Realizability in Matoušek Unique Sink Orientations: Characterization and Complexity Gap

Bernd Gärtner Simon Weber Joel Widmer

Submitted: Feb 5, 2024; Accepted: Feb 27, 2025; Published: Apr 11, 2025

Abstract

Various algebraic and geometric problems reduce to the sink-finding problem in *unique sink orientations (USOs)*, which are orientations of the hypercube graph that have a unique sink in every subcube. A USO is called *realizable* if it can arise from applying one of these reductions. We study how realizability influences the query complexity of the sink-finding problem. To this end, we consider a specific subclass of USOs, the so-called *Matoušek USOs*. The Matoušek USOs are a family of USOs which are a translation of the LP-type problems used by Matoušek to show that the Sharir-Welzl algorithm for LP-type problems may require at least subexponential time [Matoušek, 1994]. Gärtner showed that the *Random Facet* algorithm for USO sink-finding requires at least subexponentially many vertex evaluation queries on Matoušek USOs, but at most quadratically many queries on realizable Matoušek USOs [Gärtner, 2002]. However, Gärtner did not fully characterize this realizable subset. In this paper, we fully characterize the realizable subset of the Matoušek-type USOs (the USOs isomorphic to a Matoušek USO) and also provide concrete realizations using instances of the P-Matrix Linear Complementarity Problem, basing our work on the so-called *cyclic-P-matroids* studied by Fukuda, Klaus, and Miyata. We further extend the results of Matoušek and Gärtner for the Random Facet algorithm to the query complexity of the sink-finding problem itself: we show that sink-finding is strictly easier on realizable Matoušek-type USOs than on all Matoušek-type USOs. We show that in the realizable case $O(\log^2 n)$ queries suffice, while in general exactly n queries are needed.

Mathematics Subject Classifications: 68R05, 05C75, 52C40

Acknowledgments. This research was supported by the Swiss National Science Foundation under project number 204320. An extended abstract [27] covering parts of this paper has been previously published in Lecture Notes in Computer Science, Volume 14079, pages 704–718, 2023, by Springer Nature. The other parts have only appeared on arXiv [26].

Department of Computer Science, ETH Zürich (`gaertner@inf.ethz.ch`, `simon.weber@inf.ethz.ch`, `joelwidmerch@gmail.com`).

1 Introduction

A Unique Sink Orientation (USO) is an orientation of the hypercube graph with the property that each non-empty face has a unique sink. The most studied algorithmic problem related to USOs is that of finding the global sink: An algorithm has access to a *vertex evaluation oracle*, which can be queried with a vertex and returns the orientation of all incident edges. The task is to determine the unique sink of the USO using as few such vertex evaluation queries as possible.

Progress on this problem has stalled for a long time. Since Szábo and Welzl introduced USOs in 2001 [24], their deterministic and randomized algorithms — both requiring an exponential number of queries in terms of the hypercube dimension — are still the best known in the general case. Only for special cases, such as for acyclic USOs, better algorithms are known [9].

Realizability. The study of USOs and the sink-finding problem was originally motivated by a reduction of the P-Matrix Linear Complementarity Problem (P-LCP) to sink-finding in USOs [23]. Many widely studied optimization problems have since been shown to be reducible either to the P-LCP or to sink-finding in USOs directly, the most notable example being Linear Programming (LP) [12], but also Convex Quadratic Programming [19] or the Smallest Enclosing Ball problem [13]. It has also been shown that various games on graphs reduce to the P-LCP, such as simple stochastic games, mean payoff games, and parity games [11], the unresolved complexity statuses of which have sparked considerable interest. To make progress on this wide array of problems, one would not need to find an algorithm which can find the sink quickly in *all* USOs, but only in the USOs which can arise from these reductions. In fact all these problems reduce to sink-finding in USOs that can also be generated by the reduction from P-LCP. We call USOs that can be obtained from this reduction *realizable*. In the literature, the realizable USOs are also referred to as *P-USOs* [6], *PLCP-orientations* [16], or *P-cubes* [18].

The number of n-dimensional realizable USOs is much smaller than the total number of USOs, namely $2^{\Theta(n^3)}$ in contrast to $2^{\Theta(2^n \log n)}$ [6]. Furthermore, a simple combinatorial property, the Holt-Klee condition, is known to hold for all realizable USOs [10, 14]: In a realizable USO, there must be n vertex-disjoint directed paths from the source to the sink. Thus, one can for example conclude that the USO shown in Figure 1 is not realizable. The best-known lower bound for the query complexity of the sink-finding problem (for deterministic algorithms) is $\Omega(n^2/\log n)$ [20]. This lower bound however relies on USOs which fail the Holt-Klee condition and are therefore not realizable [2]. For sink-finding on realizable USOs, only a lower bound of $\Omega(n)$ is known [2].

We thus see differences between general and realizable USOs in their number, structure, as well as the current knowledge of lower bounds. These three facts indicate that it may be possible to algorithmically exploit the features of realizable USOs to beat the algorithms of Szábo and Welzl on this important class of USOs. For P-LCP and certain cases of LP, the fastest deterministic combinatorial algorithms are already today based on sink-finding in USOs [5, 12]. Improved sink-finding algorithms for realizable USOs would directly translate to advances in P-LCP and LP algorithms. In particular, a sink-finding

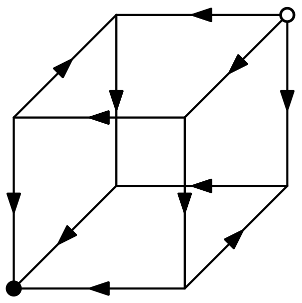


Figure 1: A three-dimensional USO that is not realizable: there are no three vertex-disjoint paths from the source (top right) to the sink (bottom left). This USO thus does not satisfy the Holt-Klee condition.

algorithm using polynomially many vertex evaluations on any realizable USO would imply the existence of a strongly polynomial-time algorithm for LP, answering a question from Smale's list of *mathematical problems for the next century* [21].

Matoušek(-type) USOs. In 1994, Matoušek introduced a family of LP-type problems (a combinatorial generalization of linear programs) to show a superpolynomial lower bound on the runtime of the Sharir-Welzl algorithm [17]. This result was later translated by Gärtner into the framework of USOs, where the *Matoušek USOs* provide a superpolynomial lower bound on the query complexity of the *Random Facet* algorithm [9]. In the same paper it was also shown that the sink of all Matoušek USOs that fulfill the Holt-Klee property (thus including all realizable ones) is found by the Random Facet algorithm in a quadratic number of queries. Therefore, Random Facet is strictly (and substantially) faster on the realizable Matoušek USOs than on all Matoušek USOs. In this paper we aim to provide a similar result for the query complexity of the problem itself, instead of a concrete algorithm.

The Matoušek USOs all have the sink at the same vertex. This does not pose a problem when analyzing a fixed algorithm which does not exploit this fact (e.g., Random Facet), but it does not allow us to derive algorithm-independent lower bounds. To circumvent this issue, we consider the class of *Matoušek-type USOs*, which simply contains all orientations isomorphic to classical Matoušek USOs.

To study the query complexity of sink-finding on (realizable) Matoušek-type USOs, we first prove that the Holt-Klee condition is not only necessary but also sufficient for realizability on Matoušek-type USOs. We thus provide a full characterization of the realizable USOs among the Matoušek-type USOs. We achieve this by working with *P-matroids*, another combinatorial abstraction of the P-LCP. Here, we show a connection between realizable Matoušek-type USOs and simple extensions of cyclic-P-matroids, a subclass of P-matroids first defined by Fukuda, Klaus, and Miyata [7, 15].

1.1 Results

We first show that the Holt-Klee condition is both necessary and sufficient for realizability in Matoušek-type USOs. In doing such, we also show that the set of realizable Matoušek-type USOs is exactly the set of USOs associated with simple extensions of cyclic-P-matroids.

Theorem 1. *The following families of USOs are all equivalent:*

- *The USOs associated with simple extensions of cyclic-P-matroids*
- *The realizable Matoušek-type USOs*
- *The Matoušek-type USOs fulfilling the Holt-Klee condition*

We then show a query complexity gap between sink-finding on the realizable and sink-finding on all Matoušek-type USOs. We achieve this by proving the following two main theorems:

Theorem 2. *For every deterministic sink-finding algorithm $\mathcal{A}$ and any $n \geqslant 2$, there exists an n-dimensional Matoušek-type USO on which $\mathcal{A}$ requires at least n vertex evaluations to find the sink.*

Theorem 3. *The sink of any n-dimensional* realizable *Matoušek-type USO can be found deterministically using $O(\log^2 n)$ vertex evaluations in the worst case.*

In addition, we show that the result about general Matoušek-type USOs is tight. For the realizable case, we provide a simple lower bound of $\Omega(\log n)$ vertex evaluations.

1.2 Discussion

The Matoušek-type USOs form the first natural family of USOs for which we have a full characterization of realizability, as well as the first found to admit a query complexity gap. We hope that this result motivates further research into structural and algorithmic aspects of realizability for larger, more relevant classes of USOs.

Note that an artificial class of USOs exhibiting such a complexity gap could easily be constructed by combining a set R of easy-to-solve realizable USOs with a set N of difficult-to-solve non-realizable USOs. For R, one can take any set of realizable USOs which all have the same vertex as their sink. An algorithm to find the sink of USOs in R could then always output this vertex without needing to perform any vertex evaluations. For the set N, one can take the set of USOs constructed in the lower bound of Schurr and Szábo [20], and change each USO such that it becomes non-realizable. This can be achieved without destroying the lower bound. The resulting class $R \cup N$ then also exhibits a complexity gap.

The Matoušek-type USOs are not such an artificially constructed class of USOs. First off, they are well-studied due to their significance in proving the lower bound for the

Random Facet algorithm [17, 9]. Second, they can be considered a natural choice for proving unconditional lower bounds for realizable USOs: The only known unconditional lower bound for randomized algorithms on general USOs uses decomposable USOs [20], and the realizable Matoušek-type USOs are the only known class of realizable decomposable USOs.

Even on a natural USO class, a complexity gap could be trivial to obtain, for example if the class contains no (or only very few) realizable USOs. This is also *not* the case for the Matoušek-type USOs, as there are $2^{\Theta(n \log n)}$ realizable n-dimensional Matoušek-type USOs, while the overall number of Matoušek-type USOs is $2^{\Theta(n^2)}$. This is a much larger realizable fraction than one observes on the set of all USOs.

There is also an interesting connection between Matoušek-type USOs and *D-cubes*. The D-cubes are a subset of realizable USOs, obtained by the reduction to sink-finding from P-LCP instances where the P-matrix M is *symmetric*. They also include the USOs arising from the reduction of LP to sink-finding. Gao, Gärtner and Lamperski recently discovered that in a D-cube, the *L-graphs* at all vertices have to be acyclic [8]. The L-graphs are graphs that describe the local structure of the USO around a single vertex, and can be seen as a local version of the global *dimension influence graph* encoding the structure of a Matoušek-type USO, which we introduce in this paper. In fact, a USO is a Matoušek-type USO if and only if the L-graphs are the same at every vertex. The techniques developed in this work to find the sink in the more rigid Matoušek-type USOs may be useful in developing algorithms for D-cubes, but this would first require a better understanding of the possible L-graphs in D-cubes. While all Matoušek-type USOs fulfill the necessary condition for being D-cubes, it remains open whether the realizable Matoušek-type USOs are in fact D-cubes.

1.3 Paper Overview

In Section 2, we lay out the necessary notations and definitions. In Section 3, we analyze simple extensions of cyclic-P-matroids and provide the full characterization for the realizable Matoušek-type USOs. In Section 4, we then prove the query complexity gap between sink-finding in all Matoušek-type USOs and their realizable subset. Finally, we discuss remaining open questions in Section 5.

2 Preliminaries

We begin with some basic notation. All vectors and matrices in this paper are defined over the field $\mathbb{Z}_2$. We write $\oplus$ for bit-wise addition ("xor") in $\mathbb{Z}_2$. By $\mathbf{0}$ (or $\mathbf{1}$) we denote the all-zero (or all-ones) n-dimensional vector. By e_i we denote the i-th standard basis vector. I_n denotes the $n \times n$ identity matrix. For a natural number x, we write $Bin(x)_i \in \{0, 1\}$ for the i-th least significant bit of the binary representation of x, such that $\sum_{i=0}^{\infty} Bin(x)_i \cdot 2^i = x$.

2.1 Hypercubes, Orientations, and USOs

The n-dimensional hypercube is an undirected graph (V, E) consisting of the vertex set $V = \{0,1\}^n$, where two vertices are connected by an edge if they differ in exactly one coordinate. An edge $\{v, v \oplus e_i\}$ is called an *i-edge*.

A subcube of a hypercube is called a *face*. A face F_f is specified by a vector $f \in \{0,1,*\}^n$, with $v \in F_f$ if and only if for all $i \in [n]$, either $f_i = *$ or $f_i = v_i$. The number of "$*$" entries in f denotes the *dimension* of F_f, and we say F_f is *spanned by* the dimensions i for which $f_i = *$. A face of dimension $n-1$ is also called a *facet*. The facet F_f with $f_d = 1$ is called the *upper d-facet*, and the opposite facet $F_{f'}$ with $f'_d = 0$ is called the *lower d-facet*.

An orientation of the hypercube assigns a direction to each of the $n2^{n-1}$ edges.

Definition 4 (Hypercube Orientation)**.** An *orientation o* of the n-dimensional hypercube is described by a function $o : \{0,1\}^n \to \{0,1\}^n$ assigning each vertex its *outmap*: An edge $\{v, v \oplus e_i\}$ is directed away from v (outgoing) if $o(v)_i = 1$, otherwise it is directed towards v (incoming). To ensure consistent orientation of all edges, o must fulfill $o(v)_i \neq o(v \oplus e_i)_i$ for all $v \in V$ and $i \in [n]$.

We use the same notation to refer to orientations as directed graphs as well as their outmap functions.

Definition 5 (Unique Sink Orientation)**.** A *unique sink orientation (USO)* is an orientation of the hypercube, such that for each non-empty face F, the subgraph induced by F has a unique sink, i.e., a unique vertex with no outgoing edges.

Szábo and Welzl [24] provide the following characterization of USOs:

Lemma 6 (Szábo-Welzl condition [24])**.** *An orientation o is a USO if and only if*

$$\forall v, w \in \{0,1\}^n : \big(v \oplus w\big) \cap \big(o(v) \oplus o(w)\big) \neq \mathbf{0}.$$

In other words, for any two vertices, their outmaps must differ within the subcube they span. Equivalently, this means that o must be a bijection, even when the domain is constrained to any face F, and the codomain is restricted to the dimensions spanning F.

2.2 Matoušek(-type) USOs

A Matoušek USO, as defined by Gärtner [9], is an orientation o characterized by an invertible, upper-triangular matrix $A \in \{0,1\}^{n \times n}$ (thus all diagonal entries of A are 1). The matrix defines the orientation $o(v) = Av$. Since each principal submatrix of A is also upper-triangular with all diagonal entries equal to 1, each principal submatrix must also be invertible. This implies that there must be a unique sink in each face of the hypercube, and thus o is a USO. In particular, the global sink lies at the vertex $\mathbf{0}$.

This commonality among Matoušek USOs would pose a problem in Section 4 where we aim to derive lower bounds for the sink-finding problem on Matoušek USOs; an optimal algorithm can "find" the sink immediately. To eliminate this problem, we define

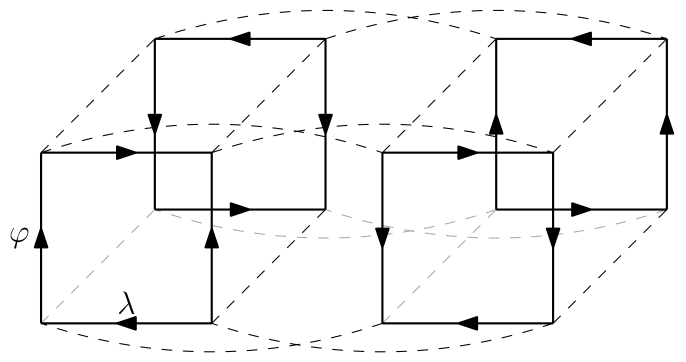


Figure 2: All 2-faces spanned by λ and φ in this 4-dimensional Matoušek-type USO have the same structure: φ *influences* λ, but not the other way around.

the Matoušek-type USOs, which are all orientations *isomorphic* to a Matoušek USO. Isomorphisms on the hypercube allow for mirroring of any subset of dimensions, and for relabeling the dimensions. This motivates the following definition:

Definition 7 (Matoušek-type USO)**.** A *Matoušek-type USO* is an orientation o, with

$$\forall v \in \{0,1\}^n: \quad o(v) = M(v \oplus s), \text{ where } M := PAP^T$$

for some permutation matrix P, an invertible, upper-triangular matrix $A \in \{0,1\}^{n\times n}$, and the desired location of the sink $s \in \{0,1\}^n$.

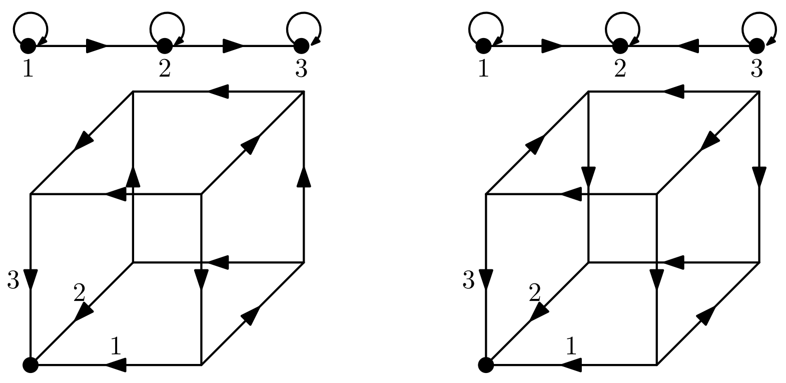
We can view the matrix M as the adjacency matrix of a directed graph $G = ([n], E_M)$, where $(i,j) \in E_M$ if $M_{j,i} = 1$. As A is invertible and upper-triangular, and as M is equal to A with rows and columns permuted in the same way, G is the *reflexive closure* of an acyclic graph, i.e., an acyclic graph with self-loops added to every vertex. We call the graph G the *dimension influence graph* of the Matoušek-type USO. The name is motivated by the following observation.

Observation 8. *Let $\lambda, \varphi \in [n]$ be two distinct dimensions of a Matoušek-type USO o. For any vertex $v \in \{0,1\}^n$, it holds that*

$$o(v)_\lambda \neq o(v \oplus e_\varphi)_\lambda \iff M_{\lambda,\varphi} = 1 \iff (\varphi, \lambda) \in E_M.$$

Intuitively, this means that in a Matoušek-type USO, any 2-dimensional face spanned by the same two dimensions λ and φ has the same structure: Travelling along a φ-edge either always changes the direction of the adjacent λ-edge, or never (see Figure 2). If it always changes, we say that φ *influences* λ, and there is an edge from φ to λ in the dimension influence graph. See Figure 3 for two example dimension influence graphs, their corresponding Matoušek-type USOs, and their adjacency matrices.

A USO in which all i-edges are oriented the same way is called *combed in dimension i*, or simply *combed.* As the dimension influence graph G is acyclic (apart from the loops), there must be a sink in every induced subgraph of G. Thus, in each face there is a dimension i which is not influenced by any other dimension, and the face must be combed in dimension i. A USO in which every face is combed, such as the Matoušek-type USOs, is also called *decomposable.*



$$\begin{pmatrix}1 & 0 & 0\\ 1 & 1 & 0\\ 0 & 1 & 1\end{pmatrix} \qquad \begin{pmatrix}1 & 0 & 0\\ 1 & 1 & 1\\ 0 & 0 & 1\end{pmatrix}$$

Figure 3: Top: Two dimension influence graphs. Middle: The corresponding Matoušek-type USOs with the sink at the bottom left vertex. Bottom: The adjacency matrices of the dimension influence graphs.

2.3 Complementarity Problems and USOs

2.3.1 P-LCP

The *P-matrix linear complementarity problem* (P-LCP) is crucial to the history of unique sink orientations [23]. An instance (M, q) of the more general linear complementarity problem (LCP) is given by a matrix $M \in \mathbb{R}^{n\times n}$ and a vector $q \in \mathbb{R}^n$. The task is to find vectors $w, z \in \mathbb{R}^n$ fulfilling both

1. Feasibility: $w - Mz = q$, and for all $i \in [n]$, $w_i = 0$ or $z_i = 0$.
2. Non-negativity: $w, z \geqslant \mathbf{0}$.

In general, an instance (M, q) is neither guaranteed to have a unique solution, nor that a solution exists at all. However, if M is guaranteed to be a *P-matrix*, we call the problem the P-LCP, and the every instance has a unique solution.

Definition 9. A matrix $M \in \mathbb{R}^{n\times n}$ is a *P-matrix* if all its principal minors, i.e., the determinants of all its principal submatrices, are strictly positive.

We call a pair of vectors (w, z) fulfilling the feasibility condition a *candidate* solution. Since M is a P-matrix and thus all principal minors are non-zero, for each choice of $B \in \{0, 1\}^n$ there exists a unique candidate solution (w, z) such that $B_i = 1$ implies $w_i = 0$ and $B_i = 0$ implies $z_i = 0$. If such a candidate solution fulfills $w_i = z_i = 0$ for some i, we say that q is *degenerate for M at B*. If q is not degenerate for M at any B, we say that the instance $LCP(M, q)$ is *non-degenerate.*

A non-degenerate P-LCP instance (M, q) realizes a USO o in the following way: For each vertex $B \in \{0, 1\}^n$, $o(B)_i = 1$ if and only if $w_i, z_i \leqslant 0$ in the unique candidate solution (w, z) corresponding to B.

A USO o is called *realizable*, if there exists a non-degenerate P-LCP instance (M, q) realizing o. No combinatorial characterization of realizability is known, however, Holt and Klee [14] proved the following necessary condition:

Lemma 10 (Holt-Klee condition [10, 14]). *Let o be a realizable n-dimensional USO. Then there exist n interior-vertex-disjoint directed paths from the unique source of o to the unique sink of o.*

Furthermore, it is known that if o is realizable, all USOs isomorphic to o as well as all restrictions of o to some subcube are also realizable [16, 23].

2.3.2 P-OMCP

An *oriented matroid* can describe the combinatorial structure of many different objects, such as vector arrangements, hyperplane arrangements, or directed graphs. Furthermore, an oriented matroid can itself be described using a number of equivalent, "cryptomorphic" axiomatizations, such as vector, circuit, or chirotope axioms. In this work we focus on oriented matroids specified according to circuit axioms and describing vector arrangements. For a more complete overview over oriented matroids, we refer the reader to the textbook of Björner, Las Vergnas, Sturmfels, White, and Ziegler [1].

For a *ground set* E, a *signed set* on E is a pair $S = (S^+, S^-)$ with $S^+ \cap S^- = \emptyset$ and $S^+, S^- \subseteq E$. We call the union $\underline{S} = S^+ \cup S^-$ the *support* of S. We write $-S$ for the signed set $-S := (S^-, S^+)$. Given a signed set S and an element $e \in E$, we write $S_e \in \{-, 0, +\}$ for the sign of e in S.

An oriented matroid in circuit representation is a pair $\mathcal{M} = (E, \mathcal{C})$ where $\mathcal{C}$ is a collection of signed sets on E, the so-called *circuits* of $\mathcal{M}$, fulfilling the following circuit axioms:

(C0) $(\emptyset, \emptyset) \notin \mathcal{C}$

(C1) Symmetry: $\forall C \in \mathcal{C} : -C \in \mathcal{C}$

(C2) Incomparability: $\forall X, Y \in \mathcal{C} : \underline{X} \subseteq \underline{Y} \Rightarrow (X = Y \lor X = -Y)$

(C3) Weak elimination: $\forall X, Y \in \mathcal{C}$ with $X \neq -Y$, and $\forall e \in X^+ \cap Y^-$, there exists a $Z \in \mathcal{C}$, such that

$$Z^+ \subseteq (X^+ \cup Y^+) \backslash \{e\} \text{ and}$$
$$Z^- \subseteq (X^- \cup Y^-) \backslash \{e\}.$$

A *basis* of $\mathcal{M}$ is an inclusion-maximal subset of E that does not contain any circuit. All bases of an oriented matroid have the same size, called the *rank* of the matroid. For any basis B and any element $e \in E \setminus B$, there is a unique *fundamental circuit* $C(B, q)$ with $\underline{C(B, q)} = B \cup \{q\}$ and $q \in C(B, q)^+$.

A vector configuration, here represented by a matrix $V \in \mathbb{R}^{d\times n}$, gives rise to an oriented matroid $\mathcal{M}$ with $E = [n]$ and $\mathcal{C}$ being the collection of inclusion-minimal linear dependencies of columns of V, i.e.,

$$\mathcal{C} = \left\{ \operatorname{sign} x : \Big(Vx = 0 \wedge \nexists x' : \big(\underline{\operatorname{sign} x'} \subset \underline{\operatorname{sign} x} \wedge Vx' = 0\big) \Big) \right\},$$

where $\operatorname{sign} x$ is the signed set $\big(\{i : x_i > 0\}, \{i : x_i < 0\}\big)$. We say that V *realizes* $\mathcal{M}$, and $\mathcal{M}$ is *realizable*. Some oriented matroids are not realizable, i.e., there exist no vector configurations realizing them.

While general oriented matroids provide a combinatorial abstraction of general vector configurations, *P-matroids* abstract the special vector configurations corresponding to P-matrices augmented with the standard identity vectors. A *P-matroid* is an oriented matroid $\mathcal{M} = (E_{2n}, \mathcal{C})$ with ground set $E_{2n} = [2n]$. E_{2n} is split in two parts, $S = [n]$ and $T = [2n]\backslash[n]$, S being a basis of $\mathcal{M}$. Elements i and $i+n$ are called *complementary*, together they form a *complementary pair*. For any i, its *complementary element* is denoted by $\bar{i}$. Any set of n elements not containing a complementary pair is a basis. $\mathcal{M}$ is a P-matroid if there is no *almost-complementary sign-reversing circuit*, i.e., no circuit $C \in \mathcal{C}$ such that $\underline{C}$ contains exactly one complementary pair $(i, i+n)$ and i and $i+n$ have different signs in C [25].

An *extension* $\widehat{\mathcal{M}} = (\widehat{E}, \widehat{\mathcal{C}})$ of a matroid $\mathcal{M} = (E, \mathcal{C})$ is an oriented matroid with ground set $\widehat{E} = E \cup \{q\}$, such that the *deletion minor* $\widehat{\mathcal{M}}\backslash q = (E, \{X : X \in \widehat{\mathcal{C}} \text{ and } X_q = 0\})$ is equal to $\mathcal{M}$.

For a P-matrix $M \in \mathbb{R}^{n\times n}$, the *associated P-matroid* is the oriented matroid realized by $\begin{bmatrix} I_n & -M \end{bmatrix}$. For a P-LCP instance (M, q), the *associated extension of a P-matroid* is realized by $\begin{bmatrix} I_n & -M & -q \end{bmatrix}$.

Given an extension of a P-matroid $\widehat{M} = (\widehat{E_{2n}}, \widehat{\mathcal{C}})$, one can obtain the associated USO u using the following procedure [15]: For each vertex $v \in \{0,1\}^n$, determine the fundamental circuit[1]

$$C_v := C(\{i + v_i \cdot n : i \in [n]\},\ q)$$

and set $u(v)_i = 1$ if $i \in C_v^-$ or $i + n \in C_v^-$. Note that a P-LCP instance (M, q) realizes the same USO as its associated P-matroid extension.

While a P-matroid extension $\widehat{M}$ can be realized by vector configurations not of the form $\begin{bmatrix} I_n & -M & -q \end{bmatrix}$, if one is given a $(2n+1)\times n$-matrix V realizing $\widehat{M}$, a P-LCP instance with the same associated P-matroid extension can be found using

$$(M, q) := (-V_S^{-1} V_T, -V_S^{-1} V_{2n+1}). \tag{1}$$

3 Characterizing the Realizable Matoušek-type USOs

In his proof that Random Facet only uses a quadratic number of vertex evaluations, Gärtner [9] showed a necessary condition on the dimension influence graphs of realizable Matoušek-type USOs:

[1] $\widehat{\mathcal{M}}$ needs to be non-degenerate in the sense that for all v, we must have $|\underline{C_v}| = n + 1$.

Lemma 11 ([9]). *If the dimension influence graph of a Matoušek-type USO contains any of the two graphs G_1 or G_2 in Figure 3 as an induced subgraph, the USO is not realizable.*

Proof. Any 3-dimensional face spanned by the dimensions inducing one of these subgraphs is isomorphic to either of the USOs shown in Figure 3. Both of these USOs do not contain three vertex-disjoint paths from their source to their sink, and are therefore not realizable by Lemma 10. As all faces and all isomorphic copies of a realizable USO are realizable, the whole Matoušek-type USO cannot be realizable either. □

In this section, we show that this condition is not only necessary for realizability of a Matoušek-type USO, but also sufficient. To do this, we analyze the USOs associated with extensions of the cyclic-P-matroids defined by Fukuda, Klaus, and Miyata [7].

3.1 Cyclic-P-Matroids

The *alternating matroid* $\mathcal{A}^{n,r}$ is an oriented matroid of rank r and on ground set $E_n = [n]$. It is realized by n sequential points on the moment curve, i.e., by the matrix

$$V \in \mathbb{R}^{r\times n} := \begin{pmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & \vdots & & \vdots \\ x_1^{r-1} & x_2^{r-1} & \dots & x_n^{r-1} \end{pmatrix}, \text{ for } x_1 < x_2 < \dots < x_n.$$

The alternating matroids are well-studied and their structure is well-understood. For our purposes here, it is only important to know that they are *uniform* (an oriented matroid $(E, \mathcal{C})$ of rank r is uniform if all subsets of E of size r are bases) and that the signs of the non-zero elements in each circuit alternate along the natural order from 1 to n [4].

A *cyclic-P-matroid* is a P-matroid $\mathcal{M}$ that is *reorientation equivalent* to the alternating matroid $\mathcal{A}^{2n,n}$ [15]. Reorientation equivalence means that there exists a set $F \subseteq E_{2n}$ and a permutation π of E_{2n} such that $\mathcal{M} = {}_{-F}(\pi^{-1} \cdot \mathcal{A}^{2n,n})$. This means that $\mathcal{M}$ can be obtained by relabeling the elements of E_{2n} according to π^{-1} in all circuits of $\mathcal{A}^{2n,n}$, and then flipping the sign of all elements in F in all resulting circuits.

As $\mathcal{A}^{2n,n}$ (and therefore also $\mathcal{M}$) is uniform, any set $S \subset E_{2n}$ of size n is a base of $\mathcal{M}$, and thus any set $S \subset E_{2n}$ of size $n+1$ is the support of some circuit. The signs of this circuit can be read off by ordering S according to π, giving the elements alternating signs, and finally flipping the sign of all elements that are in F. This process can be seen in Figure 4. Note that there are always two circuits for each S', each being the negation of the other.

Fukuda, Klaus, and Miyata [7, 15] characterized for which choices of π and F this leads to $\mathcal{M}$ being a P-matroid. Due to a small mistake, their characterization did not exclude some choices for which $\mathcal{M}$ is not a P-matroid. We therefore provide a corrected version of [7, Theorem 4.2]:

Theorem 12. *An oriented matroid $\mathcal{M}$ on E_{2n}, where $\pi \cdot ({}_{-F}\mathcal{M}) = \mathcal{A}^{2n,n}$ for some permutation π of E_{2n} and $F \subseteq E_{2n}$, is a P-matroid if and only if:*

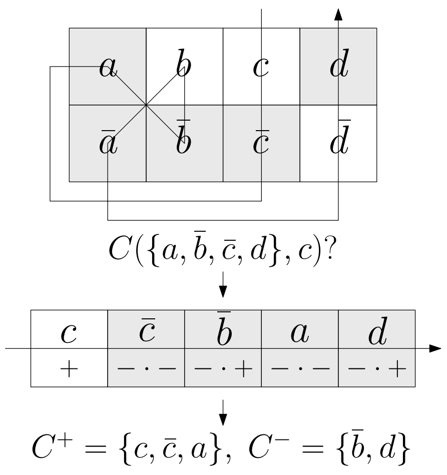


Figure 4: Reading off the signs of a fundamental circuit in a cyclic-P-matroid. The arrow indicates the order of the elements in π. The shaded elements are those in F. Among the two circuits with support $\{a, \bar{b}, c, \bar{c}, d\}$, the one for which c is positive is the desired fundamental circuit.

- *π is such that for all $e \in E_{2n}$ with $\pi(e) < \pi(\overline{e})$, and all $f \in E_{2n}$,*

$$\pi(f) \in [\pi(e), \pi(\overline{e})] \iff \pi(\overline{f}) \in [\pi(e), \pi(\overline{e})].$$

- *and F is such that for every $e \in E_{2n}$, exactly one of e and $\overline{e}$ is in F if*

$$\frac{|\pi(e) - \pi(\overline{e})| - 1}{2}$$

is even, and both or none in F otherwise.

In Theorem 4.2 of [7], the condition on π was such that the intervals $[\pi(e), \pi(\overline{e})]$ were required to contain an even number of elements, but not necessarily both or none of $\pi(f)$ and $\pi(\overline{f})$ for every f.

Proof. We will examine the almost-complementary circuits to prove the two directions individually. Recall that an oriented matroid is a P-matroid if and only if it contains no sign-reversing almost-complementary circuit.

We first show that the conditions are sufficient, and thus assume that π and F fulfill the conditions in Theorem 12. Let $\underline{C}$ be the support of some almost-complementary circuit C, and let e be the element for which both $e, \overline{e} \in \underline{C}$. As C is almost-complementary, the number of elements of $\underline{C}$ between e and $\overline{e}$ according to π must be $\frac{|\pi(e)-\pi(\overline{e})|-1}{2}$, as for each $f \in E_{2n}$, exactly one of f and $\overline{f}$ is contained in $\underline{C}$. The signs of e and $\overline{e}$ in C must therefore be the same, as F flips the sign of exactly one of them if and only if the parity of their positions in $\underline{C}$ according to π is different. As this holds for all almost-complementary circuits, $\mathcal{C}$ contains no almost-complementary sign-reversing circuit, and $\mathcal{M}$ must be a P-matroid.

Next, we show that the conditions are necessary, and thus assume that π and F do not fulfill the conditions in Theorem 12. Then either π does not fulfill the first condition, or F does not fulfill the second. If the condition on π is not fulfilled, then there must be elements e, f such that the interval $[\pi(e), \pi(\overline{e})]$ contains exactly one of $\pi(f)$ and $\pi(\overline{f})$. Let D be a maximal complementary set $D \subseteq E_{2n} \setminus \{e, \overline{e}, f, \overline{f}\}$ and compare the fundamental circuits $C_1 := C(D \cup \{\overline{e}, f\}, e)$ and $C_2 := C(D \cup \{\overline{e}, \overline{f}\}, e)$. The sign of e must be the same in C_1 and C_2, while the sign of $\overline{e}$ must be different in both. We conclude that at least one of C_1 and C_2 must be an almost-complementary sign-reversing circuit.
If the condition on π is fulfilled, the condition on F must be violated for some e. For every maximal complementary set $D \subset E_{2n} \setminus \{e, \overline{e}\}$, the fundamental circuit $C(D \cup \{e\}, \overline{e})$ must be an almost-complementary sign-reversing circuit, by the same arguments as in the proof of the "if" direction.
We conclude that $\mathcal{C}$ must contain an almost-complementary sign-reversing circuit, therefore $\mathcal{M}$ is not a P-matroid. □

With the conditions laid out here, the complementary pairs of E_{2n} permuted by π form a family of properly nested intervals. In this view, we can create a directed graph $G_\pi = ([n], A)$, where $(i, j) \in A$ if the elements $j, \overline{j}$ are contained within the interval formed by $i, \overline{i}$ (thus we also have $(i, i) \in A$ for all i).

Lemma 13. *For π fulfilling the conditions of Theorem 12, G_π is the reflexive transitive closure of a* branching[2]*. Furthermore, the transitive closure of any branching is the graph G_π for some valid π.*

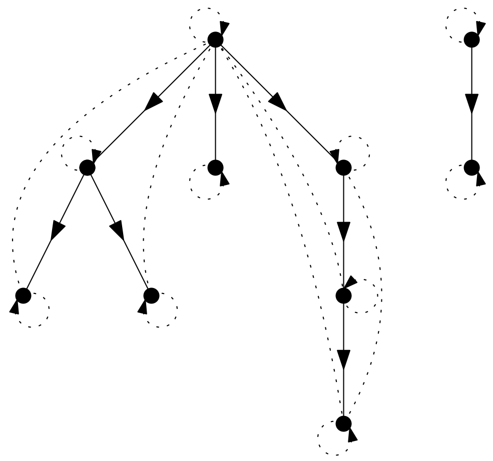


Figure 5: A branching (solid) with the edges added by taking the reflexive transitive closure (dotted).

Proof. We prove both directions independently.

[2]An *arborescence* is a directed rooted tree with all edges pointing away from the root. A *branching*, or a forest of arborescences, is the union of disjoint arborescences.

First, we show that for any valid π, G_π is the reflexive transitive closure of a branching. By definition, G_π contains a loop at every vertex. Furthermore, G_π must be transitive as it is built from the transitive relation of relative containment of intervals. We can assign each interval (and therefore each vertex in G_π) a depth indicating the number of intervals it is contained in. Whenever an interval is contained in two other intervals, one of these two intervals must contain the other. Therefore an interval of depth k is contained in exactly one interval of depth l for any $l < k$ and thus each vertex in G_π has exactly one incoming edge from any lower depth. We conclude that G_π must be the reflexive transitive closure of a branching.

Now we show that any reflexive transitive closure of a branching can be realized by some valid π. Each arborescence of the underlying branching forms an independent part of the permutation, ordered arbitrarily. The interval corresponding to the root of the arborescence encompasses the intervals of all its descendants. The descendants of the root again form a reflexive transitive closure of a branching, which can be converted into a permutation recursively. $\square$

3.2 Simple Extensions of Cyclic-P-Matroids

We study only a small subclass of the extensions of cyclic-P-matroids:

Definition 14 ([15, Proposition 8.18])**.** A *simple extension* $\widehat{\mathcal{M}}$ of a cyclic-P-matroid is an oriented matroid $\widehat{\mathcal{M}} = (\widehat{E_{2n}}, \widehat{\mathcal{C}})$ with $\pi \cdot ({}_{-F}\widehat{\mathcal{M}}) = \mathcal{A}^{2n+1,n}$ for some permutation π of $\widehat{E_{2n}} = E_{2n} \cup \{q\}$ with $\pi(q) = q$ and $F \subseteq \widehat{E_{2n}}$, such that π and F restricted to E_{2n} fulfill the conditions of Theorem 12.

We will first look at the USOs associated to simple extensions of cyclic-P-matroids with q being the last element in the permutation, i.e., $x_q > x_{2n}$.

If q is at the end of the permutation, there is always the same number (n) of elements before q in any circuit C. When reading off the fundamental circuit $C(B, q)$, we will therefore always assign the first element the same sign (before possibly flipping it due to it being in F).

Lemma 15. *Let $\widehat{M}$ be a simple extension of a cyclic-P-matroid with $\pi \cdot ({}_{-F}\widehat{\mathcal{M}}) = \mathcal{A}^{2n+1,n}$, with q last in π. The USO u associated to $\widehat{M}$ is a Matoušek-type USO with dimension influence graph $G_{\pi'}$, where π' is π constrained to E_{2n}.*

Proof. We will prove that for any vertex v, we have $u(v)_j \neq u(v \oplus e_i)_j$ if and only if $(i, j) \in E(G_{\pi'})$. As $G_{\pi'}$ is the reflexive closure of an acyclic graph, this proves that the USO is a Matoušek-type USO with dimension influence graph $G_{\pi'}$.

Let $v \in \{0,1\}^n$ be some vertex of u, and C_v its corresponding fundamental circuit with support $\underline{C_v} = B \cup \{q\}$. Let $i \in [n]$ be some dimension. Note that B contains either i or $i + n$. Assume w.l.o.g. that $i \in B$, and $\pi'(i) < \pi'(i + n)$. We can now split the set $B \setminus \{i\}$ in three parts along π': The part before i, the part between i and $i + n$, and the part after $i + n$. We will analyze what happens to the sign of the elements in each part when i is removed from the circuit and replaced by $i + n$.

The sign of an element j of the first part will not change, as neither i nor $i+n$ is located before j. This means that the number of elements of the circuit before j and thus its sign in the circuit stays the same. The same holds for all elements of the third part, as both i and $i+n$ are located before them. For an element j of the middle part, the number of elements in the circuit coming before j decreases by one when i is exchanged with $i+n$, and therefore its sign in the circuit flips.

This shows that $u(v \oplus e_i)$ differs from $u(v)$ in exactly the dimensions whose elements lie between i and $i+n$ in π' (including i itself), which is exactly the set of out-neighbours of i in $G_{\pi'}$. We conclude that u is a Matoušek-type USO with dimension influence graph $G_{\pi'}$. □

We are now ready to characterize the realizable Matoušek-type USOs.

Theorem 16. *A Matoušek-type USO is realizable if and only if its dimension influence graph is the reflexive transitive closure of a branching. Furthermore, every realizable Matoušek-type USO is the USO associated with a simple extension of a cyclic-P-matroid with q as the last element of π.*

Proof. Let G be a reflexive closure of an acyclic graph that is not the reflexive transitive closure of a branching. Assume first that G is not transitive. In this case G contains three vertices x, y, and z, with $(x,y),(y,z) \in E$, but $(x,z) \notin E$. This means the graph contains the left forbidden subgraph of Figure 3. Thus, assume that G is transitive. As G is not the transitive closure of a branching, there must be a vertex y that has an incoming edge from at least two vertices x and z, with no edge between x and z. Thus G contains the right forbidden subgraph of Figure 3. By Lemma 11, Matoušek-type USOs with dimension influence graph G are not realizable.

On the other hand, let G be the reflexive transitive closure of a branching. Combining Lemmas 13 and 15 we know that there is a Matoušek-type USO u with dimension influence graph G associated with a simple extension of a cyclic-P-matroid with q as the last element of π, and u is realizable. Mirroring of dimensions can be easily implemented in simple extensions of cyclic-P-matroids by appropriately adapting F. Thus, all Matoušek-type USOs with this dimension influence graph are realizable and associated with such a simple extension. □

Note that we can easily find a P-LCP instance realizing every realizable Matoušek-type USO, by first determining a simple extension of a cyclic-P-matroid, and then applying Equation (1) to the matrix of $2n+1$ (permuted and negated) points on the moment curve realizing that matroid.

Having understood the USOs associated with simple extensions of cyclic-P-matroids with q at the end of π, we would like to understand the influence of moving q towards the front of π. As we will see, this does not change the set of USOs we obtain.

Observation 17. *Let $\widehat{\mathcal{M}}$ be some simple extension of a cyclic-P-matroid with corresponding USO u. If q is pushed towards the front of π past some element $i \in [n]$, the direction of all edges between vertices in the lower i-facet is flipped.*

Similarly, if q is pushed past an element $i + n \in [n]$, the direction of all edges between vertices in the upper i-facet is flipped.

Observation 18. *If all edges within some i-facet of a Matoušek-type USO u are flipped, this changes whether (i, j) is an edge in the dimension influence graph of u for all $j \neq i$.*

When moving q towards the front of π, facets are flipped from the back of π' towards the front. The flipped facets therefore form a suffix of π'. Note that if both the upper and lower facet of some dimension are flipped, this yields an isomorphic USO. The sets of dimensions for which exactly one facet can be flipped at the same time therefore form a path in $G_{\pi'}$ starting at some root, following only edges of the underlying branching (solid arrows in Figure 5).

Since under this operation the associated USO remains a realizable Matoušek-type USO, we get the following observation:

Observation 19. *Let G be the reflexive transitive closure of some branching. Let S be the set of vertices on some directed path from some root of G, such that there is no $v \notin S$ with vertices $u, w \in S$ and $(u, v), (v, w) \in E$. Flipping whether $(s, t) \in E$ for all $s \in S$ and all $t \neq s$ results in some graph G_S which is also the reflexive transitive closure of a branching.*

Intuitively, this operation removes the vertices of S from their descendants, reverses their order, and adds them as a parent to all other vertices of the same depth, as can be seen in Figure 6.

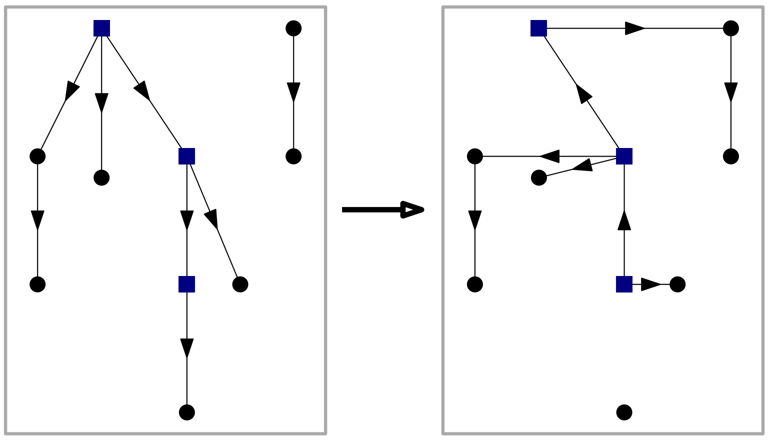


Figure 6: Flipping whether $(s, t) \in E$ for all s in some path (blue square vertices) starting at a root and all $t \neq s$. The transitive edges are left out for clarity.

Putting together Observations 17 to 19, we get the equivalences of our first main result:

Theorem 1. *The following families of USOs are all equivalent:*

- *The USOs associated with simple extensions of cyclic-P-matroids*

- *The realizable Matoušek-type USOs*
- *The Matoušek-type USOs fulfilling the Holt-Klee condition*

3.3 Counting Matoušek-type USOs

Given the results above we can easily count the number of (realizable) Matoušek-type USOs using classical graph-theoretic results. For general Matoušek-type USOs, the dimension influence graph can be the reflexive closure of any directed acyclic graph.

Lemma 20. *There are $2^{\Theta(n^2)}$ Matoušek-type USOs.*

Proof. Each pair of dimension influence graph and sink location describes a distinct Matoušek-type USO. There are 2^n sink locations, and there are $2^{\Theta(n^2)}$ labelled directed acyclic graphs on n vertices [22]. Thus there are $2^n \cdot 2^{\Theta(n^2)} \in 2^{\Theta(n^2)}$ Matoušek-type USOs. □

Similarly, a realizable Matoušek-type USO can be described by the branching underlying the dimension influence graph, and the sink location.

Lemma 21. *There are $2^n \cdot (n+1)^{n-1} \in 2^{\Theta(n \log n)}$ realizable Matoušek-type USOs.*

Proof. By Cayley's formula [3], there are $(n+1)^{n-1}$ labelled rooted forests on n vertices. By directing all edges away from the roots, there is a bijection from the set of labelled rooted forests to the set of labelled branchings. All branchings have distinct reflexive transitive closures, thus there are $(n+1)^{n-1}$ valid dimension influence graphs for realizable Matoušek-type USOs. □

As each vertex evaluation only provides n bits of information, these counts also give us lower bounds on the number of vertex evaluations required to *distinguish* Matoušek-type USOs, but sink-finding can of course be easier.

Corollary 22. *It takes at least $\Omega(n)$ vertex evaluations to distinguish all Matoušek-type USOs, and at least $\Omega(\log n)$ vertex evaluations to distinguish all realizable Matoušek-type USOs.*

4 Sink-Finding in Matoušek-type USOs

We now switch our attention from the *structure* of (realizable) Matoušek-type USOs to the *algorithmic query complexity* of sink-finding on (realizable) Matoušek-type USOs. We first introduce an alternative view on the sink-finding problem in Matoušek-type USOs.

4.1 Equivalence of Sink-Finding and Solving $\boldsymbol{Mx = y}$

By simply rearranging some terms in Definition 7, we get the following observation:

Observation 23. *An orientation u is a Matoušek-type USO with dimension influence graph G given by the adjacency matrix M if and only if it fulfills*

$$\forall x, y \in \{0,1\}^n : u(x) \oplus u(y) = M(x \oplus y).$$

This looks quite innocent, and maybe not too useful, but we can use this equation to formulate an algebraic problem which is equivalent to sink-finding in Matoušek-type USOs.

Definition 24 ($\boldsymbol{Mx = y}$ Problem)**.** For a matrix $M \in \{0,1\}^{n\times n}$ and a vector $y \in \{0,1\}^n$, the associated $Mx = y$ problem is to find the vector $x \in \{0,1\}^n$ fulfilling $Mx = y$. M is not explicitly included as part of the problem instance, but only an oracle is provided to the algorithm. The oracle answers *matrix-vector queries*: for any query $q \in \{0,1\}^n$, it returns Mq.

Using Observation 23, we can now show that up to a single additional query, the query complexities of the sink-finding problem and the $Mx = y$ problem are the same. We formalize this in the following theorem:

Theorem 25. *Let $\mathcal{U}$ be a subclass of Matoušek-type USOs closed under reorientations. There exists a deterministic algorithm $\mathcal{A}$ to find the sink in USOs in $\mathcal{U}$ in $f(n)$ vertex evaluations if and only if there exists a deterministic algorithm $\mathcal{B}$ to find x fulfilling $Mx = y$ where M can be the adjacency matrix of the dimension influence graph of any USO $u \in \mathcal{U}$ in $f(n) - 1$ matrix-vector queries.*

Proof. We first prove the “if” direction. Given an algorithm $\mathcal{B}$ for solving $Mx = y$ in $f(n) - 1$ queries we construct a sink-finding algorithm $\mathcal{A}$. The algorithm $\mathcal{A}$ first chooses an arbitrary vertex of the USO u, say $\mathbf{0}$, and queries it to receive its outmap $u(\mathbf{0})$. It sets $y := u(\mathbf{0})$ and sets M to the (still unknown) adjacency matrix of the dimension influence graph of u. If $\mathcal{B}$ makes a query $q \in \{0,1\}^n$, $\mathcal{A}$ can simulate the matrix-vector oracle and answer this query by querying the vertex q in the USO vertex evaluation oracle. The reply Mq can be computed as $u(\mathbf{0}) \oplus u(q)$ by Observation 23. Once $\mathcal{B}$ has found x fulfilling $Mx = y$ (in at most $f(n) - 1$ queries by assumption), $\mathcal{A}$ knows that x must be the sink, as

$$y = Mx = u(\mathbf{0}) \oplus u(x) = y \oplus u(x) \text{ and thus } u(x) = \mathbf{0}.$$

For each query of $\mathcal{B}$, $\mathcal{A}$ had to perform one query, with an additional query at the beginning to determine $y = u(\mathbf{0})$. In conclusion, $\mathcal{A}$ required at most $f(n) - 1 + 1 = f(n)$ queries.

Next, we prove the “only if” direction. Given an algorithm $\mathcal{A}$ for finding the sink in a USO u from $\mathcal{U}$ in $f(n)$ queries, we construct an algorithm $\mathcal{B}$ to solve $Mx = y$. When $\mathcal{A}$ makes its first query, vertex v_0, $\mathcal{B}$ simulates the vertex evaluation oracle and answers with $u(v_0) := y$ from its given instance. Whenever $\mathcal{A}$ makes a query v, $\mathcal{B}$ computes the outmap $u(v)$ using

$$u(v) := u(v_0) \oplus (M(v \oplus v_0)) = y \oplus (M(v \oplus v_0)).$$

By Observation 23, u is thus a Matoušek-type USO with dimension influence graph with adjacency matrix M and with $u(v_0) = y$. As M is the dimension influence graph of a USO in $\mathcal{U}$, and $\mathcal{U}$ is closed under reorientations, u must be in $\mathcal{U}$. Once $\mathcal{A}$ has found the sink s (in at most $f(n)$ queries by assumption) $\mathcal{B}$ can compute the solution $x := s \oplus v_0$ to the system $Mx = y$, as

$$M(s \oplus v_0) \overset{\text{Obs.23}}{=} u(s) \oplus u(v_0) = \mathbf{0} \oplus u(v_0) = y.$$

For each query performed by $\mathcal{A}$ apart from the first one, $\mathcal{B}$ had to perform exactly one query. In conclusion, $\mathcal{B}$ required at most $f(n) - 1$ queries. □

As the mapping $x \mapsto Mx$ computed by the oracle is linear and invertible, we can make the following two observations about any optimal algorithm for the $Mx = y$ problem:

Observation 26. *If x' is a linear combination of previously asked queries $x^{(1)}, \ldots, x^{(k)}$, the response Mx' can be computed without querying the oracle. Thus, every optimal algorithm only emits linearly independent queries.*

Observation 27. *If y is a linear combination of previously given replies $y^{(1)}, \ldots, y^{(k)}$, i.e., if $y \in \operatorname{span}(y^{(1)}, \ldots, y^{(k)})$, the algorithm can find the solution x with no additional queries.*

In some places it will be useful to interpret the matrix-vector queries in terms of vertex sets of the dimension influence graph.

Observation 28. *For a query $q \in \{0,1\}^n$, the vertex set $\{i \in [n] : (Mq)_i = 1\}$ contains exactly the vertices which have an odd number of in-neighbors among the vertex set $\{i \in [n] : q_i = 1\}$ in G.*

4.2 The General Case

It is easy to derive algorithms using a linear number of queries to find the sink in Matoušek-type USOs.

Theorem 29. *There exists an algorithm that finds the sink of an n-dimensional Matoušek-type USO in n vertex evaluations.*

Proof. The JumpAntipodal algorithm begins at some arbitrary vertex $v := v_0$. In each step, it queries v, then jumps to the vertex $v \oplus o(v)$. As Matoušek-type USOs are decomposable, the USO is combed in some dimension d. After the first jump, the algorithm arrives at a vertex v' in the facet towards which all edges of dimension d are pointed. This facet can never be left again, and it is itself a decomposable USO of dimension $n - 1$. Applying this argument recursively, after at most n jumps JumpAntipodal has reached a 0-dimensional USO — a single vertex that must be the sink. It does not need to query this vertex anymore, and thus requires at most n vertex evaluations. □

As can be seen from this proof, JumpAntipodal requires only n vertex evaluations on all decomposable USOs, not only on the Matoušek-type USOs. In contrast, it has been shown that even on some acyclic USOs it requires exponentially many queries [19]. Note that JumpAntipodal only finds the sink. If we are interested in recovering the whole structure of a Matoušek-type USO, this can be easily achieved in n matrix-vector queries (by querying $e_1, \dots, e_n$), or $n+1$ vertex evaluations.

We are now going to prove Theorem 2, the matching lower bound to Theorem 29. The proof works in the framework of the $Mx = y$ problem, and is quite algebraic in nature. Algorithm 1 shows a strategy for an adversary to adaptively construct the matrix M in a way to force every deterministic algorithm to use at least $n-1$ queries to find x. In terms of the dimension influence graph, this strategy can be seen as picking a sink j in every iteration, and adding/removing some edges towards j. This ensures that the graph represented by M always remains acyclic apart from the loops at each vertex.

Algorithm 1 Adversarial Construction

```
1:  $M^{(0)} \leftarrow I$
2:  **for** $k \in \{1, \dots, n-1\}$ **do**
3:      $x^{(k)} \leftarrow$ new linearly independent query from algorithm
4:      **if** $y \in \operatorname{span}(M^{(k-1)}x^{(1)}, \dots, M^{(k-1)}x^{(k)})$ **then**
5:          $X \leftarrow \begin{pmatrix} x^{(1)} & \cdots & x^{(k)} \end{pmatrix}^T$
6:          $freevars \leftarrow$ free variables of linear system of equations determined by $X$
7:          Pick $z^{(k)}$ such that $Xz^{(k)} = e_k$ and $z_i^{(k)} = 0$ for all $i \in freevars$
8:          Pick $j \in freevars$ such that $e_j$ is an eigenvector of $M^{(k-1)}$          ▷ $j$ is a sink
9:          $M^{(k)} \leftarrow M^{(k-1)} + e_j {z^{(k)}}^T$          ▷ Add $z^{(k)}$ to $j$-th row of $M^{(k-1)}$
10:     **else**
11:         $M^{(k)} \leftarrow M^{(k-1)}$          ▷ No need to change $M$
12:     Answer query with $y^{(k)} := M^{(k)}x^{(k)}$
```

Theorem 2. *For every deterministic sink-finding algorithm $\mathcal{A}$ and any $n \geqslant 2$, there exists an n-dimensional Matoušek-type USO on which $\mathcal{A}$ requires at least n vertex evaluations to find the sink.*

Proof. We show that the adversarial construction in Algorithm 1 ensures that no algorithm can find the solution to the $Mx = y$ problem in fewer than $n-1$ queries. To prove this, we first show four auxiliary properties of the instances constructed by Algorithm 1:

- Feasibility: We can always pick $z^{(k)}$ and j on lines 7 and 8 as defined.
- Consistency: The replies to previous queries remain consistent.
- Legality: The graph defined by M remains a legal dimension influence graph, i.e., it is acyclic with added loops at every vertex.

- Uncertainty: After $k < n-1$ queries, the algorithm cannot yet distinguish between some instances with different solutions.

Feasibility: By Observation 26, we can assume all queries $x^{(i)}$ to be linearly independent. Thus, the $k \times n$-dimensional matrix X has rank $k < n$ and is underdetermined. We can thus set all free variables of $z^{(k)}$ given $Xz^{(k)} = e_k$ to be zero, and get a unique $z^{(k)}$ to be picked at line 7.

Whenever an additional linearly independent row is added to X, exactly one variable is removed from the set of free variables. Thus, after k queries $n-k$ free variables remain. These were also free variables in all previous iterations, and therefore all vectors $z^{(k')}$ of iterations $k' \leqslant k$ have a 0 at these coordinates. Therefore, for any variable $j \in freevars$, it holds that the column j of $M^{(k-1)}$ must be equal to e_j, and thus e_j must be an eigenvector of $M^{(k-1)}$. We conclude that any $j \in freevars$ can be picked on line 8.

Consistency: We prove that the possible change to $M^{(k-1)}$ in iteration k has no effect on any query $x^{(k')}$ for $k' < k$. Note that if M is changed in iteration k, we have $M^{(k)} = M^{(k-1)} + e_j {z^{(k)}}^T$, and thus $M^{(k)}x^{(k')} = M^{(k-1)}x^{(k')} + e_j {z^{(k)}}^T x^{(k')}$. As $z^{(k)}$ was picked such that $Xz^{(k)} = e_k$, we have in particular ${x^{(k')}}^T z^{(k)} = 0$, and thus

$$M^{(k)}x^{(k')} = M^{(k-1)}x^{(k')} + e_j {z^{(k)}}^T x^{(k')} = M^{(k-1)}x^{(k')} + e_j 0 = M^{(k-1)}x^{(k')}.$$

Legality: As we start with $M^{(0)} = I$, we start with a loop at every vertex. As $z^{(k)}$ is added to the j-th row, and $z_j^{(k)} = 0$, these loops are never removed. As j is picked such that e_j is an eigenvector of $M^{(k-1)}$, the j-th column of $M^{(k-1)}$ must be equal to e_j. This corresponds to the vertex j having no outgoing edges apart from the loop, i.e., j is a sink. Changing the j-th row of $M^{(k-1)}$ only adds or removes edges pointing *towards* j. As j is a sink, this cannot introduce any cycles. The graph described by $M^{(k)}$ thus remains a legal dimension influence graph.

Uncertainty: We first show that after each iteration, the algorithm cannot deduce the solution through linear combination, i.e.,

$$\forall 0 \leqslant k \leqslant n-1: \quad y \notin \operatorname{span}(M^{(k)}x^{(1)}, \ldots, M^{(k)}x^{(k)}). \tag{2}$$

We prove this by induction. For $k = 0$, the statement is trivially true. Assuming it holds for some $k-1 < n-1$, we show that it also holds for k. If in the k-th iteration the condition at line 4 is false, the statement also trivially follows. Otherwise, we must have

$$y \notin \operatorname{span}(M^{(k-1)}x^{(1)}, \ldots, M^{(k-1)}x^{(k-1)}), \text{ but} \tag{3}$$
$$y \in \operatorname{span}(M^{(k-1)}x^{(1)}, \ldots, M^{(k-1)}x^{(k-1)}, M^{(k-1)}x^{(k)}). \tag{4}$$

Since j is picked as a free variable of X, $e_j \notin \operatorname{span}(x^{(1)}, \ldots, x^{(k)})$, and as e_j is an eigenvector of the (invertible) $M^{(k-1)}$, it must also hold that

$$e_j \notin \operatorname{span}(M^{(k-1)}x^{(1)}, \ldots, M^{(k-1)}x^{(k)}). \tag{5}$$

Equations (3) and (4) show that $M^{(k-1)}x^{(k)}$ is a required element in the linear combination of y. Equation (5) shows that e_j cannot be expressed as a linear combination of the $M^{(k-1)}x^{(k')}$. Therefore, if we add e_j to the required element $M^{(k-1)}x^{(k)}$, y can not be in the span anymore, i.e.,

$$y \notin \operatorname{span}(M^{(k-1)}x^{(k)} + e_j, M^{(k-1)}x^{(1)}, \dots, M^{(k-1)}x^{(k-1)}). \tag{6}$$

This is equivalent to the desired Equation (2) for k, as $M^{(k)}x^{(k)} = M^{(k-1)}x^{(k)} + e_j$, and as shown in paragraph "Consistency", $M^{(k)}x^{(k')} = M^{(k-1)}x^{(k')}$ for all $k' < k$.

We can now show that after $k < n-1$ queries, there exist two matrices which are both consistent with the given replies but have different solutions. The first such matrix is $M^{(k)}$. The second matrix is the matrix $M^{(k+1)}$ constructed by the adversary if it would be given the solution for $M^{(k)}$ as an additional linearly independent query $x^{(k+1)} := {M^{(k)}}^{-1}y$. As proven in previous paragraphs, $M^{(k+1)}$ is legal and consistent with $M^{(k)}$ on all queries $x^{(1)}, \dots, x^{(k)}$. Equation (2) implies that $x^{(k+1)}$ is not the solution to $M^{(k+1)}$, proving that the two indistinguishable matrices have different solutions.

Conclusion: Given any $n-1$ queries, Algorithm 1 produces a series of legal matrices $M^{(k)}$ (Feasibility + Legality) which are always consistent with the previously given replies (Consistency). The algorithm cannot know the solution in fewer than $n-1$ queries, as it cannot distinguish between matrices with different solutions (Uncertainty). By Theorem 25, we conclude that no algorithm can find the sink of an n-dimensional Matoušek-type USO in fewer than n vertex evaluations in the worst case. □

4.3 The Realizable Case

In this section we prove our second main result, the upper bound for the realizable case.

Theorem 3. *The sink of any n-dimensional* realizable *Matoušek-type USO can be found deterministically using $O(\log^2 n)$ vertex evaluations in the worst case.*

To prove Theorem 3, we provide a concrete algorithm in the matrix-vector query model to recover the matrix M. The algorithm makes heavy use of the structure of the graph G described by this matrix, which has to be the reflexive transitive closure of a branching (recall Theorem 16). Recall that we can view the matrix-vector queries also as sets of vertices of the dimension influence graph (Observation 28). In a slight abuse of notation, we will sometimes use the name v of a vector $v \in \{0,1\}^n$ to also denote the set $\{i \in [n] : v_i = 1\}$.

We first take a closer look at the structure of G. The underlying branching (technically, the unique reflexive transitive reduction of G) can be decomposed into levels, where the roots are on level 0, and the children of a vertex on level ℓ are on level $\ell + 1$. In the reflexive transitive closure G, we can see that the in-degree of each vertex is equal to its level plus 1 (due to the loops).

Observation 30. *A vertex v on level ℓ has exactly $\ell+1$ incoming edges, and v has exactly one in-neighbor on each level $\ell' \in \{0, \dots, \ell\}$.*

Definition 31. For a vertex v on level ℓ and some level $\ell' < \ell$, the *ℓ'-ancestor of* v is the unique in-neighbor of v on level ℓ'. The *parent of* v is the $\ell - 1$-ancestor of v. The maximum level of any vertex in G is denoted by ℓ_{max}.

Our proposed algorithm works in two main phases. In the first phase, the *levelling*, it determines the level of each vertex in $O(\log n)$ queries. In the second phase, we use a divide-and-conquer approach to partition the vertices and perform queries to find the edges within each partition simultaneously, requiring $O(\log^2 n)$ queries in total.

Algorithm 2 Levelling

```
1: $lvl \leftarrow$ array of $n$ zeroes                      ▷ Stores the level for every vertex
2: $q \leftarrow \mathbf{1}$
3: for $i \in \{0, \ldots, \lceil \log_2 n \rceil - 1\}$ do
4:     $r \leftarrow (Mq) \oplus q$                                       ▷ Issues 1 query
5:     for $v \in \{1, \ldots, n\}$ do
6:         if $r_v = 1$ then
7:             $lvl[v] \leftarrow lvl[v] + 2^i$
8:     $q \leftarrow q \odot r$                                 ▷ Bit-wise "and" operation
9: return $lvl$
```

Lemma 32. *Algorithm 2 correctly computes the level of each vertex, using $O(\log n)$ queries.*

Proof. Algorithm 2 issues $O(\log n)$ queries, one per iteration of the loop at line 3.

To prove correctness, we show that in each iteration i, the vertices in the vector r are exactly those on levels ℓ where $Bin(\ell)_i = 1$. From this follows that the level of each vertex is correctly recovered on line 7, one bit at a time. We show this by induction on i.

Note that a vertex v is in $r = (Mq) \oplus q$ if it has an odd number of non-self in-neighbors in q, i.e., in-neighbors in $q \setminus \{v\}$.

For $i = 0$ as the base case of this induction, $q = \mathbf{1}$ and r therefore contains all vertices with an odd number of non-self in-neighbors. By Observation 30, these are exactly the vertices on odd levels, i.e., those on levels where the least significant bit is 1.

For the induction step, assume that for some i, the statement holds for all iterations $i' \leqslant i$. Thanks to the bit-wise "and" on line 8, the queried vertices q in iteration $i + 1$ are the vertices on levels ℓ' with $Bin(\ell')_{i'} = 1$ for all $i' \leqslant i$, i.e., the binary representation of ℓ' ends with at least i ones. The vector $(Mq) \oplus q$ contains all vertices with an odd number of non-self in-neighbors among these queried vertices. By Observation 30, these are all vertices on levels ℓ with an odd number of queried levels strictly above, i.e., with $|\{\ell' < \ell : \forall i' \leqslant i, Bin(\ell')_{i'} = 1\}| =_2 1$. This holds exactly for the levels ℓ with $Bin(\ell)_{i+1} = 1$, thus proving the claim. □

We now know how to compute the level of each vertex in $O(\log n)$ time. It remains to show that we can also determine the edges connecting each consecutive two levels, and thus recover the whole graph.

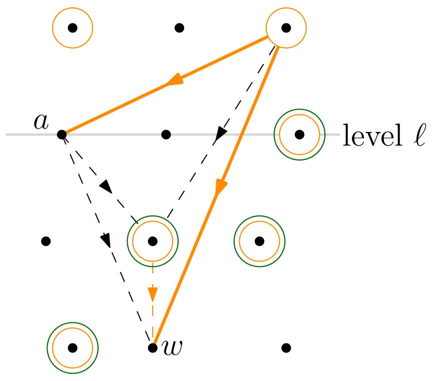


Figure 7: The orange-circled vertices are those included in the query q, the green-circled vertices are also in the query q'. All loops and all edges not induced by the vertices in q, a, and w are omitted for legibility. The effect (bold) of the two vertices in q but not in q' on a is the same as their effect on w, as a is an ancestor of w and the graph is transitive. This shows that $(Mq')_w$ can be computed from $(Mq)_w$ and $(Mq)_a$.

We first give the intuition for a simple strategy that given a level ℓ finds the ℓ-ancestor of all vertices on levels $\geqslant \ell + 1$. We can perform $\lceil \log_2 n \rceil$ queries, where the i-th query $q^{(i)}$ contains all vertices v such that $\{v \text{ on level } \ell \text{ with } Bin(v)_i = 1\}$. If a vertex is "hit", i.e., it is contained in the i-th reply $Mq^{(i)}$, we know that its ℓ-ancestor must be in $q^{(i)}$. As each binary representation uniquely determines an integer, after all $\lceil \log_2 n \rceil$ queries, the ℓ-ancestor is found.

It would be too costly to use this procedure for all levels on their own, as there can be up to n levels. We thus make use of the following observation, which follows directly from the transitivity and reflexivity of the dimension influence graph, as illustrated in Figure 7:

Observation 33. *Let ℓ be some level, q be some query and Mq the corresponding response. Furthermore, let w be some vertex on level $\ell' > \ell$, with the ℓ-ancestor of w being a. We form the alternative query q' with $q'_v = 1 \iff (q_v = 1 \wedge level(v) \geqslant \ell \wedge v \neq a)$ by removing from q the vertex a as well as all vertices strictly above level ℓ. It holds that*

$$(Mq')_w = (Mq)_w \oplus (Mq)_a.$$

This observation allows us to "filter out" the effect of querying vertices above a certain level ℓ on the vertices below ℓ, as long as their ℓ-ancestors are known. Using this crucial tool, we can now solve the $Mx = y$ problem with a divide-and-conquer approach. If we use the previously given strategy to find all ℓ-ancestors for a level ℓ roughly in the middle of the branching, we can then split the graph into two halves, the one above ℓ and the one below ℓ. Using Observation 33, we can proceed in each of the two subproblems simultaneously, as the effect of queries used to make progress in the upper half can be filtered out of the responses to the queries used to make progress in the lower half. We describe this process in Algorithm 3.

This algorithm keeps a list of subproblems, where each subproblem is described by an interval of levels $\{a_i, a_i+1, \ldots, b_i\}$, denoted by the tuple (a_i, b_i). It is important that these subproblems are disjoint. In each iteration of the main loop, a median level m_i of each subproblem is picked, and the previously mentioned procedure is performed to determine the m_i-ancestors of all vertices in the subproblem (lines 5–17). Observation 33 is applied with $\ell := b_i$ to filter out the effect of the vertices queried for the subproblem (a_i, b_i) from the later subproblems (lines 13 and 14). The transitive property of the ancestor relation is applied to make sure that the m_i-ancestors are also known for all vertices on levels below the considered subproblem (a_i, b_i) (lines 18–20). Finally, each subproblem is split into two parts, the one above m_i, and the one strictly below m_i. Subproblems consisting of a single level are ignored, as they contain no more edges to discover (lines 21–26).

Lemma 34. *Knowing the level of each vertex, Algorithm 3 correctly determines the parent of every vertex v in $O(\log^2 n)$ queries.*

Proof. Let us first assume that all *ancestor* values required on line 14 are already known. By Observation 33 being applied on lines 13 and 14, the values r_v read on line 16 are only influenced by the queried vertices on the level m_i — the effect of earlier subproblems is filtered out. Lines 4–17 therefore correctly determine the m_i-ancestors for all vertices on levels in $\{m_i + 1, \dots, b_i\}$. It only remains to prove that all *ancestor* values required on line 14 are known, and that in the end of the algorithm, *all* $ancestor[\ell][v]$ values are known.

We show the invariant that at the beginning of the main while loop, the values $ancestor[b_i][v]$ are known for all $v \in [n]$ and all $(a_i, b_i) \in subproblems$. At the beginning of the algorithm, this trivially holds, as there is only one subproblem $(0, \ell_{max})$, and no vertex has an ℓ_{max}-ancestor. Whenever a subproblem (a_i, b_i) is split into (a_i, m_i) and $(m_i + 1, b_i)$, there is only one new end of a subproblem, namely m_i. On lines 4–17, the m_i-ancestor has just been computed for all vertices on levels in $\{m_i + 1, \dots, b_i\}$, and the b_i-ancestor was previously known for all vertices. On lines 18–20, this is combined to compute the m_i-ancestor for all vertices. The invariant thus holds.

To prove that all $ancestor[\ell][v]$ values are known at the end of the algorithm, we observe that every level is the end or the median of some subproblem at least once. In case the level is the end b_i of a subproblem, the b_i-ancestors are known by the previously proven invariant. In case the level is the median m_i of a subproblem, the m_i-ancestors are computed in that iteration.

We conclude that Algorithm 3 correctly determines all $ancestor[\ell][v]$ values, and thus also the parent of every vertex. In each iteration of the main while loop, $O(\log n)$ queries are issued, and as the size of the subproblems is halved in each iteration, there are at most $O(\log n)$ iterations. We conclude that Algorithm 3 requires $O(\log^2 n)$ queries. □

Proof of Theorem 3. Using Algorithm 2 to compute the level of each vertex in the dimension influence graph G and Algorithm 3 to then compute all ancestors of all vertices, G can be completely recovered in $O(\log^2 n)$ matrix-vector queries. Knowing M, $Mx = y$ can be solved with no additional queries using Gaussian elimination. By Theorem 25, the sink of a realizable Matoušek-type USO can be found in $O(\log^2 n)$ vertex evaluations. □

We believe that $\Theta(\log^2 n)$ is the best-possible number of queries to find the sink of a realizable Matoušek-type USO. Due to the rigid structure of the dimension influence graph, a large portion of the n bits of information in every reply from the oracle is redundant. We did not manage to prove this matching lower bound, but we can show a lower bound of $\Omega(\log n)$. Note that our algorithm solves the harder problem of recovering the whole structure of the USO, while the following lower bound holds for the easier problem of only finding the sink.

Theorem 35. *Every deterministic algorithm requires at least $\Omega(\log n)$ queries to find the sink of an n-dimensional realizable Matoušek-type USO in the worst case.*

Proof. We prove this in the matrix-vector query model. We set $y = \mathbf{1}$ and begin with $M = I$. As $y = \mathbf{1}$, the desired x such that $Mx = y$ is the vector corresponding to all roots of the dimension influence graph described by M, as their out-neighbor sets form a partition of all vertices.

During the construction, we enforce the invariant that the branching underlying the dimension influence graph is the union of disjoint paths. We call a path p *good*, if for each previously given query q, there is an even number of queried vertices within the path, i.e., $|q \cap p| =_2 0$.

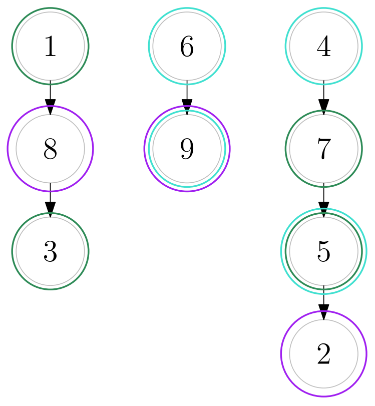
As long as there is at least one good path p_1 and at least one other path p_2, all replies given to the algorithm are also consistent with the graph in which p_2 is attached to the end of p_1. In this alternative graph, the first vertex of p_2 is not a root. These two graphs thus have different solutions but are indistinguishable to the algorithm, and we conclude that the algorithm cannot know the set of roots at this point.

Note that at the beginning of the construction, when $M = I$ and no queries have arrived yet, the branching underlying the graph consists of n paths of length 0, which are all good.

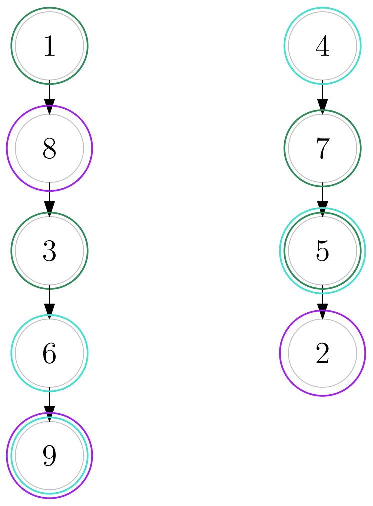
Whenever the algorithm queries an odd number of vertices of some good paths, these paths are paired up. When two paths p_1 and p_2 are paired up, p_2 is attached to the end of p_1 (see Figure 8). The query is then answered according to this new graph. As each previous query contained an even number of vertices in p_1 (as p_1 is good), the influence of these vertices onto the vertices in p_2 cancels out. The replies to these queries therefore remain consistent. As the newest query would have contained an odd number of vertices of both p_1 and p_2, the joined path is still good.

If there is an odd number of paths to be paired up, there is one leftover path. This path can not be paired up. It is no longer good, and will not be changed anymore.

We observe that when there are k good paths before a query, there are at least $\lfloor k/2 \rfloor$ good paths remaining after the query. We showed that the algorithm can only know the solution when there are no good paths left, or only a single path in total. As the graph contains n good paths before the first query, it takes at least $\Omega(\log n)$ queries until the algorithm can know the solution. $\square$



(a) Before the purple query.



(b) After the purple query.

Figure 8: After the green and cyan queries, all three paths in the left figure are good. To accommodate the purple query, the left and middle paths are paired up and joined. The combined path remains good. The right path is leftover, and is no longer good after the purple query. Loops and transitive edges are not shown.

5 Conclusion

We have shown that the Holt-Klee condition is also sufficient on the Matoušek-type USOs and thus fully characterized their realizable subset. Finding stronger necessary or some sufficient conditions for realizability in larger USO classes would be very interesting future work.

We have determined the query complexity of finding the sink in general Matoušek-type USOs exactly. For realizable Matoušek-type USOs, there remains an $\Theta(\log n)$ gap between our lower and upper bound. While it would be interesting to close this gap, our main result — the gap between the realizable and the general case — is already well established by our bounds.

As the best-known sink-finding algorithms are randomized, it would be desirable to establish a complexity gap also for randomized algorithms. The upper bound naturally carries over, as all deterministic algorithms are also randomized algorithms. The most natural approach to establish a lower bound would be applying Yao's principle [28]. On the flip side, it might also be possible to improve upon our algorithms both for the realizable and the general case using randomness, but we did not observe any straightforward benefit of randomness.

The most important open question implied by our complexity results is whether there are other (larger and more practically relevant) USO classes which also admit such a complexity gap. Ultimately, we hope for such a gap to exist for the class of *all* USOs. Considering the lack of a strong lower bound and good characterizations of realizability, this goal is still far away.

Finally, the connections between Matoušek-type USOs and D-cubes mentioned in Section 1.2 can be examined further. Are all or at least some of the realizable Matoušek-type USOs also D-cubes? Can our techniques used to find the sink of a Matoušek-type USO be adapted to work for the less rigid D-cubes?

Algorithm 3 Divide-And-Conquer

```
1: $\mathit{subproblems} \leftarrow \{(0, \ell_{max})\}$
          $\triangleright$ $\mathit{subproblems}$: a sorted set of disjoint intervals $(a_i, b_i)$ with $a_i < b_i < a_{i+1}$
2: $\mathit{ancestor} \leftarrow \ell_{max} \times n$-dimensional matrix of zeroes
   $\triangleright$ $\mathit{ancestor}[\ell][v]$: contains the $\ell$-ancestor of vertex $v$, or 0 if this is unknown/does not exist
3: **while** $\mathit{subproblems} \neq \emptyset$ **do**
4:     $q \leftarrow (0, \ldots, 0)^T$
5:     **for** $s \in \{0, \ldots, \lceil \log_2 n \rceil - 1\}$ **do**                    $\triangleright$ Perform binary search
6:         **for all** $(a_i, b_i) \in \mathit{subproblems}$ **do**
7:             $m_i \leftarrow \lfloor \frac{a_i + b_i}{2} \rfloor$                    $\triangleright$ Compute median level of subproblem
8:             **for all** vertices $v$ on level $m_i$ **do**
9:                 **if** $Bin(v)_s = 1$ **then**
10:                    $q_v \leftarrow 1$                                    $\triangleright$ Add $v$ to query
11:        $r \leftarrow Mq$                                                $\triangleright$ Issue a query
12:        **for all** $(a_i, b_i) \in \mathit{subproblems}$ **do**
13:            **for all** vertices $v$ on levels $> b_i$ **do**
14:                $r_v \leftarrow r_v \oplus r_{\mathit{ancestor}[b_i][v]}$          $\triangleright$ Filter effect of level $m_i$ on levels $> b_i$
15:            **for all** vertices $v$ on levels $\{m_i + 1, \ldots, b_i\}$ **do**
16:                **if** $r_v = 1$ **then**                                 $\triangleright$ Detect ancestor
17:                    $\mathit{ancestor}[m_i][v] \leftarrow \mathit{ancestor}[m_i][v] + 2^s$
18:    **for all** vertices $v$, levels $\ell_1 > \ell_2$ **do**                      $\triangleright$ "Transitivify" $\mathit{ancestor}$
19:        **if** $\mathit{ancestor}[\ell_1][v] \neq 0$ and $\mathit{ancestor}[\ell_2][\mathit{ancestor}[\ell_1][v]] \neq 0$ **then**
20:            $\mathit{ancestor}[\ell_2][v] \leftarrow \mathit{ancestor}[\ell_2][\mathit{ancestor}[\ell_1][v]]$
21:    **for all** $(a_i, b_i) \in \mathit{subproblems}$ **do**                    $\triangleright$ Split subproblems in half
22:        remove $(a_i, b_i)$ from $\mathit{subproblems}$
23:        **if** $m_i > a_i$ **then**
24:            add $(a_i, m_i)$ to $\mathit{subproblems}$
25:        **if** $b_i > m_i + 1$ **then**
26:            add $(m_i + 1, b_i)$ to $\mathit{subproblems}$
```

References

[1] Anders Björner, Michel Las Vergnas, Bernd Sturmfels, Neil White, and Gunter M. Ziegler. *Oriented Matroids*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2nd edition, 1999. doi:10.1017/CBO9780511586507.

[2] Michaela Borzechowski and Simon Weber. On degeneracy in the P-matroid oriented matroid complementarity problem. In *Abstracts of the 39th European Workshop on Computational Geometry (EuroCG'23)*, pages 9(1)–9(7), 2023. arXiv:2302.14585.

[3] Arthur Cayley. A theorem on trees. *The Quarterly Journal of Mathematics*, 23:376–378, 1889.

[4] Raul Cordovil and Pierre Duchet. Cyclic polytopes and oriented matroids. *European Journal of Combinatorics*, 21:49–64, 2000. doi:10.1006/eujc.1999.0317.

[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:10.1016/j.jcss.2020.05.007.

[6] Jan Foniok, Bernd Gärtner, Lorenz Klaus, and Markus Sprecher. Counting unique-sink orientations. *Discrete Applied Mathematics*, 163:155–164, 2014. doi:10.1016/j.dam.2013.07.017.

[7] Komei Fukuda, Lorenz Klaus, and Hiroyuki Miyata. A new subclass of P-matrix linear complementarity problems. *IEICE Technical Report*, 113(50):25–32, 2013. URL: https://ken.ieice.org/ken/paper/20130517HBd7/.

[8] Yuan Gao, Bernd Gärtner, and Jourdain Lamperski. A new combinatorial property of geometric unique sink orientations, 2020. arXiv:2008.08992.

[9] Bernd Gärtner. The random-facet simplex algorithm on combinatorial cubes. *Random Structures & Algorithms*, 20(3):353–381, 2002. doi:10.1002/rsa.10034.

[10] Bernd Gärtner, Walter D. Morris jr., and Leo Rüst. Unique sink orientations of grids. *Algorithmica*, 51(2):200–235, 2008. doi:10.1007/s00453-007-9090-x.

[11] Bernd Gärtner and Leo Rüst. Simple stochastic games and P-matrix generalized linear complementarity problems. In *Fundamentals of Computation Theory (FCT'05)*, pages 209–220. Springer, 2005. doi:10.1007/11537311_19.

[12] Bernd Gärtner and Ingo Schurr. Linear programming and unique sink orientations. In *17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06)*, pages 749–757, 2006. doi:10.5555/1109557.1109639.

[13] Bernd Gärtner and Emo Welzl. Explicit and implicit enforcing - randomized optimization. In *Computational Discrete Mathematics: Advanced Lectures*, pages 25–46. Springer, 2001. doi:10.1007/3-540-45506-X_3.

[14] Fred Holt and Victor Klee. A proof of the strict monotone 4-step conjecture. *Contemporary Mathematics*, 223:201–216, 1999. doi:10.1090/conm/223/03138.

[15] Lorenz Klaus. *A fresh look at the complexity of pivoting in linear complementarity*. PhD thesis, ETH Zürich, 2012. doi:10.3929/ethz-a-007604201.

[16] Lorenz Klaus and Hiroyuki Miyata. Enumeration of PLCP-orientations of the 4-cube. *European Journal of Combinatorics*, 50:138–151, 2015. doi:10.1016/j.ejc.2015.03.010.

[17] Jiří Matoušek. Lower bounds for a subexponential optimization algorithm. *Random Structures & Algorithms*, 5(4):591–607, 1994. doi:10.1002/rsa.3240050408.

[18] Walter D. Morris jr. Randomized pivot algorithms for P-matrix linear complementarity problems. *Mathematical Programming*, 92(2):285–296, 2002. doi:10.1007/s101070100268.

[19] Ingo Schurr. *Unique Sink Orientations of Cubes.* PhD thesis, ETH Zürich, 2004. doi:10.3929/ethz-a-004844278.

[20] Ingo Schurr and Tibor Szabó. 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):627–642, 2004. doi:10.1007/s00454-003-0813-8.

[21] Steve Smale. Mathematical problems for the next century. *The Mathematical Intelligencer*, 20:7–15, 1998. doi:10.1007/BF03025291.

[22] Richard P. Stanley. Acyclic orientations of graphs. *Discrete Mathematics*, 5(2):171–178, 1973. doi:10.1016/0012-365X(73)90108-8.

[23] Alan Stickney and Layne Watson. Digraph models of Bard-type algorithms for the linear complementarity problem. *Mathematics of Operations Research*, 3(4):322–333, 1978. URL: https://www.jstor.org/stable/3689630.

[24] Tibor Szabó and Emo Welzl. Unique sink orientations of cubes. In *42nd IEEE Symposium on Foundations of Computer Science (FOCS'01)*, pages 547–555, 2001. doi:10.1109/SFCS.2001.959931.

[25] Michael J. Todd. Complementarity in oriented matroids. *SIAM Journal on Algebraic Discrete Methods*, 5(4):467–485, 1984. doi:10.1137/0605046.

[26] Simon Weber and Bernd Gärtner. A characterization of the realizable Matoušek unique sink orientations, 2021. arXiv:2109.03666.

[27] Simon Weber and Joel Widmer. Realizability makes a difference: A complexity gap for sink-finding in USOs. In *Algorithms and Data Structures (WADS'23)*, pages 704–718. Springer Nature Switzerland, 2023. doi:10.1007/978-3-031-38906-1_47.

[28] Andrew C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 222–227, 1977. doi:10.1109/SFCS.1977.24.