Discrete Comput Geom 31:627–642 (2004)
DOI: 10.1007/s00454-003-0813-8

Discrete & Computational
Geometry

Finding the Sink Takes Some Time: An Almost Quadratic Lower Bound for Finding the Sink of Unique Sink Oriented Cubes*

Ingo Schurr and Tibor Szabó

Theoretical Computer Science, ETH Zürich,
CH-8092 Zürich, Switzerland
{schurr,szabo}@inf.ethz.ch

Abstract. We give a worst-case $\Omega(n^2/\log n)$ lower bound on the number of vertex evaluations a deterministic algorithm needs to perform in order to find the (unique) sink of a unique sink oriented n-dimensional cube. We consider the problem in the vertex-oracle model, introduced in [18]. In this model one can access the orientation implicitly, in each vertex evaluation an oracle discloses the orientation of the edges incident to the queried vertex. An important feature of the model is that the access is indeed arbitrary, the algorithm does *not* have to proceed on a directed path in a simplex-like fashion, but could "jump around". Our result is the first superlinear lower bound on the problem. The strategy we describe works even for acyclic orientations. We also give improved lower bounds for small values of n and fast algorithms in a couple of important special classes of orientations to demonstrate the difficulty of the lower bound problem.

1. Introduction

Notation, *Definitions*. For our purposes a cube is the power set of a set. More precisely, for $A \subseteq B$ finite sets the cube $\mathfrak{C} = \mathfrak{C}^{[A,B]}$ is the edge labeled graph with vertex set $\mathrm{V}(\mathfrak{C}) := [A, B] := \{X \mid A \subseteq X \subseteq B\}$, edge set

$$\mathrm{E}(\mathfrak{C}) := \{\{v, v \oplus \{l\}\} \mid v \in \mathrm{V}(\mathfrak{C}), l \in B \setminus A\}$$

* An extended abstract of this paper appeared in the *Proceedings of ESA* 2002. Both authors were supported by the joint Berlin/Zürich graduate program Combinatorics, Geometry, and Computation (CGC), financed by German Science Foundation (DFG) and ETH Zurich, and Tibor Szabó was also supported by NSF Grant DMS 99-70270.

and edge labeling

$$\lambda(\{v, v \oplus \{l\}\}) := l,$$

where the symbol $\oplus$ denotes the symmetric difference of two sets. We say that an edge e is *l-labeled* if $\lambda(e) = l$, and for a subset L of the labels we say *L-labeled edges* to refer to the set of all edges which are l-labeled with some $l \in L$.

From a purely combinatorial point of view, the cube $\mathfrak{C}^{[A,B]}$ is sufficiently described by its *label set* $\operatorname{carr} \mathfrak{C}^{[A,B]} := B \backslash A$. Up to graph-isomorphism even $\dim \mathfrak{C}^{[A,B]} := |B \backslash A|$, the *dimension*, determines the cube. For most of the time we work with cubes $\mathfrak{C}^B := \mathfrak{C}^{[\emptyset,B]}$, i.e. power sets. In cases where $\operatorname{carr} \mathfrak{C}$ does not play any role we even abandon the superscript and write $\mathfrak{C}$.

The additional parameter A is helpful in naming the subcubes of a cube. A *subcube* of a cube $\mathfrak{C}^{[A,B]}$ is a cube $\mathfrak{C}^{[X,Y]}$ with $A \subseteq X \subseteq Y \subseteq B$. These subcubes correspond to the *faces* of a geometric realization of $\mathfrak{C}^{[A,B]}$.

Instead of writing X and Y it is often more convenient to describe a subcube by a vertex and a set of labels generating it: we write $\mathfrak{C}(v, L)$ for the smallest subcube of $\mathfrak{C}$ containing v and $v \oplus L$. In fact, for $X = v \cap \bar{L}$ and $Y = v \cup L$ we have $\mathfrak{C}(v, L) = \mathfrak{C}^{[X,Y]}$, where $\bar{L}$ denotes the complement of L with respect to $\operatorname{carr} \mathfrak{C}$.

Given an orientation of a cube (i.e. an orientation of its edges), a vertex is called a *sink* if all its incident edges are incoming. An orientation ψ of $\mathfrak{C}$ is called a *unique sink orientation* or USO if ψ restricted to any subcube has a unique sink. We usually do not distinguish between a USO and the cube equipped with that USO, and refer to the cube as USO as well. An orientation is called *acyclic* if it contains no directed cycle. Acyclic unique sink orientations are abbreviated by AUSO.

The Problem. Easily stated: find the sink of a USO. Following [18] we assume that the orientation is given implicitly, i.e. we can access an *arbitrary* vertex of the USO through an oracle, which then reveals the orientation of the edges incident to the requested vertex. This basic operation is called *vertex evaluation* (sometimes we refer to it as *step* or *query*), and we are interested in evaluating the (unique) sink of a cube by as few vertex evaluations as possible. Formally, let $\operatorname{eval}(\mathbf{A}, \psi)$ be the number of vertex evaluations it takes for a deterministic algorithm $\mathbf{A}$ to evaluate the sink of a USO (or an AUSO) ψ, and define $t(n)$ (or $t_{\text{acyc}}(n)$) to be $\min_{\mathbf{A}} \max_{\psi} \operatorname{eval}(\mathbf{A}, \psi)$. Obviously, $t(n) \geq t_{\text{acyc}}(n)$. Let $\tilde{t}(n)$ and $\tilde{t}_{\text{acyc}}(n)$ be the corresponding functions for randomized algorithms; the expected number of evaluations a fastest randomized algorithm takes until it finds the sink of any USO (or AUSO).

USOs provide a common framework for several seemingly different problems. Related abstractions were considered earlier, mainly to deal with geometric problems. LP-type problems, Abstract Objective Functions and Abstract Optimization Problems [3], [9]–[12], [14] have a rich literature and provide the fastest known algorithms for several geometric problems in the *unit cost model*. As is always the case with abstractions, it is very well possible that the model of unique sink orientations is in fact too general to be of any use, but there are a number of results suggesting otherwise.

At first it is not even clear how to find the sink of a USO in $o(2^n)$ evaluations; whether USOs have enough structure, which distinguishes them from just any orientation of the

cube. Indeed, in order to find the sink in an *arbitrary* orientation (with a sink), one needs at least $2^{n-1}+1$ vertex evaluations [19].

In [18] a deterministic algorithm running in 1.61^n evaluations and a randomized algorithm running in 1.44^n evaluations (using that $\tilde{t}(3) = 4074633/1369468$ [16]) were given for evaluating the sink of a USO. These algorithms indicate that USOs have some, actually quite rich, structure. Another result, quantitatively pointing to this direction, is due to Matoušek [13]; he showed that the number of USOs is $2^{\Theta(\log n 2^n)}$, which is *significantly less* than $2^{n2^{n-1}}$, the number of all orientations. A bonus for optimists is that within the class of orientations providing the matching lower bound for Matoušek's upper bound the sink can be found in *five*(!) steps.

Motivation. The most obvious and actually quite widely investigated appearance of USOs is the special case of linear programming; i.e. when the polyhedron in question is a slanted geometric cube. Then a linear objective function (in general position) canonically defines an AUSO on the cube and finding the sink of this orientation obviously corresponds to finding the minimum of the objective function on the polyhedron. The importance of this question is well demonstrated by the number of papers investigating the running time of specific randomized simplex-like (edge-following) algorithms, like `RandomEdge` or `RandomFacet`, on (sometimes even specific) acyclic orientations. The `RandomEdge` algorithm proceeds on a directed path by choosing uniformly at random among the outgoing edges at each vertex. The `RandomFacet` algorithm chooses a random facet of the cube where the smaller-dimensional algorithm is run and after finding the sink of the facet (and in case that sink is not the global sink) it proceeds to the antipodal facet to find its sink by another smaller-dimensional `RandomFacet` algorithm. Gärtner [4],[5] showed that the expected number of evaluations it takes for the `RandomFacet` algorithm to evaluate the sink of an AUSO is at most $e^{2\sqrt{n}}$. Gärtner et al. [7] analyzed the behavior of `RandomEdge` and `RandomFacet` on a specific, particularly interesting orientation, the so-called Klee–Minty cubes.

USOs also appear in less obvious ways, where the correspondence between the problem and the orientation is more abstract. Stickney and Watson [17] defined a combinatorial correspondence between the solution of certain linear complementarity problems [2] (defined by P-matrices) and finding the sink in an appropriate USO. In this correspondence the appearance of cycles in the USO is possible.

Similarly, certain quadratic optimization problems, like finding the smallest enclosing ball of $d+1$ affinely independent points in the Euclidean d-space can be reduced to finding the sink in an appropriate USO. For each such point set, there is a corresponding USO, where the sink corresponds to the smallest enclosing ball of the point set [8]. Here again cycles can arise. The general problem of n points in d-space is known to be reducible to the case when n is small (i.e. around d^2) compared with d. Then one can place the ambient d-space into $\mathbb{R}^{n-1}$ and perturb the points for affine independence.

Finally, Gärtner [6] showed that *every* linear program with n variables and m constraints can be translated into a unique sink orientation of dimension $2(n+m)$ via certain convex programs. Through this detour unique sink orientations can be seen as an abstraction of linear programming in general (not only for cubelike polyhedra).

Results. The main finding of our paper is a lower bound of order $n^2/\log n$ for the number of evaluations a deterministic algorithm needs in order to find the sink of an n-dimensional AUSO. Our result is the first nonlinear lower bound for the function $t(n)$. On the way we organize our current knowledge about producing USOs. We also prove better lower bounds for small values of n. To motivate our results we look at a couple of simpler classes of orientations.

Lower bounds were found earlier for several related problems or special cases. Aldous [1] considered orientations given canonically by an ordering of the vertices of the cube, which have a unique sink (but not necessarily a unique sink on every subcube). These orientations are acyclic by definition. For them he proves an exponential lower bound of $\sqrt{2}^{n(1+o(1))}$ on the running time of randomized algorithms, using the hitting times of a random walk on the cube. He also provides a simple randomized algorithm which is essentially best possible. Since his model is in some sense weaker and in some sense stronger than our USOs, his results do not imply anything for our problem.

Other known lower bounds are related to specific randomized simplex-like algorithms, for example, `RandomEdge` or `RandomFacet`. On AUSOs Matoušek [12] proved an $e^{\Omega(\sqrt{n})}$ lower bound for the expected running time of `RandomFacet`, which nicely complements the $e^{O(\sqrt{n})}$ upper bound of Gärtner [4], [5]. Gärtner et al. [7] showed that the expected running time of `RandomEdge` on the Klee-Minty cubes is $\Omega(n^2/\log n)$. A negative result of Morris [15] could also be interpreted as a lower bound for general USOs. He constructs USOs on which the expected running time of `RandomEdge` is $((n-1)/2)!$, more than the number of vertices.

Our paper is organized as follows. In Section 2 we collect notations, definitions and several known facts about USOs that we will make use of. In Section 3 we establish our basic building blocks: two different ways of obtaining new USOs from old ones. In Lemma 3 we define a certain product construction of USOs, while in Lemma 5 we describe circumstances under which a local change in an existing USO produces another USO. In order to motivate our lower bound on the general problem, in Section 4 we discuss two important smaller classes of USOs for which very fast algorithms exist. *Decomposable orientations* are built recursively by taking two decomposable orientations of dimension $n-1$ on two disjoint facets of an n-cube and orient all edges between them in the same direction. Williamson Hoke [20] gave a $O(n^2)$ simplex-like algorithm for them. In Proposition 7 we observe that our extra power of being able to "jump around" lets us find the sink in only $n+1$ evaluations. We also observe that this is best possible; i.e. on the class of decomposable orientations we know *a* fastest algorithm. In Proposition 8 we consider the class of the so-called *matching-flip* orientations. This class was introduced by Matoušek and Wagner to give a lower bound for the number of all USOs. Matoušek [13] proved that the number of all USOs is $2^{O(\log n 2^n)}$. The class of matching-flip orientations is in fact so rich that it provides a lower bound with the same order of magnitude in the exponent. Thus it is somewhat surprising that for this class there is an algorithm finding the sink in five steps.

In Section 5 we provide a strategy which forces every deterministic algorithm to at least $\Omega(n^2/\log n)$ evaluations while finding the unique sink of an n-dimensional cube with an AUSO. We also give a matching lower bound for the four-dimensional algorithm of [18].

In Section 6 we list several open problems.

2. Preliminaries

For an orientation ψ of a cube $\mathfrak{C}^{[A,B]}$ the *outmap* s_ψ *of* ψ is the map assigning to every vertex v the set of labels of outgoing edges, i.e. $s_\psi\colon \mathrm{V}(\mathfrak{C}) \to 2^{\operatorname{carr}\mathfrak{C}}$ with

$$s_\psi(v) = \{l \mid \{v, v \oplus \{l\}\} \text{ is outgoing from } v\}.$$

Obviously, a vertex v_0 is a sink iff $s_\psi(v_0) = \emptyset$.

An important property of outmaps of USOs was proved in [18].

Lemma 1 [18, Lemma 2.2]. *Let ψ be a USO. Then s_ψ is a bijection.*

Its proof relies on a simple observation, which is also a tool to produce new USOs from old ones. If ψ is an orientation of $\mathfrak{C}$ and $L \subseteq \operatorname{carr}\mathfrak{C}$, then let $\psi^{(L)}$ be the orientation of $\mathfrak{C}$ which agrees with ψ on $\bar{L}$-labeled edges and differs on L-labeled edges.

Lemma 2 [18, Lemma 2.1]. *For any $L \subseteq \operatorname{carr}\mathfrak{C}$, if ψ is a USO, then $\psi^{(L)}$ is a USO as well.*

We remark here that acyclicity does not necessarily survive this ''label-flip''; there are examples of AUSOs ψ and labels $l \in \operatorname{carr}\mathfrak{C}$ (already for $\dim\mathfrak{C} = 3$), such that $\psi^{(\{l\})}$ is *not* an AUSO.

In Lemma 2.3 of [18] outmaps of USOs were characterized; $s\colon \mathrm{V}(\mathfrak{C}) \to 2^{\operatorname{carr}\mathfrak{C}}$ is the outmap of a USO, iff

$$(s(v) \oplus s(w)) \cap (v \oplus w) \neq \emptyset \qquad \text{for all} \quad v \neq w \in \mathrm{V}(\mathfrak{C}).$$

We call such outmaps *unique sink outmaps* and—as the acronyms conveniently coincide—we abbreviate them by USO as well. This usually does not cause any confusion since orientations and outmaps determine each other uniquely.

Note that for cubes $\mathfrak{C}^B$ every outmap is a permutation of 2^B. An important example of a unique sink outmap on every $\mathfrak{C}^B$ is the identity. More generally, for every $w \in \mathrm{V}(\mathfrak{C})$ the map ι_w on $\mathrm{V}(\mathfrak{C})$ defined by $\iota_w(v) := v \oplus w$ is a USO (note that $\iota_\emptyset$ is the identity). The corresponding orientation directs every edge towards w. We refer to such orientations as *uniform orientations*. Note that the sink of ι_w is w and the source is $\bar{w} = B \setminus w$.

3. Basic Constructions

Take a USO and replace every vertex by a USO cube of a fixed dimension. This construction defines a *product structure* for USOs. Figure 1 tries to illustrate that: in a three-dimensional USO the vertices are replaced by two-dimensional USOs.

Figure 1 suggests thinking about the product construction as a bunch of identical frame-orientations glued together at the hypervertices by arbitrary USOs. As an alternative, one can also visualize the construction by taking a bunch of different USOs on the

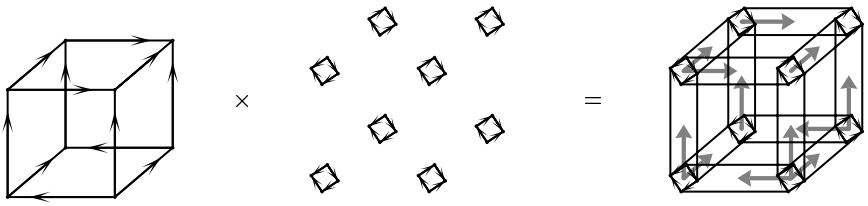

Fig. 1. A product of a three-dimensional USO with two-dimensional ones.

frame glued together by identical USOs in the hypervertices. In the proof of our main result, both approaches are helpful.

The statement of the next lemma provides the formal definition of this product construction, and shows that it indeed produces a USO.

Lemma 3. *Let A be a set of labels, $B \subseteq A$ and $\bar{B} = A \setminus B$. For a USO $\tilde{s}$ on $\mathfrak{C}^B$ and for USOs s_u on $\mathfrak{C}^{\bar{B}}$, $u \in \mathrm{V}(\mathfrak{C}^B)$, the map s on $\mathfrak{C}^A$ defined by*

$$s(v) = \tilde{s}(v \cap B) \cup s_{v \cap B}(v \cap \bar{B})$$

is a USO. Furthermore, if $\tilde{s}$ and all s_u are acyclic, then so is s.

Proof. Let $L \subseteq A$ be a subset of the label set and let $v \in \mathrm{V}(\mathfrak{C}^A)$ be an arbitrary vertex. We show that the subcube $\mathfrak{C}(v, L)$ containing v and generated by the set of labels L has exactly one sink. We note that a vertex $w \in \mathrm{V}(\mathfrak{C}(v, L))$ is a sink of the subcube iff $s(w) \cap L = \emptyset$. Therefore, by definition of s, w is a sink of $\mathfrak{C}(v, L)$ iff

1. $\tilde{s}(w \cap B) \cap L = \emptyset$, i.e. $w \cap B$ is the sink of the subcube $\mathfrak{C}(w \cap B, B \cap L)$ with respect to $\tilde{s}$; and
2. $s_{w \cap B}(w \cap \bar{B}) \cap L = \emptyset$, i.e. $w \cap \bar{B}$ is the sink of the subcube $\mathfrak{C}(w \cap \bar{B}, \bar{B} \cap L)$ with respect to $s_{w \cap B}$.

Since $\tilde{s}$ is a unique sink outmap, there is a unique sink u_1 of the subcube $\mathfrak{C}(v \cap B, B \cap L) = \mathfrak{C}(w \cap B, B \cap L)$. Thus $u_1 = w \cap B$ is determined uniquely. Also, s_{u_1} has a unique sink $u_2 = w \cap \bar{B}$ on the subcube $\mathfrak{C}(w \cap \bar{B}, \bar{B} \cap L) = \mathfrak{C}(v \cap \bar{B}, \bar{B} \cap L)$. Therefore $w = u_1 \cup u_2$ is the unique sink of $\mathfrak{C}(v, L)$.

For the second part of the lemma note that a path $v_1, \ldots, v_k$ in $\mathfrak{C}^A$ with respect to s induces a walk $v_1 \cap B, \ldots, v_k \cap B$ in $\mathfrak{C}^B$with respect to $\tilde{s}$. If $v_1, \ldots, v_k$ would form a cycle, i.e. $v_1 = v_k$, then the induced walk $v_1 \cap B, \ldots, v_k \cap B$ either contains a cycle in $\tilde{s}$ or $v_1 \cap B = v_2 \cap B = \cdots = v_k \cap B = u$. Since $\tilde{s}$ is acyclic, we must have the second case. However, then $v_1 \cap \bar{B}, \ldots, v_k \cap \bar{B}$ forms a cycle in $\mathfrak{C}^{\bar{B}}$ with respect to s_u. This is a contradiction to s_u being acyclic, i.e. s has to be acyclic. □

For $|B| = 1$, Lemma 3 says that two $(n-1)$-dimensional USOs can be combined to an n-dimensional one by placing them in two disjoint facets and directing all edges in between in the same direction.

The other extreme case $|\bar{B}| = 1$ shows that if a cube contains two opposite facets with the same $(n-1)$-dimensional USO, the edges between these facets can be directed arbitrarily.

It is easy to see that we can direct all edges of one label arbitrarily only if the USOs in the two facets not containing edges with this label are the same. On the other hand, by placing a USO in one facet and the orientation which has all edges flipped in the other facet, we are forced to direct all edges in between in the same direction. Therefore Lemma 3 is best possible in some sense, one cannot expect to get a more general product construction without further exploring $\tilde{s}$ and/or some s_u.

A simple consequence of Lemma 3 is that in an AUSO the sink and source could be placed anywhere, independently of each other. This fact is stated in the next corollary and will be utilized in the proof of our main result.

Corollary 4. *For any two distinct vertices $u, v \in \mathrm{V}(\mathfrak{C}^A)$, there is an acyclic unique sink outmap with sink u and source v.*

Proof. Choose a label $l \in u \oplus v$ and set $B = \{l\}$. Without loss of generality $l \in v$. Choose two uniform USOs, ι_u and $\iota_{\bar{v}}$, on $\mathfrak{C}^{[n]\setminus\{l\}}$ with sink u and source $v\setminus\{l\}$, respectively. Let $\tilde{s}$ be the uniform outmap $\iota_\emptyset$ on $\mathfrak{C}^{\{l\}}$ with sink $\emptyset$. As uniform orientations are acyclic, the application of Lemma 3 yields an acyclic unique sink outmap with sink u and source v. □

Another way to construct new USOs is by local modification. Given an n-dimensional USO one can replace the orientation of a subcube under certain conditions. It is clear that the replacing orientation has to be a USO, but that does not suffice.

Lemma 5. *Let s be a unique sink outmap on a cube $\mathfrak{C}^A$ and let $\mathfrak{C}_0$ be a subcube with label set $B \subseteq A$. If $s(v) \cap \bar{B} = \emptyset$ for all $v \in \mathrm{V}(\mathfrak{C}_0)$ and s_0 is a unique sink outmap on $\mathfrak{C}^B$, then the map $s' : 2^A \to 2^A$ defined by $s'(v) = s_0(v \cap B)$ for $v \in \mathrm{V}(\mathfrak{C}_0)$ and $s'(v) = s(v)$ otherwise is a unique sink outmap on $\mathfrak{C}^A$.*

If s and s_0 are acyclic, then s' is acyclic as well.

For example, in Fig. 2 the lower three-dimensional subcube can be directed arbitrarily, since all edges incident to it are incoming.

Proof. Let $v \in \mathrm{V}(\mathfrak{C}^A)$ be a vertex and let $L \subseteq A$ be a set of labels. We show that the subcube $\mathfrak{C}(v, L)$ has exactly one sink with respect to s'.

If the subcubes $\mathfrak{C}(v, L)$ and $\mathfrak{C}_0$ have an empty intersection, then s' induces the same orientation on $\mathfrak{C}(v, L)$ as s does. Therefore $\mathfrak{C}(v, L)$ has a unique sink.

If $\mathfrak{C}(v, L)$ and $\mathfrak{C}_0$ *do* intersect, then the intersection is a subcube $\mathfrak{C}'$ of $\mathfrak{C}_0$ and thus has a unique sink u_0 with respect to s_0. We claim u_0 is the unique sink of $\mathfrak{C}(v, L)$ as well. It is clear that u_0 is a sink since $(s'(u_0) \cap L)\setminus B = \emptyset$ as u_0 is in $\mathfrak{C}_0$, and $(s'(u_0) \cap L) \cap B = \emptyset$ since u_0 is the sink of $\mathfrak{C}' = \mathfrak{C}(u_0, L \cap B)$.

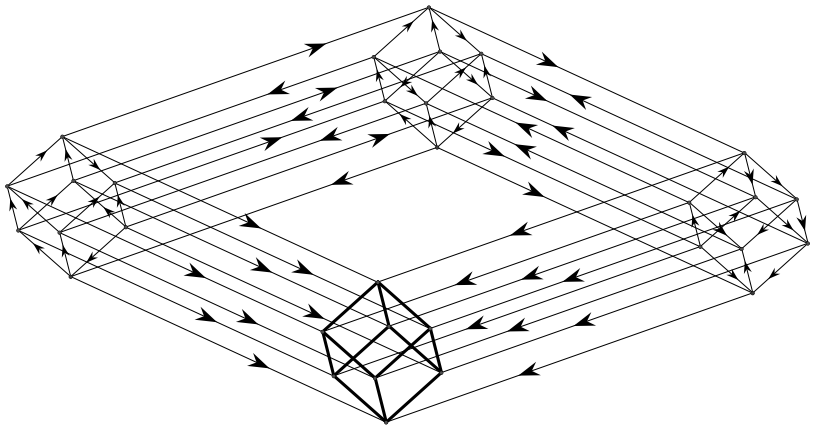

Fig. 2. Three-dimensional flippable subcube.

On the other hand, no $w \in \mathrm{V}(\mathfrak{C}(v, L)) \setminus \mathrm{V}(\mathfrak{C}_0)$ is the sink of $\mathfrak{C}(v, L)$ according to s, and since $s(w) = s'(w)$, they do not become sinks in s' either. Thus the sink of $\mathfrak{C}(v, L)$ lies in $\mathfrak{C}_0$, where it is unique.

Now let s and s_0 be acyclic. Note that no directed path in $\mathfrak{C}^A$ can leave $\mathfrak{C}_0$. Therefore a potential cycle of s' is contained either in $\mathfrak{C}_0$ and therefore is a cycle with respect to s_0 or in $\mathfrak{C}^A \setminus \mathfrak{C}_0$, when it is a cycle with respect to s. Consequently, if s_0 and s are acyclic, s' is acyclic as well. □

Corollary 6. *Let s be a unique sink outmap on a cube $\mathfrak{C}^A$ and let $\mathfrak{C}_0$ be a subcube with label set $B \subseteq A$. Suppose that $s(v) \cap \bar{B} = s(v') \cap \bar{B}$ for all $v, v' \in \mathrm{V}(\mathfrak{C}_0)$. If s_0 is a unique sink outmap on $\mathfrak{C}^B$, then the map $s' : 2^A \to 2^A$ defined by $s'(v) = s_0(v \cap B) \cup (s(v) \cap \bar{B})$ for $v \in \mathrm{V}(\mathfrak{C}_0)$ and $s'(v) = s(v)$ otherwise is a unique sink outmap on $\mathfrak{C}^A$.*

Proof. Apply Lemma 2 for $L = s(v) \cap \bar{B}$ ($v \in \mathrm{V}(\mathfrak{C}_0)$). The obtained outmap $s^{(L)}$ is thus a USO with all $\bar{B}$-labeled edges incoming into $\mathfrak{C}_0$. Thus Lemma 5 applies and the change of $s^{(L)}$ on $\mathfrak{C}_0$ to s_0 results in a USO. Now switching all L-labeled edges back we obtain s', which is therefore a USO. □

We remark that unlike in Lemma 5, here acyclicity does not necessarily carry over to s'.

Again, the special case $|B| = 1$ is of some interest. In this scenario $\mathfrak{C}_0$ is a single edge and by the preceding corollary, we see that an edge can be flipped, if the outmaps of the two adjacent vertices only differ in the label of the edge. Since flipping an edge solely affects the outmap of the adjacent vertices the converse also holds. Therefore an edge $\{v, w\}$ is flippable iff $s(v) \oplus s(w) = v \oplus w$.

In particular, if ι_w is the outmap of a uniform orientation, then one can flip the edges of an arbitrary matching of the cube and the result is still a USO. As we shall see in the next section this construction is particularly interesting.

4. Interlude: Two Simple Classes

Recall that in the special case $|B| = 1$ of Lemma 3 we obtained an n-dimensional USO out of two (possibly distinct) $(n-1)$-dimensional USOs. The new orientation is special in the sense that all edges of one label point in the same direction. This leads us to the notion of combed orientations; an orientation is called *combed* if there is a label l for which all l-labeled edges are oriented towards the same facet.

Iterating the construction we get the class of *decomposable* USOs. Let $\mathcal{D}_1$ be the class of all one-dimensional USOs and denote by $\mathcal{D}_{n+1}$ the class of all orientations constructed from two orientations from $\mathcal{D}_n$ using Lemma 3. We call $\mathcal{D}_n$ the class of decomposable USOs in dimension n. Equivalently, an orientation is decomposable iff every nonzero-dimensional subcube is combed.

In general, it is not easy to check whether a USO is decomposable or even combed (one has to to evaluate half of the vertices), but in the class of decomposable USOs it is easy to find the sink.

Proposition 7. *For a decomposable USO of dimension n one needs at most $n+1$ vertex evaluations to evaluate the sink. Moreover, $n+1$ is best possible; i.e. for every deterministic algorithm there is a decomposable USO on which the algorithm needs at least $n+1$ evaluations.*

Proof. The following algorithm works in at most $n+1$ evaluations. Start by evaluating an arbitrary vertex v_0, then perform the following procedure. For any i, if $s(v_i) = \emptyset$, then stop, v_i is the sink. Otherwise set $v_{i+1} = v_i \oplus s(v_i)$ and repeat.

We show by induction that for each i the sink u of the orientation and v_i are both in the same subcube $\mathfrak{C}(u, L_i)$, where $|L_i| = n - i$ and $s(v_i) \subseteq L_i$. This implies that v_n is the sink itself.

The claim is definitely true for $i = 0$ with $L_0 = A$, the entire label set of the cube. Suppose now that it is true for i. Since the orientation is decomposable it is combed on $\mathfrak{C}(u, L_i)$ and splits it into two facets $\mathfrak{C}_1, \mathfrak{C}_2$ along a label l, with all edges between $\mathfrak{C}_1$ and $\mathfrak{C}_2$ oriented in the same direction. We set $L_{i+1} = L_i \setminus \{l\}$. Since $s(v_i) \subseteq L_i$, $v_{i+1} = v_i \oplus s(v_i)$ is in $\mathfrak{C}(u, L_i)$ as well.

We assume that $u \in \mathfrak{C}_1$. If $v_i \notin \mathfrak{C}_1$, then the l-labeled edge incident to v_i is outgoing. Thus $v_{i+1} = v_i \oplus s(v_i) \in \mathfrak{C}_1$.

If $v_i \in \mathfrak{C}_1$, then the l-labeled edge incident to v_i is incoming, i.e. $l \notin s(v_i)$. Thus v_{i+1} stays in $\mathfrak{C}_1$.

In both cases v_{i+1} is in the same facet as u, thus $v_{i+1} \in \mathfrak{C}(u, L_{i+1})$ and $s(v_{i+1}) \subseteq L_{i+1}$.

For the other direction, suppose an algorithm first requests the vertex u_0. We (the oracle) return the source, $s(u_0) = A$. Let u_1 be the second request and let $l \in u_0 \oplus u_1$. Thus u_1 is in the other facet $\mathfrak{C}'$ with respect to label l as u_0. We reveal to the algorithm that the first combed label is l, thus the sink is in $\mathfrak{C}'$. There we follow a strategy (existing by induction) which forces the algorithm to n evaluations in $\mathfrak{C}'$. That means $n+1$ evaluations altogether. □

Note that when applied to nondecomposable orientations, the algorithm in the proof does not even need to terminate.

Our second example in this section arises from Corollary 6. Consider a uniform orientation and a matching of the cube. According to Corollary 6 every edge in our matching is flippable. Since flipping one of them does not affect the flippability of other edges of the matching, we can flip any number of them simultaneously.

In consequence, for every matching and every uniform orientation we obtain different USOs. These "matching-flip" USOs were first constructed by Matoušek and Wagner. By counting perfect matchings of the hypercube one obtains a very good lower bound for the number of USOs, which has the same order of magnitude in the logarithm as the number of all USOs.

Thus it is somewhat surprising (and encouraging for the general problem) that it is very easy to find the sink of an orientation from this very large class.

Proposition 8. *For a USO coming from a uniform orientation by flipping a matching one needs at most five steps to find the sink. The value five here is best possible.*

Proof. Let s be a unique sink outmap coming from a uniform orientation ι_u by flipping a matching. Then at every vertex s differs from ι_u in at most one label.

If we knew the sink u of ι_u, then we would find the sink of s with at most two queries. Either it is u or there is one flipped edge at u, when the neighbor along this edge is the sink. In other words the sink is either u or $u \oplus s(u)$.

First we evaluate an arbitrary vertex v_1. Our second inquiry, the vertex $v_2 = v_1 \oplus s(v_1)$ has distance at most one from u, since $s(v_1)$ and $\iota_u(v_1) = u \oplus v_1$ differ in at most one label. Therefore $\iota_u(v_2)$ has at most one element and $|s(v_2)| \leq 2$.

If $|s(v_2)| = 0$ we have found the sink in two steps.

For $|s(v_2)| = 1$, we ask $v_3 = v_2 \oplus s(v_2)$ next. If v_2 was u, v_3 has to be the sink. If not, $v_3 = u$ and either v_3 is the sink or $v_4 = v_3 \oplus s(v_3)$. Therefore after at most four queries we found the sink.

For $|s(v_2)| = 2$ we know that v_2 is a neighbor of u and so is $v_3 = v_2 \oplus s(v_2)$. Either v_3 is the sink or $v_4 = v_3 \oplus (s(v_3) \cap s(v_2))$ is the original u and we need one more query to find the sink. This makes five queries altogether.

We leave the optimality of five to the reader. □

5. The Strategy

In this section we prove the main result of the paper. In order to show lower bounds on $t(n)$ we consider ourselves the oracle playing against a deterministic algorithm, we call Al. We try to make sure that our answers to Al's evaluation requests (i) force Al to evaluate many vertices before the sink, and (ii) could be extended to an AUSO of the whole cube.

Given a sink-finding algorithm Al, we construct an AUSO for which Al needs an almost-quadratic number of queries. For the first $n - \lceil \log_2 n \rceil$ inquiries we maintain a partial outmap $s\colon\ W \to \operatorname{carr} \mathfrak{C}^{[n]}$ on the set $W \subseteq \mathrm{V}(\mathfrak{C}^{[n]})$ of queried vertices, containing the answers we gave Al so far. We also maintain a set L of labels and an acyclic unique sink outmap $\tilde{s}$ on $\mathfrak{C}^L$. This smaller-dimensional outmap $\tilde{s}$ is our "building block" which enables us to extend our answers to a global USO at any time.

Before the first inquiry we set $L = W = \emptyset$. After each inquiry we answer Al by revealing the value of the outmap s at the requested vertex. Then we update L and $\tilde{s}$, such that the following conditions hold:

(a) $|L| \leq |W|$,
(b) $w' \cap L \neq w'' \cap L$ for every $w' \neq w'' \in W$ and
(c) $s(w) = \tilde{s}(w \cap L) \cup \bar{L}$ for every $w \in W$.

Informally, condition (b) means that the projections of the queried vertices to $\mathfrak{C}^L$ are all distinct. This we shall achieve by occasionally adding a label to L if the condition would be violated. Condition (c) exhibits two properties of our answers to the inquiries of Al. First that our answers are consistent with $\tilde{s}$ on L-labeled edges, and then that all $\bar{L}$-labeled edges, i.e. the edges leaving $\mathfrak{C}(w, L)$, are outgoing.

Suppose now that Al requests the evaluation of the next vertex u. We can assume that $u \notin W$. Depending on whether there is a $w \in W$ with $u \cap L = w \cap L$ or not we have to distinguish two cases.

Case 1: *For every* $w \in W$ *we have* $w \cap L \neq u \cap L$. Then we answer $s(u) = \tilde{s}(u \cap L) \cup \bar{L}$ and leave L and $\tilde{s}$ unchanged. (a)-(c) all hold trivially by the definition of the updates and the assumption of Case 1.

Case 2: *There is a* $v \in W$, *such that* $u \cap L = v \cap L$. Let us immediately note that by condition (b), there is exactly one such $v \in W$. The assumption of Case 2 and $u \neq v$ implies that we can fix a label $l \in \bar{L}$ such that $l \in u \oplus v$.

We answer $s(u) = s(v) \backslash \{l\}$. Note that as $l \notin L$, $l \in s(v)$.

Now we have to update L and $\tilde{s}$. We get our new L by adding l. To define the new orientation $\tilde{s}$ we take two copies of the old $\tilde{s}$ on the two facets determined by l. Then, by Lemma 3, we can define the orientation of the edges going across arbitrarily, so we make them such that condition (c) is satisfied. More formally, let

$$\tilde{s}(z) = \begin{cases} \tilde{s}(v \cap L) & \text{if} \quad z = u \cap (L \cup \{l\}), \\ \tilde{s}(w \cap L) \cup \{l\} & \text{if} \quad z = w \cap (L \cup \{l\}), \\ & \text{for some} \quad w \in W, \quad w \neq u, \\ \tilde{s}(z) \cup (z \cap \{l\}) & \text{otherwise.} \end{cases}$$

Next we have to check conditions (a)–(c) for our new W, L and $\tilde{s}$. Condition (a) still holds, because we added one element to each of W and L. Since L just got larger, we only need to check condition (b) for the pairs of vertices containing u. By the uniqueness of v, it is actually enough to check (b) for the pair u, v. Since l was chosen from $u \oplus v$ and is now included in L, $u \cap L \neq v \cap L$. Condition (c) is straightforward from the definitions.

We proceed until $|W| = n - \lceil \log_2 n \rceil$, and then change the strategy. By condition (a), $|L| \leq n - \lceil \log_2 n \rceil$. We choose an arbitrary superset $L' \supseteq L$ of L such that $|L'| = n - \lceil \log_2 n \rceil$. The set of labels $\bar{L}'$ determines at least $2^{|\bar{L}'|} \geq n$ disjoint subcubes generated by L'. As $|W| < n$, we can select one of them, say $\mathfrak{C}_0$, which does not contain any point evaluated so far.

Our plan is to apply Lemma 5 with $\mathfrak{C}_0$ and an orientation s, which is consistent with the outmaps of the vertices evaluated so far. The lemma will then enable us to reveal s on $\mathrm{V}(\mathfrak{C}^{[n]}) \backslash \mathrm{V}(\mathfrak{C}_0)$ to Al and still be able to start a completely "new game" on a cube of

relatively large dimension. To construct s satisfying the conditions of Lemma 5 we use Lemma 3 twice.

First we define an orientation $\bar{s}$ of $\mathfrak{C}^{L'}$ using Lemma 3 with $A = L'$, $B = L$, $\tilde{s}$ as defined above, and outmaps s_u on $\mathfrak{C}^{L'\setminus L}$ with the property that for every $w \in W$ the map $s_{w\cap L}$ has its source at $w \cap (L'\setminus L)$. This last requirement can be fulfilled because of condition (b).

Thus the resulting outmap $\bar{s}$ is consistent with the evaluated vertices in the sense that $s(w) \cap L' = \bar{s}(w \cap L')$ for each $w \in W$.

Next we apply Lemma 3 again with $B = L'$, so we have to construct USOs s_u of $\mathfrak{C}^{\bar{L}'}$ for every $u \in \mathrm{V}(\mathfrak{C}^{L'})$. In doing so, we only take care that the sinks of all these orientations are in $\mathfrak{C}_0$, and if $u = w \cap L'$ for some $w \in W$ then $w \cap \bar{L}'$ is the source of s_u. (By condition (b) there can be at most one such vertex w for each u.) The appropriate s_u is constructed in Corollary 4. Now Lemma 3, applied with L', $\bar{s}$ and these s_u's, provides us with an orientation s which agrees with our answers given for the evaluated vertices, and all $\bar{L}'$-labeled edges are incoming into $\mathfrak{C}_0$.

By Lemma 5 we can reveal s on $\mathrm{V}(\mathfrak{C}^{[n]})\setminus\mathrm{V}(\mathfrak{C}_0)$ to Al, and still be able to place *any* orientation on $\mathfrak{C}_0$, a subcube of dimension $n - \lceil\log_2 n\rceil$. Therefore we just proved

$$t_{\mathrm{acyc}}(n) \geq n - \lceil\log_2 n\rceil + t_{\mathrm{acyc}}(n - \lceil\log_2 n\rceil).$$

Theorem 9. *A deterministic algorithm needs $\Omega(n^2/\log n)$ many steps to find the sink of an AUSO on an n-dimensional cube.*

Proof. We prove by induction for $n \geq 2$,

$$t_{\mathrm{acyc}}(n) \geq \frac{n^2}{2\lceil\log_2 n\rceil} - \frac{n}{2}.$$

We do not make an attempt to optimize the constants.

First note that $t_{\mathrm{acyc}}(2) \geq 1 = \frac{4}{2} - \frac{2}{2}$ and $t_{\mathrm{acyc}}(3) \geq 1 \geq \frac{9}{4} - \frac{3}{2}$, i.e. for $n = 2, 3$ the inequality holds.

Now let $n \geq 4$, such that for $2 \leq k < n$ we have $t_{\mathrm{acyc}}(k) \geq k^2/2\lceil\log_2 k\rceil - k/2$. Since $n - \lceil\log_2 n\rceil \geq 2$ we get

$$\begin{aligned}
t_{\mathrm{acyc}}(n) &\geq n - \lceil\log_2 n\rceil + t_{\mathrm{acyc}}(n - \lceil\log_2 n\rceil) \\
&\geq n - \lceil\log_2 n\rceil + \frac{(n - \lceil\log_2 n\rceil)^2}{2\lceil\log_2(n - \lceil\log_2 n\rceil)\rceil} - \tfrac{1}{2}(n - \lceil\log_2 n\rceil) \\
&\geq \tfrac{1}{2}n - \tfrac{1}{2}\lceil\log_2 n\rceil + \frac{n^2 - 2n\lceil\log_2 n\rceil + \lceil\log_2 n\rceil^2}{2\lceil\log_2 n\rceil} \\
&= \frac{n^2}{2\lceil\log_2 n\rceil} - \frac{n}{2}
\end{aligned}$$

and the inequality also holds for n. □

For small dimensions $t(n)$ is easy to check; $t(0) = 1$, $t(1) = 2$, $t(2) = 3$ and $t(3) = 5$. Here we give a lower bound complementing the `SevenStepsToHeaven`

algorithm of [18] to prove $t(4) = 7$. During the course of the argument we also show $t(2) = 3$ and $t(3) = 5$ in a stronger sense.

Proposition 10. *Every deterministic algorithm needs three queries to find the sink of a two-dimensional USO, even if there is a fixed vertex which is known not be the sink.*

Proof. The strategy answers with the source for the first query of Al. Then all remaining three unqueried vertices are still potential sinks. Even if Al has a knowledge that one of them is not the sink, there are two possibilities left. That is no deterministic algorithm can evaluate the sink in two steps. □

Proposition 11. *Every deterministic algorithm needs five queries to find the sink of a three-dimensional USO, even if it is known that the vertex antipodal to its first query is* not *the sink.*

Proof. For the first query u_1 of Al we answer with the source. By our assumption the orientation we are about to construct cannot have the antipodal $\overline{u_1}$ as its sink.

There are two cases according to the second query u_2 of Al.

Case 1: *u_1 and u_2 are not antipodal.* Then there is a facet $\mathfrak{C}'$ of the cube containing both u_1 and u_2. We construct a combed USO by placing an arbitrary two-dimensional USO on $\mathfrak{C}'$ with source u_1 and directing all edges *outward* from $\mathfrak{C}'$ into the antipodal facet $\mathfrak{C}''$. Then, in order to evaluate the sink of $\mathfrak{C}$, Al needs to find the sink of $\mathfrak{C}''$. For that, by the previous proposition, Al needs three more evaluations even though he knows that $u_1 \oplus [3]$ is not the sink.

Case 2: *u_1 and u_2 are antipodal.* By evaluating the vertex which is known not to be the sink, Al loses its edge and the problem reduces to a usual three-dimensional problem. (A strategy forcing five steps in the usual three-dimensional game could be obtained similarly to the four-dimensional strategy of Proposition 12; just the role of Proposition 11 should be substituted with Proposition 10.) □

Remark 1. One can prove even stronger statements. For example, it is true that even the knowledge that two fixed points of distance either three or one are not the sink would not help Al: it would still need to evaluate five vertices. Curiously, if it is known about two vertices of distance two that they are not the sink, an algorithm finding the sink in four steps exists.

Proposition 12. $t(4) = 7$.

Proof. To the first query u_1 we answer the source, $s(u_1) = [4]$. If next Al does *not* ask the vertex antipodal to u_1, it is easy to enforce seven steps. The first two queries are then in a common facet $\mathfrak{C}'$, on which we choose and reveal an arbitrary USO. We also tell Al that the label separating $\mathfrak{C}'$ from its antipodal facet $\mathfrak{C}''$ is combed and all edges are oriented *out* of facet $\mathfrak{C}'$ into $\mathfrak{C}''$. On $\mathfrak{C}''$ then we can play a three-dimensional strategy and force Al into five more queries.

If Al's second query $u_2 = \overline{u_1}$ is the antipodal vertex to u_1 we choose an arbitrary label, say 4, and split the cube into two facets $\mathfrak{C}_1$ and $\mathfrak{C}_2$ with respect to 4 ($u_i \in \mathfrak{C}_i$). Except maybe in an answer to the sixth query u_6, we always orient 4-labeled edges *from* $\mathfrak{C}_1$ *into* $\mathfrak{C}_2$. On $\mathfrak{C}_2$ we play the advanced version of the three-dimensional game, namely we avoid that $u_1 \oplus \{4\}$ (which is the antipodal vertex to u_2 in $\mathfrak{C}_2$) is the sink. By the previous proposition, even with this extra information Al needs five steps to find the local sink of $\mathfrak{C}_2$. If Al asks a vertex in $\mathfrak{C}_1$ before he evaluated the sink of $\mathfrak{C}_2$, we reveal the orientation of $\mathfrak{C}_1$ and direct all 4-labeled edges away from $\mathfrak{C}_1$. This makes the local sink in $\mathfrak{C}_2$ the global sink, and Al needs five queries altogether in $\mathfrak{C}_2$ to find it. Since he already used two in $\mathfrak{C}_1$, he needed seven steps altogether.

If Al is patient enough to first find the sink v of $\mathfrak{C}_2$ without further looking at $\mathfrak{C}_1$, then this will happen in step 6 the earliest. Then we answer $s(v) = \{4\}$ and reveal an orientation in $\mathfrak{C}_1$, such that u_1 is the source and $v \oplus \{4\}$ is the sink of $\mathfrak{C}_1$ (Corollary 4). The edge between the sinks v, $v \oplus \{4\}$ of $\mathfrak{C}_1$ and $\mathfrak{C}_2$ is flippable (Corollary 6), so our answer $s(v) = \{4\}$ does not prevent an extension of our answers to a USO of the whole cube. Therefore Al needs one more query to evaluate the global sink $v \oplus \{4\}$, which gives seven queries altogether. □

Remark 2. We note that with similar methods a lower bound of $2n - 1$ could be given in any dimension, which is better for small values than our asymptotic lower bound from Theorem 9. Since the result is not known to be sharp for any value $n \geq 5$ and the analysis is somewhat delicate, we omit the proof for $n \geq 5$.

6. Comments and Open Problems

Of course in an ideal world one could hope to determine the functions $t(n)$, $\tilde{t}(n)$, $t_{\text{acyc}}(n)$ and $\tilde{t}_{\text{acyc}}(n)$ exactly. There are more realistic goals, though, for the near future. A natural question concerning the strategy in our paper is whether the AUSOs we use (i.e. the orientations arising from Lemmas 3 and 5 can be realized geometrically, i.e. as a linear program. A positive answer would be of great interest.

The lone lower bound we are able to give for $\tilde{t}(n)$ is a mere $n/2$, which works even on the set of decomposable orientations. Any nonlinear lower bound would be interesting. An extension of our method for the deterministic lower bound seems plausible.

There is ample space for improvement on the upper bounds as well. The fastest known deterministic algorithm does work for general USOs; a first goal would be to exploit acyclicity in order to separate $t(n)$ and $t_{\text{acyc}}(n)$ from each other.

The *only* known nontrivial randomized algorithms for the general USO problem work in a product-like fashion. An improvement of $\tilde{t}(n)$ for a small value of n implies better algorithms for large n. Unfortunately the determination of $\tilde{t}(n)$ becomes unmanageable relatively soon; the determination of $\tilde{t}(3)$ by Rote [16] already presented a significant computational difficulty, and $\tilde{t}(4)$ is still not known. It would be really desirable to create nontrivial randomized algorithms working by an idea *different* from the Product Algorithm of [18]. A possible approach would be to find a way to randomize the deterministic FibonacciSeesaw algorithm [18].

`RandomEdge` and `RandomFacet` are very natural randomized algorithms, their analysis is extremely inviting but might be hard. The behavior of `RandomFacet` is well understood for AUSO, for general USOs nothing is known. By the construction of Morris [15] `RandomEdge` is not a good choice to find the sink of a general USO, but for AUSOs not much is known about its behavior.

Acknowledgments

We are grateful to Alan Frieze, Jirka Matoušek, Uli Wagner and Emo Welzl for useful comments and conversations. We also thank Emo Welzl for suggesting the algorithm for decomposable orientations in Proposition 7.

References

1. David Aldous. Minimization algorithms and random walk on the d-cube. *The Annals of Probability*, 11:403–413, 1983.
2. Richard W. Cottle, Jong-Shi Pang, and Richrad E. Stone. *The Linear Complementary Problem*. Academic Press, New York, 1992.
3. Bernd Gärtner. A subexponential algorithm for abstract optimization problems. *SIAM Journal on Computing*, 24:1018–1035, 1995.
4. Bernd Gärtner. Combinatorial linear programming: geometry can help. In *Proceedings of the* 2*nd International Workshop on Randomization and Approximation Techniques in Computer Science*, volume 1518 of Lecture Notes in Computer Science, pages 82–96. Springer-Verlag, Berlin, 1998.
5. Bernd Gärtner. The random-facet simplex algorithm on combinatorial cubes. *Random Structures & Algorithms*, 20(3):353–381, 2002.
6. Bernd Gärtner. Linear programming via unique sinks. Manuscript in preparation, 2002.
7. Bernd Gärtner, Martin Henk, and Günter M. Ziegler. Randomized simplex algorithms on Klee–Minty cubes. *Combinatorica*, 18(3):349–372, 1998.
8. Bernd Gärtner. Combinatorial structure in convex programs, Manuscript, 2001. http://www.ti.inf.ethz.ch/ew/workshops/01-lc/cp.html.
9. Bernd Gärtner and Emo Welzl. Linear programming–randomization and abstract frameworks. In *Proceedings of the* 13*th Annual ACM Symposium on Theoretical Aspects of Computer Science*, volume 1046 of Lecture Notes in Computer Science, pages 669–687. Springer-Verlag, Berlin, 1996.
10. Bernd Gärtner and Emo Welzl. Explicit and implicit enforcing–randomized optimization. In *Lectures of the Graduate Program Computational Discrete Mathematics*, volume 2122 of Lecture Notes in Computer Science, pages 26–49. Springer-Verlag, Berlin, 2001.
11. Gil Kalai. Linear programming, the simplex algorithm and simple polytopes. *Mathematical Programming*, 79:217–233, 1997.
12. Jiří Matoušek. Lower bounds for a subexponential optimization algorithm. *RSA*: *Random Structures & Algorithms*, 5(4):591–607, 1994.
13. Jiří Matoušek. The number of unique-sink orientations of the hypercube, to appear in *Combinatorica*.
14. Jiří Matoušek, Micha Sharir, and Emo Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498–516, 1996.
15. Walter D. Morris. Randomized principal pivot algorithms for P-matrix linear complementarity problems. *Mathematical Programming*, Series A, 92:285–296, 2002.
16. Günter Rote. Personal communication.
17. Alan Stickney and Layne Watson. Digraph models of bard-type algorithms for the linear complementary problem. *Mathematics of Operations Research*, 3:322–333, 1978.

18. Tibor Szabó and Emo Welzl. Unique sink orientations of cubes. In *Proceedings of the* 42*nd IEEE Symposium on Foundations of Computer Science*, pages 547–555, 2001.
19. Emo Welzl. Personal communication.
20. Kathy Williamson Hoke. Completely unimodal numberings of a simple polytope. *Discrete Applied Mathematics*, 20:69–81, 1988.

Received October 28, 2002, *and in revised form February* 4, 2003. *Online publication October* 6, 2003.