. [7] as a combinatorial abstraction of linear programming over products of simplices and the generalized linear complementarity problems over P-matrices. There is no polynomial time algorithm known to solve GRID-USO.

4 Grid-USO is in UEOPL

▶ **Theorem 4.1.** *GRID-USO can be reduced via a promise preserving reduction to* UNIQUE FORWARD EOPL*.*

Proof. Given an instance $I = (\Gamma, \sigma)$ of GRID-USO, we construct one instance of UNIQUE FORWARD EOPL $I' = (S, c)$ such that solutions and violations can be mapped accordingly.

Idea of the reduction. We construct a line following algorithm that finds for any given grid its unique sink or a violation, such that each step can be calculated in polynomial time. The vertices of I' are encodings of the states of this line following algorithm. The successor function S calculates the next state. To do so, it is allowed to call the outmap function σ from the GRID-USO instance.

▶ **Lemma 4.2.** *([2, Lemma 6.3.5]) Given an outmap σ on a grid Γ, two disjoint induced subgrids Γ' and Γ'' of Γ whose union is again a valid subgrid, and their respective unique sinks x and y, then* (a) *the unique sink of the grid $(\Gamma' \cup \Gamma'')$ is either x or y or* (b) *σ is not a unique sink orientation and we found a violation of* GRID-USO*.*

The line following algorithm starts at the bottom left point of the grid. It iterates over all directions, in each step looking at the subgrid that is formed by the union of the previous directions (e.g., the subgrid colored in blue in Figure 5) and its sink x. By adding the next lexicographic direction, we add another subgrid (e.g., the yellow subgrid in Figure 5) for which we can find the sink y recursively. The sink of the grid which is the union of the blue and the yellow subgrids is either x or y. None of the other nodes in the subgrids is a sink of the respective subgrid, and thus not a sink of the combined grid. If neither x nor y is a sink, then it can be proven that there is a refined index violation of type *(GUV2)*.
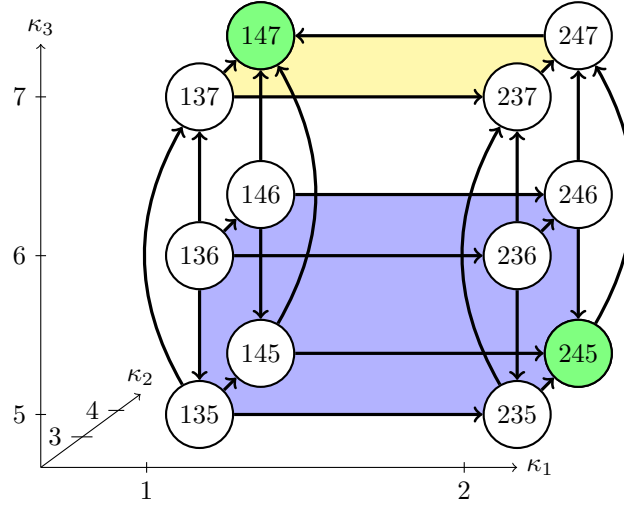
Figure 5 Step of the line-following algorithm: either (245) or (147) is the sink of the whole grid.

Algorithm 1: find_sink($\sigma, \mathcal{M}, \kappa_1, \ldots, \kappa_d$)

---

1 $x := ((\kappa_1)_1, \ldots, (\kappa_d)_1)$;
2 for $i \in \mathcal{M}$ **do**
3 **if** $i \notin \sigma(x)$ // x is also sink in i'th subgrid **then**
4 | continue with next i;
5 Let j be the index such that $i \in \kappa_j$;
6 $\mathcal{M}' := (\mathcal{M} \setminus \kappa_j) \cup \{i\}$;
7 $y := $ find_sink($\sigma, \mathcal{M}', (\kappa_1 \cap \mathcal{M}'), \ldots, (\kappa_d \cap \mathcal{M}')$) // Search recursively for sink in yellow subgrid ;
8 **for** $k \in \kappa_j \wedge k \leq i$ **do**
9   **if** $k \in \sigma(y)$ // If y is not sink in i'th subgrid **then**
10 | return Violation;
11 $x := y$;
12 return x;

Construction of the vertices.   Each node of I' is the bit-encoding of an $(n + 1)$-tuple of points of the grid. Let $V' := (V \cup \{\bot\})^{n+1}$. Its contents encode the state of Algorithm 1. The points stored in the vertices are the unique sinks of the subgrids in which Algorithm 1 searches recursively. The position corresponds to the directions through which the algorithm iterates. Thus, we can identify for each tuple which step of the algorithm it represents. Let the start node be the $(n + 1)$-tuple consisting of the bottom left point of the grid and n many $\bot$'s: $(((\kappa_1)_1, \ldots, (\kappa_d)_1), \bot, \ldots, \bot)$. We define the function isVertex which checks in polynomial time, whether any given $(n + 1)$-tuple is a valid step of Algorithm 1, the representation of a GRID-USO violation or not a proper encoding at all.

Construction of the successor function S**.**   The successor function S checks for the given node whether it encodes a valid step of Algorithm 1 and if so, calculates the next step in polynomial time. Each step, including the steps of the recursive calls, is a separate node on the resulting unique line. The line corresponds to traversing the tree of the call hierarchy.

Let $S\colon V' \to V'$. Given a vertex $v = (p_1, \dots, p_n, p_{n+1})$, let $S(v)$ be:

1. If $\mathtt{isVertex}(v)$ says v is *not* a valid encoding, then set $S(v) := v$.
2. If $\mathtt{isVertex}(v)$ says v is a valid encoding and not a violation:

   a. If the node has the form $v = (\bot, \dots, \bot, p_{n+1})$ it encodes the end of the for-loop in line 2 of Algorithm 1. Thus, the algorithm returns the sink p_{n+1}. Thus, set $S(v) := v$ to indicate the end of the line.
   b. If the node has the form $v = (\bot, \dots, \bot, p_i, p_{i+1}, p_{i+2}, \dots, p_{n+1})$, then p_i is the sink x of the i'th iteration of the for-loop in line 2.

   i. If the check in line 3 is true, we know that x is also the sink of the next bigger subgrid. Thus, set $S(v) := (\bot, \dots, \bot, \bot, p_i, p_{i+2}, \dots, p_{n+1})$.
   ii. If the check in line 3 is false, then we know that x is *not* the sink of the next bigger subgrid. By Lemma 4.2, we must search recursively for the sink of the yellow subgrid. But we want to remember x, so that we can identify a violation if one exists. Thus, set $S(v) := (s, \bot, \dots, \bot, p_i, p_{i+1}, \dots, p_{n+1})$, where s is the start point of the subgrid in the recursive call.

Construction of the cost function c. Let $\omega = n + 2$ and $h\colon V' \times \{1, \dots, n\} \times \{1, \dots, d\} \to \{0, \dots, \omega - 1\}$ a help function with

$$h(v, i, j) := \begin{cases} 0 & \text{if } p_i = \bot, \\ \omega - 1 & \text{if } \sigma(p_i) \cap \{1, \dots, i\} = \emptyset, \text{i.e., step } \textit{(2.b.i)} \text{ holds}, \\ (p_i)_j & \text{otherwise.} \end{cases} \tag{1}$$

Because the grid works with ordered sets, lexicographically bigger points have a higher value in h. Also, if two points are sink of their corresponding subgrid, they have the same value in h. The help values of each point and every dimension are scaled and summed up. The cost of a node then is the sum of these scaled values, where each summand is scaled again to enforce that the value for a point at position i which is not $\bot$ is always higher than the sum of all values of points with indices smaller than i. Thus, we always give steps later on in the algorithm higher cost.

$$c(v) := \left(\sum_{i=1}^{n} \left(\omega^{i-1} \cdot \sum_{j=1}^{d} \omega^j h(v, i, j) \right) \right) + \begin{cases} 0 & \text{if } p_{n+1} = \bot, \\ \omega^{nd+1} & \text{if } p_{n+1} \neq \bot. \end{cases} \tag{2}$$

Correctness The successor function and the cost function can be constructed in polynomial time. It can be proven that every solution of type *(GU1)* is only mapped to solutions of type *(UF1)*. Every violation of the created Unique Forward EOPL instance can be mapped back to a violation of the Grid-USO instance. Therefore this reduction is promise preserving. It follows that Grid-USO is in UEOPL and the promise version of Grid-USO is in PromiseUEOPL. The full proof can be found in [2, Proof of Theorem 6.3.1]. ◀

─── **References** ───

1 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009.
2 Michaela Borzechowski. The complexity class unique end of potential line. Master's thesis, 2021. URL: `https://www.mi.fu-berlin.de/inf/groups/ag-ti/theses/download/Borzechowski21.pdf`.

3 Man-Kwun Chiu, Aruni Choudhary, and Wolfgang Mulzer. Computational complexity of the α-ham-sandwich problem. *arXiv preprint arXiv:2003.09266*, 2020.

4 John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *CoRR*, abs/1811.03841, 2018. URL: `http://arxiv.org/abs/1811.03841`.

5 John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *Journal of Computer and System Sciences*, 114:1 – 35, 2020. `doi:https://doi.org/10.1016/j.jcss.2020.05.007`.

6 Bernd Gärtner, Thomas Dueholm Hansen, Pavel Hubáček, Karel Král, Hagar Mosaad, and Veronika Slívová. ARRIVAL: next stop in CLS. *CoRR*, abs/1802.07702, 2018. URL: `http://arxiv.org/abs/1802.07702`.

7 Bernd Gärtner, D Walter Jr, Leo Rüst, et al. Unique sink orientations of grids. *Algorithmica*, 51(2):200–235, 2008. `doi:https://doi.org/10.1007/s00453-007-9090-x`.

8 Mika Göös, Alexandros Hollender, Siddhartha Jain, Gilbert Maystre, William Pires, Robert Robere, and Ran Tao. Further collapses in tfnp, 2022. `arXiv:2202.07761`.

9 Nimrod Megiddo and Christos H Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991. `doi:https://doi.org/10.1016/0304-3975(91)90200-L`.