DISS. ETH NO. 15747

Unique Sink Orientations of Cubes

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZÜRICH
for the degree of Doctor of Sciences

presented by
Ingo Andreas Schurr
Dipl. Math., Freie Universität Berlin, Germany
born March 12, 1972 in Baden–Baden, Germany
citizen of Germany

accepted on the recomodation of
Prof. Dr. Emo Welzl, ETH Zürich, examiner
Dr. Tibor Szabó, ETH Zürich, co-examiner
Prof. Günter Ziegler, TU Berlin, co-examiner

Acknowledgments

I would like to express my deep gratitude to my advisor Emo Welzl. His idea of a Pre-Doc program gave me the chance to sneak into the world of discrete mathematics. Furthermore, all I know about how science works, I know from him.

I am also deeply grateful for being the first Ph.D. student of Tibor Szabó. He was a wonderful advisor. Thanks for reading and commenting (nearly) every single page of this thesis.

Thanks to Günter Ziegler, not only for being my third supervisor, but also for letting me stay in his group in Berlin for one semester.

Without Bernd Gärtner a whole chapter of this thesis would never have been written. Thanks for all the fruitful discussions about linear programing and the business class.

I would like to thank Andrea Hoffkamp (for all the ”germanisms” and the (soap) operas), Arnold Wassmer (for sharing his mathematical thoughts with me), Carsten Lange (for sharing his experience), Enno Brehm (for all I know about programming), Falk Tschirschnitz (for proofreading), Iris Klick (for at least trying to read), Kaspar Fischer (for all his advices and the roar of the tiger), Mark de Longueville (for teaching me Analysis and all the comments), Nando Cicalese (for the espresso and last minute words of wisdom), Shankar Ram Lakshminarayanan (for sharing a flat and all the questions and answers), Uli Wagner (for forcing me into the pre-doc program, sharing a flat and enlightening me) and Zsuzsa (for bringing me to Zürich, sharing a flat and criticizing my criticisms). Without you I would not have been able to finish this thesis.

Thanks to the first pre-docs Frank Vallentin, Godfrey Njulumi Justo, Hiroyuki Miyazawa, Sai Anand, and Stamatis Stefanakos. It was a great half-a-year.

Last, but not least, I am very thankful for being part of the gremo group. Thank you all, it was a pleasure working with you.

Abstract

Subject of this thesis is the theory of unique sink orientations of cubes. Such orientations are suitable to model problems from different areas of combinatorial optimization. In particular, unique sink orientations are closely related to the running time of the simplex algorithm.

In the following, we try to answer three main questions: How can optimization problems be translated into the framework of unique sink orientations? What structural properties do unique sink orientations have? And how difficult is the algorithmic problem of finding the sink of a unique sink orientation?

In connection to the first question, the main result is a reduction from linear programming to unique sink orientations. Although the connection to linear programming was the core motivation for our studies, it was not clear in the beginning how general linear programs can be fit into the theory of unique sink orientations. The reduction presented in this thesis closes this gap.

For the second question we can provide several construction schemes for unique sink orientations. On the one hand we know schemes which allow us to construct all unique sink orientations. On the other hand we present easier constructions, which are still powerful enough to provide us with a number of interesting orientations. The hope that unique sink orientations on their own carry an interesting algebraic structure turns out to be wrong.

Equipped with the construction schemes just mentioned we are able to give some answers to the third question about the algorithmic complexity. The true complexity of the problem of finding a sink of a unique sink orientation remains open. But we can provide first lower bounds for special algorithms as well as for the general case. Furthermore, it turns out that the algorithmic problem is NP-hard only if NP=coNP.

Zusammenfassung

Gegenstand der vorliegenden Arbeit sind Orientierungen des Kanten-Graphs eines Hyperwürfels, so dass jeder Unterwürfel eine eindeutige Senke hat (im folgenden ESE-Würfel genannt). Solche Orientierungen modellieren etliche Probleme aus unterschiedlichen Bereichen der kombinatorischen Optimierung. Insbesondere besteht ein enger Zusammenhang zu der (seit langem) offenen Frage nach der Laufzeit des Simpex-Algorithmus.

Im folgenden werden im Wesentlichen drei Themenkomplexe behandelt: Wie können Optimierungs-Probleme auf ESE-Würfel reduziert werden? Welche strukturellen Aussagen kann man über ESE-Würfel machen? Und wie schwer ist das algorithmische Problem, die Senke eines ESE-Würfels zu finden?

Im Zusammenhang mit der ersten Frage ist die Reduktion von linearem Programmieren auf ESE-Würfel hervorzuheben. Die Verbindung zwischen linearem Programmieren und ESE-Würfeln war zwar von Anfang an Motivation für die vorliegenden Studien, lange Zeit jedoch war es nicht klar, wie allgemeine lineare Programme in die Theorie der ESE-Würfel passen. Die hier vorgestellte Reduktion schliesst diese Lücke.

Bezüglich der zweien Frage sind vor allem eine Reihe von Konstruktionsvorschriften für ESE-Würfel zu nennen. Auf der einen Seite haben wir ein allgemeines Konstruktions-Schema, das stark genug ist, alle ESE-Würfel zu generieren, auf der anderen Seite stellen wir einige deutlich einfachere Konstruktionen vor, die allerdings noch mächtig genug sind, um interessante ESE-Würfel zu beschreiben.

Mit Hilfe dieser Konstruktionsvorschriften ist es uns möglich, auch Antworten auf die dritte Frage nach der algorithmischen Komplexität zu finden. Zwar ist die wirkliche Komplexität des Problems die Senke zu finden weiterhin offen, aber es ist uns möglich, untere Schranken herzuleiten, sowohl für spezielle Algorithmen, als auch für den allgemeinen Fall. Des weiteren wird gezeigt, dass das algorithmische Problem nur dann NP-schwer sein kann, wenn NP=coNP gilt.

Contents

1 Motivation

It's a very funny thought that, if Bears were Bees, They'd build their nests at the bottom of trees. And that being so (if the Bees were Bears), We shouldn't have to climb up all these stairs.

(Winnie The Pooh)

1.1 Linear Programming

Back in the 1940, the term "program" (as military term) referred to a plan or schedule for training, logistical supply or deployment of men. To automatize such programming, the young mathematician George Dantzig introduced a mathematical model for *programming in a linear structure*. He first presented his idea to a broader audience in 1948 at a meeting of the Econometric Society in Wisconsin. In [7] Dantzig writes about this meeting:

> *After my talk, the chairman called for discussion. For a moment there was the usual dead silence; then a hand was raised. It was Hotelling's. I must hasten to explain that Hotelling was fat. He used to love to swim in the ocean and when he did, it is said that the level of the ocean rose perceptibly. This huge whale of a man stood up in the back of the room, his expressive fat face took on one of those all-knowing smiles we all know so well. He said: 'But we all know the world is nonlinear.' Having uttered this devastating criticism of my model, he majestically sat down. And there I was, a virtual unknown, frantically trying to compose a proper reply.*

Even worse, the algorithm Dantzig introduced to solve such linear programs used a discrete structure only. A linear functional on a polytope attains its maximum in a vertex. Therefore, one can find such a maximal vertex by following ascending edges from vertex to vertex. This so-called simplex method only uses the edge graph of a polytope, a discrete object. Although the world is not discrete, linear programming and the simplex method proved to be very powerful tool to solve real-life optimization problems.

Despite its usefulness in practice, the simplex method is bad in theory. In 1972, Klee and Minty [24] constructed examples on which the simplex algorithm using Dantzig's pivot rule visits all vertices. Based on their example, similar worst case behavior can be attained for nearly all deterministic pivot rules (see [3] for a general construction scheme). It took another eight years before Khachiyan [23] developed an algorithm for linear programming which has polynomial running time in the bit-model. That is, given the linear constraints of a linear program encoded in a bit-sequence, Khachiyan's algorithm has a running time polynomial in the number of bits.

From the point of view of the simplex method, this bit-model is not satisfactory. As mentioned earlier, the simplex method employs a rather combinatorial structure of linear programming. In particular, perturbing the linear program (i.e., the defining hyperplanes) slightly does not change its combinatorics. But such perturbation can drastically change the size of its encoding.

This observation leads to the attempt to find algorithms which are polynomial in the combinatorial complexity of a linear program, which is given by the number of linear constraints of the underlying polytope and its dimension. So far, no such algorithm is known nor are there arguments indicating that the problem cannot be solved in polynomial time.

The best known result is achieved by RandomFacet, a randomized algorithm independently developed by Kalai [21] and Matoušek, Sharir, and Welzl [28]. RandomFacet is linear in the number of constraints and subexponential in the dimension. Remarkably, RandomFacet works in a much more general setting than the original problem of solving a linear program.

Following the approach of Kalai, RandomFacet maximizes an abstract objective function. Such a function assigns values to the vertices of the edge graph of a polytope such that every face has a unique maximal vertex. A further abstraction from the values yields an orientation of the edge graph by orienting edges towards the vertex with higher value. Since every face has a maximal vertex v, this vertex is a sink, i.e., every edge incident to v is oriented towards v.[1] Such an orientation is called unique sink orientation.

The focus of this thesis is on unique sink orientations of cubes. That is, we study orientations on the edge graph of a cube such that every subcube has a unique sink.[2] In particular, an abstract objective function on a cube induces a unique sink orientation. Although linear programming was the motivating example for us to study unique sink

[1]See Section 3.1

[2]See Chapter 2.

orientations of cubes, the concept originates from a different source. In 1978, Stickney and Watson [40] introduced such orientations to study certain linear complementarity problems.[3] In this set-up the cube is best viewed as a Boolean lattice rather than as a geometric object.

The main advantage of unique sink orientations of *cubes* over general unique sink orientations (on polytopes) lies in the additional structure of the cube. In particular, in a cube every vertex can be addressed directly. This permits formulating algorithms other than the simplex method. For instance, the fastest known deterministic sink-finding algorithm FibonacciSeesaw [42] maintains a data structure consisting of two antipodal subcubes and jumps between these two subcubes.

1.2 The Object of Interest

A unique sink orientation of a cube is an orientation of the edges of the Boolean lattice such that every subcube has a unique sink. The algorithmic problem we are mainly interested in is to find the sink of such an orientation.

In all applications, the orientation is given implicitly. We therefore assume that we have access to the orientation via an oracle. This oracle, when queried at a vertex, reveals all the edges outgoing from the vertex.

The standard scenario of such an application is the following: Given some problem $\mathcal{P}$ (like for instance a linear complementary problem), we construct an oracle based on the data of $\mathcal{P}$. To each vertex the oracle assigns a potential solution for the original problem. The orientation is obtained by comparing the "solutions" in the vertices. In particular, the "solution" of the sink solves the original problem. Thus, it is not enough to know the position of the sink, but we also want to evaluate the sink. For instance, for linear programming the position of the sink will tell us which constraints have to be tight. But an oracle query for the sink on the way will determine a minimizer of the linear program, hence the optimal value.

Our underlying complexity model[4] measures the running time of an algorithm in terms of the number of oracle queries. In-between two

[3]See Section 3.2.
[4]See Sections 2.3 and 5.1

oracle queries, any amount of computation is permitted. This model undoubtedly constitutes a severe simplification of the real complexity. However, all known algorithms for finding the sink of a unique sink orientation perform only a negligible amount of computation between queries. In fact, we do not know how to exploit the additional computational power, which is an indication that the underlying structure is still not fully understood. From this point of view even an algorithm which needs exponential (or more) time between queries would be of great interest. For now the goal is to find an algorithm which finds the sink of a unique sink orientation using only a small number of oracle queries.

For most concrete problems, the reduction to unique sink orientations of cubes is a heavy abstraction. In fact, if we count the number of d-dimensional unique sink orientations constructed by this class of problems, then this number over the number of all d-dimensional unique sink orientations vanishes as the dimension grows. Still, for linear complementarity problems, for example, this abstraction leads to the fastest known algorithms for solving such problems.

1.3 Outline of the Thesis

In Chapter 2, we present the basic definitions and notations. Cubes and unique sink orientations on them are introduced. As a warm-up, we prove a characterization of acyclic unique sink orientations of cubes. Furthermore, the basic complexity model based on oracle queries is presented.

Several reductions to unique sink orientations are presented in Chapter 3. We start with the two classical examples, linear programming and linear complementarity problems. Then we introduce strong LP-type problems, which are less general than unique sink orientations. This scheme captures the main features which make optimization problems tractable by unique sink orientations. As an example, we reduce general linear programming to strictly convex quadratic programming, and strictly convex quadratic programming to strong LP-type problems.

In Chapter 4, we summarize some structural facts about unique sink orientations of cubes. Such orientations are strongly related to permutations. Still, the set of all unique sink orientations of cubes itself does not carry an algebraic structure of its own. We give several construction schemes for unique sink orientations.

In Chapter 5, these construction schemes are used to construct examples on which deterministic algorithms for finding the sink have bad running time.

In Chapter 6, we introduce a data structure to represent unique sink orientations of cubes. Furthermore, we describe an isomorphy test. A procedure to enumerate all d-dimensional unique sink orientations is introduced. The chapter closes with some counting arguments for the number of unique sink orientations in small dimensions.

1.4 Remarks on the Notation

Throughout the thesis, objects are named according to their use rather than to their kind. For example, we use the letters I, J for sets if they denote delimiters of an interval of sets, and u, v if they denote vertices. The conventions used trough-out this thesis are listed below.

Combinatorics:

$\mathfrak{C}$	: a cube
I, J	: delimiters of an interval of sets
u, v	: vertices
V	: the set of vertices
W	: subset of V
e	: edge
E	: the set of edges
L	: subset of E
λ	: label
Λ	: set of labels
ϕ	: orientation
d	: dimension
s	: outmap
o,σ	: sink of USO

Linear Algebra:

n, m	: dimension
b, c, x	: vector
A, B, M, Q	: matrix
π,ι, κ	: projection
$\mathcal{C}$	: cone
$\mathcal{F}$	: face
$\mathcal{H}$	: affine subspace
$\mathcal{P}$	: polyhedron

Others:

α	: scalar in $\mathbb{R}$
θ	: scalar in $[0, 1]$
τ	: bijection
Ψ	: isomorphism
f	: continuous function
$\mathcal{A}$	: algorithm
p	: probability
Σ	: alphabet
L	: language

2 Basics

With the tips of your toes somewhat floating, tread firmly with your heels.

(Miyamoto Musashi)

2.1 The Cube

We refer by "cube" to the edge graph of a geometric cube rather than the geometric object itself. In this section, we introduce two formalizations of the cube, Boolean lattices and 0/1-words. We will use both interchangeably in the course of this thesis.

Boolean Lattice: Let us first fix some set theoretic notation. The power set of a set J is denoted by $2^J := \{u \subseteq J\}$. For two sets $I \subseteq J$ we define the interval between I and J by $[I, J] := \{v \mid I \subseteq v \subseteq J\}$. The symmetric difference of two sets I, J is the set $I \oplus J := (I \cup J) \setminus (I \cap J)$. Furthermore, for a positive integer d we set $[d] := \{1, \ldots, d\}$.

Definition 2.1

Given two finite sets $I \subseteq J$, the cube $\mathfrak{C}^{[I,J]}$ spanned by I and J is the graph with vertex and edge set

$$\begin{array}{rcl} \mathrm{V}(\mathfrak{C}^{[I,J]}) & := & [I, J] = \{v \mid I \subseteq v \subseteq J\} \\ \{u, v\} \in \mathrm{E}(\mathfrak{C}^{[I,J]}) & :\Leftrightarrow & |u \oplus v| = 1. \end{array} \tag{2.1}$$

Furthermore, let $\mathfrak{C}^J := \mathfrak{C}^{[\emptyset,J]}$ and $\mathfrak{C}^d := \mathfrak{C}^{[d]} = \mathfrak{C}^{[\emptyset,\{1,\ldots,d\}]}$.

Whenever possible we will choose $\mathfrak{C}^d$ as our standard model. The more involved definition of $\mathfrak{C}^{[I,J]}$ is mainly needed to properly address the subcubes of $\mathfrak{C}^d$. Let $\mathfrak{C}^{[I,J]}$ and $\mathfrak{C}^{[I',J']}$ be given. The cube $\mathfrak{C}^{[I,J]}$ is a subcube of $\mathfrak{C}^{[I',J']}$ if and only if $[I, J] \subseteq [I', J']$, i.e., $I' \subseteq I \subseteq J \subseteq J'$. In particular, the poset

$$(\{[I, J] \mid I \subseteq J \subseteq [d]\}, \subseteq)$$

describes the face-lattice of $\mathfrak{C}^d$. We will identify $[I, J]$ and $\mathfrak{C}^{[I,J]}$.

The edges of such a cube are labeled in a natural way: The label of an edge $\{u, v\}$ is the unique $\lambda \in u \oplus v$. We will refer to edges with label λ as *λ-edges*. The set of all labels is called the *carrier*,

$$\operatorname{carr} \mathfrak{C}^{[I,J]} = J \setminus I.$$

See Figure 2.1 for an example of a cube.

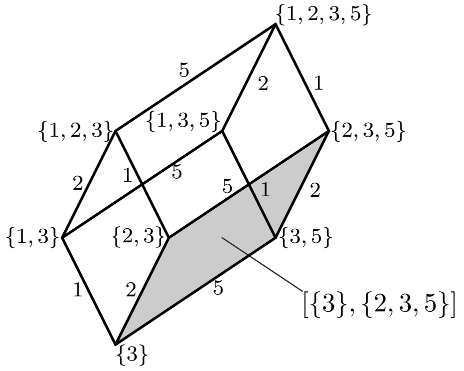


Figure 2.1: The cube $\mathfrak{C}^{[\{3\},\{1,2,3,5\}]}$ with edge-labels.

Up to isomorphism, a cube is completely determined by the cardinality of its carrier: Let $d = |\operatorname{carr}\mathfrak{C}|$. For any bijection $\pi : [d] \to \operatorname{carr}\mathfrak{C}$ and any vertex $u \in \mathfrak{C}$, the map

$$[d] \ni v \mapsto u \oplus \{\pi(\lambda) \mid \lambda \in v\}$$

is a bijection and maps λ-edges to $\pi(\lambda)$-edges. Therefore, this map is a graph-isomorphism of cubes. In other words, a cube $\mathfrak{C}$ is fully determined by its *dimension*

$$\dim\mathfrak{C} = |\operatorname{carr}\mathfrak{C}|.$$

A posteriori, this justifies talking about the standard d-cube $\mathfrak{C}^d$, since

$$\mathfrak{C}^{[I,J]} \cong \mathfrak{C}^{J\setminus I} \cong \mathfrak{C}^{|J\setminus I|}.$$

0/1**-words:** Another way to express cubes is via 0/1-words. A 0/1-*word* of length d is a sequence $u = (u_1, \dots, u_d)$ in $\{0,1\}^d$. The set of all words carries a metric structure: Two words $u, v \in \{0,1\}^d$ have *Hamming-distance* k if they disagree in exactly k positions, i.e.

$$d_H(u,v) = |\{\lambda \mid u_\lambda \neq v_\lambda\}|.$$

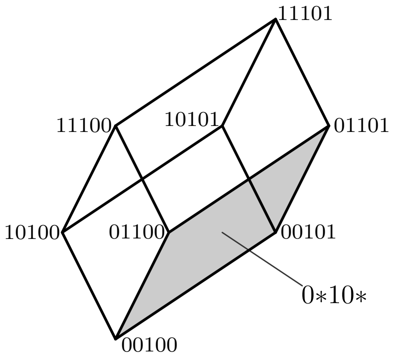


Figure 2.2: The cube $\mathfrak{C}^{**10*}$.

From this viewpoint, a d-dimensional cube $\mathfrak{C}^d$ is the graph given by

$$\begin{array}{rcl} \mathrm{V}(\mathfrak{C}^d) & := & \{0,1\}^d \\ \{u,v\} \in \mathrm{E}(\mathfrak{C}^d) & :\Longleftrightarrow & d_H(u,v) = 1. \end{array} \tag{2.2}$$

In fact, as we can identify subsets of $[d]$ with the characteristic vector (seen as a 0/1-word), this definition of a cube coincides with the previous definition. As for sets, we define $\oplus$ as the component-wise operation with $0 \oplus 1 = 1 \oplus 0 = 1$ and $0 \oplus 0 = 1 \oplus 1 = 0$.

To express subcubes in the 0/1-world, we extend the alphabet by a wildcard-symbol $*$. A word $w \in \{0,1,*\}^d$ defines a subcube $\mathfrak{C}^w$ by

$$\mathrm{V}(\mathfrak{C}^w) = \left\{ u \in \{0,1\}^d \;\middle|\; \forall \lambda : w_\lambda \neq * \quad \Rightarrow \quad u_\lambda = w_\lambda \right\}, \tag{2.3}$$

i.e., w and u can only differ in labels λ for which $w_\lambda = *$. See Figure 2.2.

The number of $*$'s in w equals the dimension of $\mathfrak{C}^w$. A subcube of dimension k is called *k-face*. Furthermore, subcubes of dimension 0, 1 and $d-1$ are called *vertices*, *edges* and *facets*, respectively. Similar to edges we can define the notion of a λ-facet. The *upper λ-facet* is the subcube defined by the word $* \dots *1* \dots *$ consisting of $*$'s and one 1 at position λ. Analogously, the *lower λ-facet* is the subcube defined by the word $* \dots *0* \dots *$ consisting of $*$'s and one 0 at position λ.

In general, for a subcube defined by w, the *antipodal* subcube is defined the following way: Extend $\oplus$ to $\{0, 1, *\}$ by $x \oplus * = * \oplus x = *$ for $x \in \{0, 1, *\}$. Then, for $w \in \{0, 1, *\}$ the antipodal word is $\bar{w} = w \oplus 1 \cdots 1$ and $\mathfrak{C}^{\bar{w}}$ is called antipodal to $\mathfrak{C}^{w}$.

2.2 Unique Sinks

Given a graph $G = (V, E)$ a *partial orientation* is a map $\phi : L \to V$, $L \subseteq E$, such that $\phi(e) \in e$ for all $e \in L$. If $L = E$ we call ϕ an orientation of G. For an edge $e = \{u, v\}$ with $\phi(e) = v$ write $u \overset{\phi}{\to} v$. If ϕ is clear from the context we simply write $u \to v$. The vertex v is called the *sink* of e and u is called the *source* of e. Furthermore, we say e is *incoming* in v, *directed towards* v, and *outgoing* from u. The *out-degree* (*in-degree*) of a vertex v is the number of outgoing (incoming) edges incident to v.

Two orientations ϕ of $G = (V, E)$ and ϕ' of $G' = (V', E')$ are *isomorphic* if there is a bijection $\alpha : V \to V'$ which preserves edges ($\{u, v\} \in E \iff \{\alpha(u), \alpha(v)\} \in E'$) and orientations:

$$u \overset{\phi}{\to} v \iff \alpha(u) \overset{\phi'}{\to} \alpha(v).$$

In other words, an isomorphism of orientations is a graph-isomorphism which maps sinks to sinks.

A vertex o in G is called *sink* of G if it has no outgoing edges, i.e., is sink of all incident edges. Accordingly, a *source* is a vertex with no incoming edges. In a subcube $\mathfrak{C}'$ of $\mathfrak{C}$, the orientation ϕ induces an orientation ϕ' by

$$u \overset{\phi'}{\to} v \iff u \overset{\phi}{\to} v$$

for an edge $\{u, v\}$ in $\mathfrak{C}'$. The orientations we are interested in have unique sinks in the induced orientations on all subcubes.

Definition 2.2
A unique sink orientation (USO) of the d-dimensional cube is an orientation ϕ of $\mathfrak{C}^d$, such that any subcube has a unique sink. More formally,

$$\forall I \subseteq J \subseteq [d] \quad \exists! u \in [I, J] \quad \forall \lambda \in J \setminus I : u \oplus \{\lambda\} \to u. \tag{2.4}$$

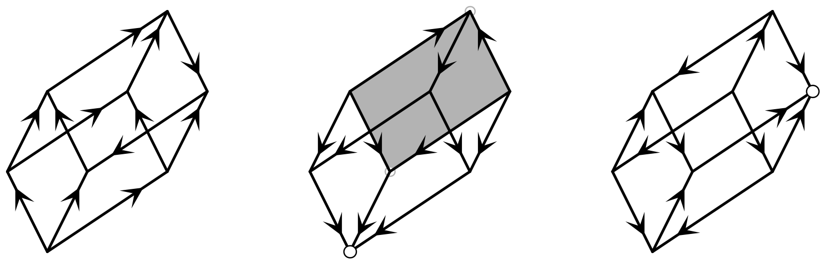


Figure 2.3: The leftmost orientation has no global sink. The center orientation has a global sink but the gray facet has two sinks. Ergo both orientations are not USOs. The rightmost orientation is a USO.

For instance, the leftmost orientation in Figure 2.3 has no global sink. The center one has a global sink but is not a USO since one of its facets has two sinks. Finally, the rightmost orientation is a USO.

As a small warm-up, we consider low dimensions. The 0-dimensional cube $\mathfrak{C}^0$ consists of exactly one vertex and no edges. Obviously, this one vertex has no outgoing edges, i.e., is the unique sink. The cube $\mathfrak{C}^1$ consists of two vertices $u = \emptyset$ and $v = \{1\}$ connected by one edge. There are only two possible orientations, namely $\phi_1(\{u, v\}) = u$ and $\phi_2(\{u, v\}) = v$. For ϕ_1 the vertex u is the unique sink and for ϕ_2 the vertex v is the unique sink. In particular, for vertices and edges the unique sink property is always satisfied.

The two-dimensional cube $\mathfrak{C}^2$ has four vertices $u_0 = \emptyset$, $u_1 = \{1\}$, $u_2 = \{2\}$, and $u_3 = \{1, 2\}$. The edges are $\{u_0, u_1\}$, $\{u_0, u_2\}$, $\{u_1, u_3\}$ and $\{u_2, u_3\}$. An orientation ϕ assigns to each of these four edges one of its incident vertices. Thus, there are 16 orientations of $\mathfrak{C}^2$.

Let ϕ be an orientation of $\mathfrak{C}^2$. If ϕ has no sink, each vertex has at least one outgoing edge. Thus, ϕ must be a cycle. See the left picture in Figure 2.4. Now assume ϕ has a sink in $v_0 \in \{u_0, \ldots, u_3\}$. That is, its two incident edges $\{v_0, v_1\}$ and $\{v_0, v_2\}$ are directed towards v_0. In particular, v_1 and v_2 have at least two outgoing edges. Thus, for the out-degree of v_1 and v_2 three possible cases remain.

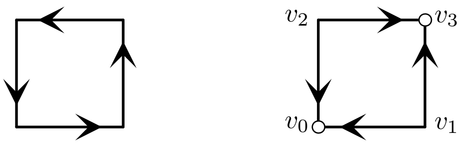


Figure 2.4: The cyclic orientation and the orientation with two sinks.

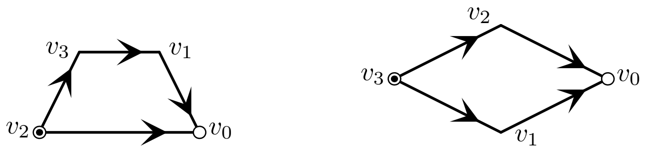


Figure 2.5: The bow and the eye.

1. The out-degree of both v_1 and v_2 is two. Then the edges $\{v_1, v_3\}$ and $\{v_2, v_3\}$ are both directed towards v_3. Hence, v_3 is also sink and ϕ is not a unique sink orientation. See the picture to the right in Figure 2.4.

2. One vertex has out-degree 1 and the other vertex has out-degree 2. Without restriction v_1 is the vertex with out-degree 1. Then the edge $\{v_1, v_3\}$ is directed towards v_1. In particular, v_3 is not a sink. Thus, ϕ is a USO. See the picture to the left in Figure 2.5.

3. Both vertices have out-degree 1. Then both edges $\{v_1, v_3\}$ and $\{v_2, v_3\}$ are outgoing in v_3. Again v_0 is the only sink and ϕ is a USO. See the picture to the right in Figure 2.5.

To construct a 2-dimensional USO we can either use the second case or the third case of the above enumeration. In the second case we first choose $v_0 \in \{u_0, \ldots, u_3\}$ and then v_2 from the two neighbors of v_0. The edges incident to v_0 we orient towards v_0 and the edges in v_2 we direct away from v_2. This leaves one edge undirected, namely the edge between the other neighbor v_1 of v_0 and the vertex v_3 antipodal to v_0. By the first case we have to orient this edge towards v_1 to obtain a USO. A

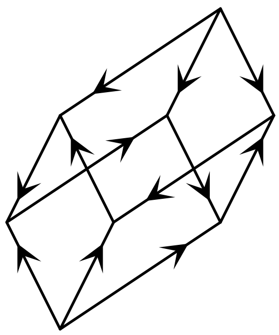


Figure 2.6: Only bows but no global sink.

USO of this type is called a *bow.* Since we have four choices for v_0 and then two choices for v_2, there are eight bows.

In the third case we again choose v_0. Then we orient the edges incident to v_0 towards v_0 and the edges incident to the antipodal point v_3 away from v_3. This type of USOs is called an *eye.* Since we only have four choices for v_0, there are four eyes.

Definition 2.3
An orientation ϕ of the d-dimensional cube $\mathfrak{C}^d$ is called 2-USO *if every 2-face has a unique sink.*

In other words, 2-USOs are the orientations which can be composed of eyes and bows. It is not too surprising that there are 2-USOs which are not USOs. See e.g. the orientation in Figure 2.6. However, the class of 2-USOs is not much larger than the class of USOs. As is shown by Matoušek [27], the number of 2-USOs and the number of USOs are asymptotically of the same order of magnitude in the exponent:

$$\begin{aligned} 2^{\Omega(2^d \log d)} &\leq \# \ d\text{-dim. USOs} \\ &\leq \# \ d\text{-dim. 2-USOs} \leq 2^{O(2^d \log d)} \end{aligned}$$

Furthermore, for acyclic orientations the two classes coincide.

Theorem 2.4 ([17])
An acyclic 2-USO of a cube is a unique sink orientation.

For proving Theorem 2.4, we need the following lemma.

Lemma 2.5
Let ϕ be an acyclic 2-USO. Then a sink in ϕ has a neighbor of out-degree 1.

Proof. Let o be a sink of ϕ and $N(o)$ the set of all neighbors of o. Assume that every $v \in N(o)$ has at least two outgoing edges. By this assumption, for every $v \in N(o)$, there is a label λ_v with $v \to v \oplus \{\lambda_v\} \neq o$. Since o, v, $v \oplus \{\lambda_v\}$ and $o \oplus \{\lambda_v\}$ form a 2-face in which o is the sink, we have $v \oplus \{\lambda_v\} \to o \oplus \{\lambda_v\}$.

In particular, ϕ induces on $N(o) \cup \{v \oplus \{\lambda_v\} \mid v \in N(o)\}$ a sink-less orientation. Such a graph (and thus ϕ on the whole cube) contains a cycle. This is a contradiction to ϕ being cyclic. Thus, at least one of the neighbors of o has out-degree 1. $\square$

Proof of Theorem 2.4. We show by induction on the dimension that an acyclic 2-USO is a USO.

The statement is trivially true for dimension 2. Now assume the statement is true for dimension $d-1$. Let ϕ be an acyclic orientation of the d-cube, such that every 2-face is a USO. By induction all facets are USOs, since they are $(d-1)$-dimensional. It remains to show that there is exactly one global sink.

In particular, the lower and the upper facets along label d have unique sinks o_l and o_u, respectively. Since all other vertices have an outgoing edge in their d-facet, o_l and o_u are the only candidates for a global sink.

If neither o_l nor o_u is a global sink, then all vertices have at least one outgoing edge, hence ϕ contains a cycle. By assumption, this case cannot occur, and at least one of the two vertices has to be a sink.

If o_l and o_u are in a common λ-facet, then, by induction, only one is a sink in this facet, say $o = o_l$. Since $\lambda \neq d$ vertex o is the unique global sink.

The case remains that o_l and o_u are antipodal and at least one of them is a global sink. We are done if we can rule out the possibility that both are global sinks. Thus, assume o_l and o_u are both global

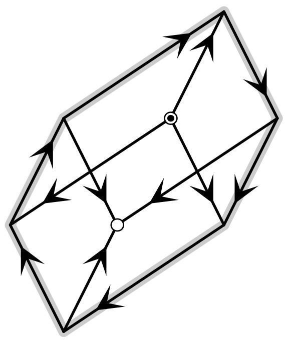


Figure 2.7: A cyclic unique sink orientation.

sinks. A neighbor v of o_l is in a common facet with o_u. In this facet, o_u is a sink. So v has an outgoing edge different from $v \to o_l$. Thus, no neighbor of o_l has out-degree 1 in contradiction to Lemma 2.5. $\square$

The following question arises: Are there cyclic USOs? Figure 2.7 answers this question affirmatively.

2.3 Complexity Issues

Given a USO, the goal is to find its sink. As there are $2^{\Theta(2^d \log d)}$ USOs of dimension d (see [27]), at least $\Theta(2^d \log d)$ bits are needed to encode a particular USO. Thus, if the USO is given explicitly as input to an algorithm, we can find the sink in time linear in the input size by just scanning through the vertices.

We will, however, adopt a different complexity model. A USO is given implicitly by an oracle. Such oracle, when queried in a vertex, reveals the orientation in this vertex.

Definition 2.6
Given a unique sink orientation ϕ on a cube $\mathfrak{C}$, the outmap *s of ϕ is the map assigning to every vertex the labels of outgoing edges, i.e.,*

$$s : \mathrm{V}(\mathfrak{C}) \to 2^{\operatorname{carr} \mathfrak{C}} \quad , \quad v \mapsto \{\lambda \mid v \to v \oplus \{\lambda\}\} \, .$$

Any algorithm can access the outmap of a USO only via an oracle. That is, the algorithm can ask for the outmap in one vertex at a time. The task is to query the sink. The running time is measured in the number of queries to the oracle.

As we will learn in the next chapter, many problems (such as linear programming) can be transformed into unique sink orientations in such a way that finding the sink of this orientation solves the original problem. More concretely, given some problem $\mathcal{P}$, we want to define an oracle for a USO based on $\mathcal{P}$. This oracle should be able to perform a vertex query fast. Furthermore, after querying the sink of the USO, the solution to $\mathcal{P}$ can be found fast. Hence, an *oracle* consists of two algorithms. One algorithm computes $s(v)$ from $\mathcal{P}$. The other algorithm determines the solution of $\mathcal{P}$ given the sink of s. If both algorithms have a polynomial running time, the oracle is called *polynomial-time unique sink oracle*. For such oracles, an algorithm finding the sink in a USO solves the original problem with an additional polynomial factor.

Obviously, $O(2^d)$ queries are enough to find the sink of a USO, since after querying all 2^{d-1} vertices of even cardinality, we know the entire orientation. We aim for algorithms asking only $o(2^d)$ queries. In consequence, we know only $o(d2^d)$ edges of the orientation. Such an algorithm is unable to verify whether the oracle was based on a USO.

Let ϕ be a partial orientation on a cube, such that exactly two adjacent edges $e_1 = \{v \oplus \{\lambda_1\}, v\}$ and $e_2 = \{v, v \oplus \{\lambda_2\}\}$ are not oriented. In the 2-face $\mathfrak{C}_0$ spanned by the vertex v and the labels λ_1 and λ_2, only the orientation in the vertex $\bar{v} = v \oplus \{\lambda_1, \lambda_2\}$ antipodal to v is known. Let ϕ' be an extension of ϕ such that

$$v \to v \oplus \{\lambda_i\} \iff \bar{v} \to \bar{v} \oplus \{\lambda_i\}$$

for $i = 1, 2$. A closer look at Figure 2.5 shows that then $\mathfrak{C}_0$ can neither be an eye nor a bow, so ϕ' is not a USO. In consequence, it is undecidable if an orientation is a USO as long as the orientation of less than $(d-1)2^{d-1}$ edges is known, since then there is at least one vertex for which the orientation of two incident edges are unknown.

Definition 2.7
Given an outmap s on a d-dimensional cube, we consider the following algorithmic problems:

- SINK *is the problem of querying the sink of s provided that s is a unique sink orientation.*
- SINKORFALSIFY *is the problem of either querying a sink of s or finding a certificate for s not being a unique sink orientation.*

If we know a bound on the number of queries for some algorithm solving SINK, then this algorithm also solves SINKORFALSIFY: If the algorithm needs too many queries on some orientation, the sequence of queried vertices is a certificate that this orientation is not a USO.

Definition 2.8
For a deterministic algorithm $\mathcal{A}$ and a unique sink orientation s, let $t_{\mathcal{A}}(s)$ be the number of queries of $\mathcal{A}$ until the sink is queried. The worst case behavior of $\mathcal{A}$ is defined by

$$t_{\mathcal{A}}(d) := \max\left\{t_{\mathcal{A}}(s) \mid s \textit{ d-dim. USO}\right\}.$$

Furthermore, let $t(d)$ be the minimal $t_{\mathcal{A}}(d)$ over all deterministic algorithms $\mathcal{A}$ solving SINK.

For randomized algorithms, in analogy to the deterministic case we define $\tilde{t}$:

Definition 2.9
For a randomized algorithm $\mathcal{A}$ and a unique sink orientation s, let $\tilde{t}_{\mathcal{A}}(s)$ be the expected number of queries of $\mathcal{A}$ until the sink is queried. The worst case behavior of $\mathcal{A}$ is defined by

$$\tilde{t}_{\mathcal{A}}(d) := \max\left\{\tilde{t}_{\mathcal{A}}(s) \;\middle|\; s \textit{ d-dim. USO}\right\}.$$

Furthermore, let $\tilde{t}(d)$ be the minimal $\tilde{t}_{\mathcal{A}}(d)$ over all randomized algorithms $\mathcal{A}$ solving SINK.

In both definitions the algorithm has to query the sink in order to terminate. For example, in dimension 0 any algorithm needs exactly one query to find the sink, although we know the position of the sink.

2.4 Remarks

The notation presented in this chapter follows [42]. To the best of our knowledge, USOs in their full generality were first studied in [42] as independent objects, even though the connection with linear programming or linear complementarity problems has been made by other authors before [40, 1]. The smaller class of *acyclic* USOs appeared much earlier under several different names, e.g. pseudo-Boolean functions [16], abstract objective functions [1], and completely unimodular numberings [18].

Having Theorem 2.4 in mind, one might hope that the acyclic USOs can further be distinguished from general USOs. In fact, for acyclic USOs there exists a randomized algorithm which requires a subexponential number of queries [9, 10]. For general USOs, no such algorithm is known. In contrast, the question about the number of acyclic USOs is still open.

Let us point out that the complexity model introduced in this chapter has to be handled with care. Not only is it necessary that the transition between concrete problems and USOs is polynomial. In addition, we are only counting the number of vertex evaluations an algorithm requires. Thus, in theory, between two vertex evaluations, an algorithm for finding the sink in a USO can spend exponential time without increasing its “running time”. However, so far, no algorithm for this problem actually makes use of this additional power. The main reason is that we do not know how to use it.

3 Sources

There was a red-haired man who had no eyes or ears. Neither did he have any hair, so he was called red-haired theoretically.

(Daniil Kharms)

In this chapter we will often consider $\mathbb{R}^d$ as a vector space. From now on we fix the standard basis $\{e_1, \ldots, e_d\}$ of unit vectors. In particular, linear functions and matrices are identified with respect to the standard basis. Furthermore, we use the standard scalar product $\langle , \rangle$ on $\mathbb{R}^d$. The symbol $\mathbb{I}$ denotes the identity matrix of the appropriate dimension. Also 0 represents the origin of the appropriate space. If not defined otherwise, operations are performed component-wise. For instance, $\mathbb{R}^n_+ := \{x \in \mathbb{R}^n \mid x \geq 0\}$ is the set of all non-negative vectors in $\mathbb{R}^n$, also known as the *non-negative orthant*.

3.1 Linear Programming

In linear programming, the aim is to maximize a linear function over a polyhedron. A *polyhedron* is defined as the intersection of finitely many half-spaces. A bounded polyhedron is called polytope. In the following, we will restrict our attention to polyhedra of the form $\mathcal{P} = \mathcal{H} \cap \mathbb{R}^n_+$, where $\mathcal{H}$ is an affine subspace of $\mathbb{R}^n$. In this setup, a polyhedron is fully determined by a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $b \in \mathbb{R}^m$, namely

$$\begin{aligned} \mathcal{H}(A,b) &:= \{x \in \mathbb{R}^n \mid Ax = b\} && (3.1) \\ \mathcal{P}(A,b) &:= \mathcal{H}(A,b) \cap \mathbb{R}^n_+ = \{x \in \mathbb{R}^n \mid Ax = b, \quad x \geq 0\}\,. && (3.2) \end{aligned}$$

In fact, up to affine transformation every polyhedron can be described by a $\mathcal{P}(A,b)$. For more details on polytopes and polyhedra see [44]. For a simple example, see Figure 3.1.

A *linear program* is defined by a matrix $A \in \mathbb{R}^{m \times n}$ and vectors $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. The goal is then to solve the following problem:

$$\begin{array}{rrcl} \max & c^T x & & \\ \text{such that} & Ax & = & b \\ & x & \geq & 0, \end{array}$$

that is, we want to maximize the linear function $x \mapsto c^T x$ over the polyhedron $\mathcal{P}(A,b)$. We denote this linear program by $\mathrm{LP}(A,b,c)$. The function $x \mapsto c^T x$ is called *objective function.*

A polyhedron may be empty or may contain points with arbitrary large values of $c^T x$. In these cases, the linear program is called infeasible, respectively unbounded.

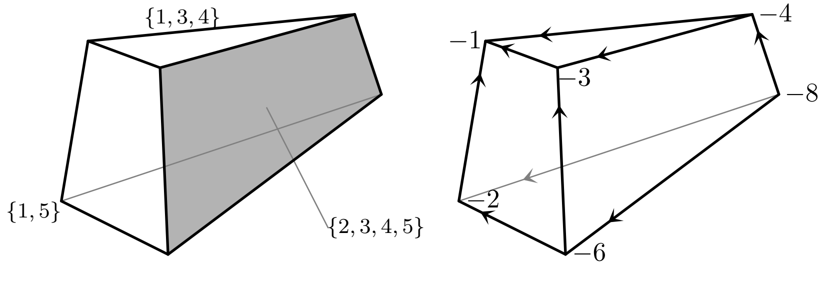


Figure 3.1: The above three-dimensional polytope $\mathcal{P}$ can be realized as a subset of $\mathbb{R}^5$. It is the intersection of the non-negative orthant and the affine subspace given by the two equations $x_1+x_2+x_3-x_4 = 1$ and $x_1+x_2+x_3+x_5 = 1/2$. On the left we see the J's for some faces with respect to this realization. On the right the same polytope is directed according to the ascent of the linear function $c = (-2, -6, -8, 0, 0)$.

A *face* $\mathcal{F}$ of a polyhedron $\mathcal{P} = \mathcal{P}(A, b)$ is defined by tightening some of the non-negativity constraints: For $J \subseteq [n]$, we require all coordinates outside J to be zero, and define the face

$$\mathcal{F}(\mathcal{P}, J) = \{x \in \mathbb{R}^n \mid Ax = b,\, x \geq 0,\, \forall i \notin J : x_i = 0\}.$$

By identifying $\mathbb{R}^J$ with $\{x \in \mathbb{R}^n \mid \forall i \notin J : x_i = 0\}$ a face can be written compactly as

$$\mathcal{F}(\mathcal{P}, J) = \mathcal{P} \cap \mathbb{R}^J. \tag{3.3}$$

A face $\mathcal{F}$ can have different J's representing it. But if $\mathcal{F}(\mathcal{P}, J_1) = \mathcal{F} = \mathcal{F}(\mathcal{P}, J_2)$ then also $\mathcal{F} = \mathcal{F}(\mathcal{P}, J_1 \cap J_2)$. Thus, there is a minimal J representing $\mathcal{F}$.

The *face lattice* of a polyhedron $\mathcal{P}$ is the poset

$$\Big(\{\mathcal{F}(\mathcal{P}, J) \mid J \subseteq [n]\}, \subseteq\Big).$$

Two polyhedra are called *combinatorially equivalent* if they have isomorphic face lattices. As usual, the faces directly above $\emptyset$ are called vertices, the faces directly above vertices are edges and facets are the faces directly below $\mathcal{P}$.

If a linear program has a finite optimal value, the optimum is always attained in a vertex. Thus, from a combinatorial point of view, we can abstract from the concrete values and consider only the ordering of the vertices induced by the objective function. To avoid technical problems we restrict ourselves in this section to polytopes. For polytopes any objective function yields a finite optimum.

Definition 3.1
An abstract objective function *on a (combinatorial) polytope $\mathcal{P}$ is a partial order $\preceq$ of the vertices of $\mathcal{P}$, such that every face has a unique maximal vertex.*

Abstract objective functions were first defined by Alder and Saigal [1]. They provide the general framework in which the first combinatorial subexponential algorithm RandomFacet [21] works.

In a sufficiently generic linear program, the linear function c assigns a different value to each vertex. (If two vertices have the same value, a slight perturbation of c resolves this problem.) Order the vertices according to their values, i.e., set $u \prec v$ if $c^T u < c^T v$. Then the relation $\prec$ is a finite linear order on the set of vertices. In particular, every set of vertices has a maximum and $\prec$ defines an abstract objective function on $\mathcal{P}$. Thus, abstract objective functions are a generalization of linear programming.

Moreover, abstract objective functions are the link between linear programming and unique sink orientations. Any abstract objective function defines an orientation of the edge-graph of $\mathcal{P}$: for an edge $\{u, v\}$, direct

$$u \to v :\Longleftrightarrow u \prec v. \tag{3.4}$$

With this orientation, a maximum of a face is a sink of this face. Consequently, if we start with an abstract objective function, every face has a unique sink.

Definition 3.2
A unique sink orientation *of a polytope $\mathcal{P}$ is an orientation of the edge graph of $\mathcal{P}$, such that every face has a unique sink.*

As we have just seen, every abstract objective function induces a unique sink orientation of $\mathcal{P}$. A unique sink orientation induced by an abstract objective function $\preceq$ cannot have cycles: A path $u_1 \to u_2 \to \ldots \to u_n$ induces $u_1 \prec u_n$. Thus, in particular, $u_1 \neq u_n$. On the other hand, if a unique sink orientation is acyclic, we can find a linear order $\prec$ on the vertices, such that an edge $u \to v$ implies $u \prec v$. Since the ordering is linear every face (even every set of vertices) has a unique maximum and $\prec$ is an abstract objective function. Therefore, abstract objective functions and acyclic unique sink orientations describe the same object.

Abstract objective functions are more general than linear programming. Figure 3.2 shows two acyclic unique sink orientations which are not induced by a linear program. In general, it is difficult to decide if an abstract objective function can be obtained from a linear program. Nevertheless, for the orientations in Figure 3.2 the following theorem by Holt and Klee [19] is sufficient.

Theorem 3.3 ([19])
Let ϕ be a unique sink orientation of a d-dimensional polytope induced by a linear program. Then every k-dimensional face ($k \leq d$) has a unique source and a unique sink and k vertex-disjoint paths from source to sink.

It is easy to check that in the two examples in Figure 3.2, there exist only two vertex-disjoint paths from the global source to the global sink. Hence, the orientations cannot be induced by a linear program. In general, the Holt-Klee condition is not sufficient. For a four-dimensional example of a USO which satisfies the Holt-Klee condition but is not realizable see [32].

Definition 3.2 already defines unique sink orientations of cubes. But it is misleading for all but this section to think of a cube as a geometric object. Instead, unique sink orientations of cubes are rather orientations of the Boolean lattice than a geometric cube.

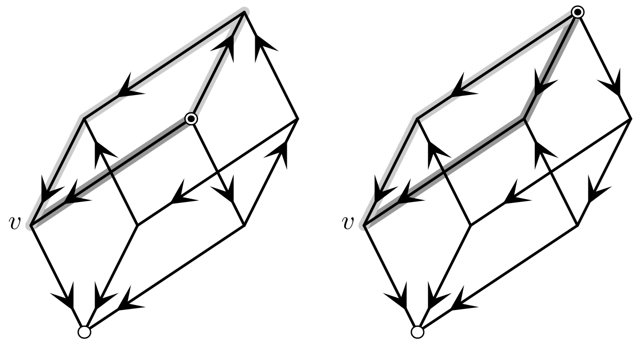


Figure 3.2: Two acyclic USOs which are not induced by a linear program. The two highlighted paths share the vertex v.

3.2 Linear Complementarity Problems

Given a matrix $M \in \mathbb{R}^{n \times n}$ and a vector $q \in \mathbb{R}^n$, the linear complementarity problem LCP(M, q) is to find vectors $w, z \in \mathbb{R}^n$, such that

$$\begin{array}{rcl} w - Mz & = & q \\ w & \geq & 0 \\ z & \geq & 0 \\ w^T z & = & 0 \end{array}$$

is satisfied. Obviously, such vectors do not exist for all M and q. For instance, $w - Mz = q$ might not be solvable. But if the vectors exist, then for i with $w_i > 0$ the coordinate z_i has to be 0 and vice versa (since w, z are non-negative). In other words, the non-zero entries of a solution to LCP(M, q) are complementary.

In the following discussion we will restrict our attention to so-called P-matrices, i.e., matrices for which all principal minors are positive. P-matrices are such that their corresponding LCP is always solvable, for any q.

Theorem 3.4 ([37])
A matrix $M \in \mathbb{R}^{n\times n}$ is a P-matrix if and only if for all $q \in \mathbb{R}^n$ the linear complementarity problem, $LCP(M,q)$, has a unique solution.

For a proof, see e.g. [6, Chapter 3.3].

If we knew the orthogonal coordinate subspaces $\mathbb{R}^{v^*}$ and $\mathbb{R}^{[n]\setminus v^*}$ in which the solution w^*, z^* of LCP(M,q) lives in, i.e., if we knew $v^* \subseteq [n]$, such that $w^* \in \mathbb{R}^{v^*}$ and $z^* \in \mathbb{R}^{[n]\setminus v^*}$, the vectors w^* and z^* would solve

$$\begin{array}{rcl} w - Mz &=& q \\ w &\in& \mathbb{R}^{v^*} \\ z &\in& \mathbb{R}^{[n]\setminus v^*}. \end{array} \tag{3.5}$$

For a P-matrix (3.5) is a system of linear equations in $2n$ variables of rank $2n$. Hence, knowing v^* solves the LCP (up to solving this system of linear equations).

The equation system (3.5) can be simplified in the following way: For $v \subseteq [n]$ and matrices $A = (a_{ij})_{i,j\in[n]}$ and $B = (b_{ij})_{i,j\in[n]}$ define $(A \mid B)_v$ to be the matrix with the columns of A at position $j \in v$ and the columns of B otherwise, i.e.

$$((A \mid B)_v)_{ij} = \left\{ \begin{array}{ll} a_{ij} & j \in v \\ b_{ij} & j \notin v \end{array} \right. . \tag{3.6}$$

For $x \in \mathbb{R}^v$ and $y \in \mathbb{R}^{[n]\setminus v}$, we get

$$(A \mid B)_v(x+y) = Ax + By.$$

Let $\mathbb{I}$ represent the identity matrix (of the appropriate dimension). Hence, for $w \in \mathbb{R}^v$ and $z \in \mathbb{R}^{[n]\setminus v}$, we want to solve:

$$q = w - Mz = (\mathbb{I} \mid -M)_v(w+z).$$

The determinant of $(\mathbb{I} \mid -M)_v$ equals (up to its sign) the determinant of the $[n] \setminus v$-minor of M. Thus, for a P-matrix, the system of linear equations has a unique solution, namely

$$x(v) = (\mathbb{I} \mid -M)_v{}^{-1} q.$$

As w and z are complementary one can extract w and z from x by setting $w_j = x(v)_j$, $z_j = 0$ for $j \in v$ and $z_j = x(v)_j$, $w_j = 0$ otherwise.

Find $w, z \in \mathbb{R}^3$ with

$$\begin{aligned} w - \begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{pmatrix} z &= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \\ w &\geq 0 \\ z &\geq 0 \\ w^T z &= 0 \end{aligned}$$

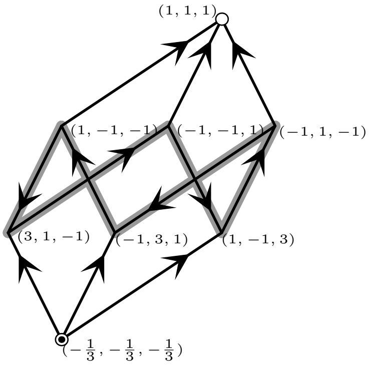


Figure 3.3: An LCP and its orientation. The vertices are labeled by their $x(v)$. The highlighted edges form a cycle.

Since $x(v)$ is unique for all v's, a strategy for solving LCP(M,q) would be to guess v and calculate $x(v)$. If $x(v)$ is a non-negative vector this solves the LCP. If not, proceed with a different v.

Definition 3.5
For a P-matrix $M \in \mathbb{R}^{n\times n}$ and $q \in \mathbb{R}^n$, define an orientation on $\mathfrak{C}^n$ by

$$v \to v \oplus \{\lambda\} \iff ((\mathbb{I} \mid -M)_v^{-1} q)_\lambda < 0. \tag{3.7}$$

As we just discussed, a sink of this orientation would yield the solution of $LCP(M,q)$. Also, the matrix $(\mathbb{I} \mid -M)_v{}^{-1}$, and therefore the orientation in a vertex, can be computed in polynomial time. In other words, if the orientation is a USO this introduces a polynomial-time unique sink oracle for linear complementarity problems. See Figure 3.3 for an example of a cyclic USO defined by an LCP.

Theorem 3.6 ([40])
For a P-matrix $M \in \mathbb{R}^{n\times n}$ and $q \in \mathbb{R}^n$ the orientation in Definition 3.5 is a unique sink orientation of $\mathfrak{C}^n$.

The crucial observation for the proof of Theorem 3.6 is that a subcube $[I,J]$ of $\mathfrak{C}^n$ corresponds to the problem of finding $w \in \mathbb{R}^J$ and $z \in \mathbb{R}^{[n]\setminus I}$

with $w - Mz = q$, $w^T z = 0$ and $w_i, z_i \geq 0$ for $i \in J \setminus I$, which again is a linear complementarity problem. For details see [40].

3.3 Strong LP-type Problems

Let us look once more at the reduction from LCP's to USOs, now from quite far away. The reduction associates to each subcube $[I, J]$ of $\mathfrak{C}^n$ a problem by strengthening/weakening constraints not in $J \setminus I$. Namely, for $i \notin J$ we require $w_i = 0$ but relax $z_i \geq 0$ and for $i \in I$ we require $z_i = 0$ and relax $w_i \geq 0$. The key observation is that for $I = J$, i.e., for a vertex of $\mathfrak{C}^n$, the corresponding problem is a system of linear equations and can be solved easily.

The goal of this section is to work out a framework under which similar reductions for optimization problems over the non-negative orthant yield a USO. Motivated by the LCP-reduction, a possible attempt to transform such a problem to a USO is the following: To sets $I \subseteq J \subseteq [d]$ (i.e., faces $[I, J]$) associate a domain (by strengthening/weakening the positivity constraints not in $J \setminus I$) and solve the optimization problem over this domain. Then compare the solutions. We do this in the hope that the case $I = J$ again is easy. This way we assign to a face given by $I \subseteq J \subseteq [d]$ a value $w(I, J)$. On an abstract level we order the set of faces.

Definition 3.7
Let $(O, \leq)$ be a poset and w a mapping from the pairs (I, J) of sets $I \subseteq J \subseteq [d]$ to O. Then w is called monotone *if for all $I \subseteq J \subseteq [d]$ and $I' \subseteq J' \subseteq [d]$*

$$I \subseteq I' \text{ and } J \subseteq J' \quad \Rightarrow \quad w(I, J) \leq w(I', J'). \tag{3.8}$$

It is called local *if for all $I_1 \subseteq J_1 \subseteq [d]$ and $I_2 \subseteq J_2 \subseteq [d]$*

$$\begin{aligned} &w(I_1, J_1) = w(I_2, J_2) \\ &\quad \iff w(I_1 \cap I_2, J_1 \cap J_2) = w(I_1 \cup I_2, J_1 \cup J_2). \end{aligned} \tag{3.9}$$

If w is monotone and local, the tuple $(d, w, O, \leq)$ is called a strong LP-type problem. *The* value *of a strong LP-type problem $(d, w, O, \leq)$ is $w(\emptyset, [d])$.*

Monotonicity for w does *not* refer to the face structure of $\mathfrak{C}^d$. For $I \subseteq J \subseteq [d]$ and $I' \subseteq J' \subseteq [d]$ the face $[I, J]$ is a subface of $[I', J']$ if and only if $I' \subseteq I \subseteq J \subseteq J'$. In contrast, w is monotone with regard to the component-wise inclusion relation on pairs of sets

$$(I, J) \subseteq (I', J') \iff I \subseteq I' \text{ and } J \subseteq J'. \tag{3.10}$$

For monotone w's the $\Leftarrow$-direction of locality is already satisfied. For $I_1 \subseteq J_1 \subseteq [d]$ and $I_2 \subseteq J_2 \subseteq [d]$, obviously (for $k = 1, 2$)

$$(I_1 \cap I_2, J_1 \cap J_2) \subseteq (I_k, J_k) \subseteq (I_1 \cup I_2, J_1 \cup J_2)$$

and therefore, by monotonicity,

$$\begin{aligned} w(I_1 \cap I_2, J_1 \cap J_2) \leq w(I_k, J_k) &\leq w(I_1 \cup I_2, J_1 \cup J_2) \\ &= w(I_1 \cap I_2, J_1 \cap J_2). \end{aligned}$$

In particular, all four values are equal. Thus, for strong LP-type problems we can (and will) use the following form of locality:

$$\begin{aligned} w(I_1, J_1) = w(I_2, J_2) &\implies \\ w(I_1 \cap I_2, J_1 \cap J_2) &= w(I_1, J_1) \\ &= w(I_2, J_2) = w(I_1 \cup J_1, I_2 \cup J_2). \end{aligned} \tag{3.11}$$

In general, the optimization problem corresponding to the value of a strong LP-type problem will be difficult to solve. But for a vertex v the value $w(v, v)$ can be found easily. A generic case for an LP-type problem is to optimize some function f over the non-negative orthant. Subcubes $[I, J]$ correspond to a strengthening of the positivity constraints on coordinates not in J and a weakening on coordinates in I. In other words, we drop all conditions on x_i, $i \in I$ and require $x_j = 0$ for $j \notin J$. For $I = \emptyset$ and $J = [d]$, we get the original problem over the non-negative orthant.

For $I = v = J$ the problem simplifies to optimizing f over $\mathbb{R}^v$. Thus, after restriciting f to $\mathbb{R}^v$, we have to solve an unconstrainted optimization problem. For a well-behaving function f, this can be solved efficiently. Hence, the aim of an abstract strong LP-type problem is to find a vertex v with $w(v, v) = w(\emptyset, [d])$.

Definition 3.8
Let $(d, w, O, \leq)$ be a strong LP-type problem and $I \subseteq J \subseteq [d]$. A vertex v, $I \subseteq v \subseteq J$, is called a basis *of $[I, J]$ with respect to w if $w(v, v) = w(I, J)$.*

We want to find a basis of $[I, J]$ to determine $w(I, J)$. In consequence, even if we would find a basis we could not verify it by its definition. A local condition is needed and can be provided by locality of w. Since $w(v, v) = w(I, J)$ for a basis v, (3.11) yields

$$w(I, v) = w(I \cap v, J \cap v) = w(I \cup v, J \cup v) = w(v, J).$$

Furthermore, by monotonicity the w-value of an edge $[v, v \cup \{\lambda\}]$ with $\lambda \in J \setminus v$ is sandwiched between $w(v, v)$ and $w(v, J) = w(v, v)$ and the w-value of $[v \setminus \{\mu\}, v]$, $\mu \in v \setminus I$ is sandwiched between $w(I, v)$ and $w(v, v) = w(v, J)$. Therefore, all these values are equal and v is a basis of all its incident edges in $[I, J]$.

Lemma 3.9
For a strong LP-type problem $(d, w, O, \leq)$, a vertex v is a basis of a subcube $[I, J]$ if and only if it is a basis of all edges $[v \setminus \{\lambda\}, v \cup \{\lambda\}]$, $\lambda \in J \setminus I$.

Proof. As we just argued for a basis v of $[I, J]$, the incident edges (i.e., all $\{v \oplus \{\lambda\}, v\}$, $\lambda \in J \setminus I$) have basis v.

Now assume v is a basis of its incident edges in $[I, J]$. We will distinguish between edges towards subsets and edges towards supersets of v and show that $w(I, v) = w(v, v) = w(v, J)$. But then, by monotonicity,

$$w(v, v) = w(I, v) \leq w(I, J) \leq w(v, J) = w(v, v)$$

and v is a basis of $[I, J]$.

For the equation $w(v, v) = w(I, v)$ we have to show that v is a basis of $[v \setminus \{\lambda\}, v]$ for $\lambda \in v \setminus I$. Let $v \setminus I = \{\lambda_1, \ldots \lambda_k\}$. By induction on k and locality, the equations $w(v, v) = w(v \setminus \{\lambda_i\}, v)$ imply

$$w(v, v) = w(v \setminus \{\lambda_1, \ldots, \lambda_i\}, v),$$

hence $w(v, v) = w(v \setminus (v \setminus I), v) = w(I, v)$. The same arguments for the edges $[v, v \cup \{\lambda\}]$, $\lambda \in J \setminus v$ proves $w(v, v) = w(v, J)$. $\square$

So far, we cannot guarantee the existence of a basis. But, what if there is a basis v of $[I, J]$? If we divide $[I, J]$ along a label $\lambda \in J \setminus I$ then the λ-facet of $[I, J]$ containing v must have the same w-value as $[I, J]$. This is a direct consequence of Lemma 3.9. Therefore, a basis has a chance to exist mainly in strong LP-type problems of the following type:

Definition 3.10
A strong LP-type problem $(d, w, O, \leq)$ is called reducible *if for any two sets $I \subseteq J \subseteq [d]$ and $\lambda \in J \setminus I$, the value $w(I, J)$ is attained in the λ-facets of $[I, J]$, i.e.*

$$w(I, J) \in \{w(I \cup \{\lambda\}, J), w(I, J \setminus \{\lambda\})\}.$$

If w is reducible then recursively we can trace the w-value of a cube $[I, J]$ over facets, ridges and so forth down to vertices. Thus, reducible strong LP-type problems have bases. Furthermore, for two bases v_1 and v_2 of $[I, J]$, by definition, $w(v_1, v_1) = w(I, J) = w(v_2, v_2)$. Therefore, by locality in the form of (3.11) we can conclude that $w(v_1 \cap v_2, v_1 \cap v_2) = w(v_1, v_1) = w(I, J)$ and $v_1 \cap v_2$ is a basis. Hence, the inclusion-minimal basis is unique. This proves the following lemma.

Lemma 3.11
Let $(d, w, O, \leq)$ be a reducible strong LP-type problem and $I \subseteq J \subseteq [d]$. Then there is a unique inclusion-minimal basis of $[I, J]$.

For a reducible strong LP-type problem $(d, w, O, \leq)$ the value of an edge $[v, v \cup \{\lambda\}]$ is either equal to $w(v, v)$ or to $w(v \cup \{\lambda\}, v \cup \{\lambda\})$. Furthermore, by monotonicity

$$w(v, v) \leq w(v, v \cup \{\lambda\}) \leq w(v \cup \{\lambda\}, v \cup \{\lambda\}).$$

It is possible that both v and $v \cup \{\lambda\}$ are a basis. But v is the inclusion-minimal basis of $[v, v \cup \{\lambda\}]$ if and only if it is a basis. If v is not basis, then $w(v, v) < w(v, v \cup \{\lambda\})$ and $v \cup \{\lambda\}$ is the inclusion-minimal basis. Thus, if we orient the cube such that an edge points towards its inclusion-minimal basis then $v \to v \cup \{\lambda\}$ hods if and only if $w(v, v) < w(v, v \cup \{\lambda\})$. This yields a unique sink orientation.

Theorem 3.12
Let $(d, w, O, \leq)$ be a reducible strong LP-type problem. Then the orientation of $\mathfrak{C}^d$ defined by

$$v \to v \cup \{\lambda\} \iff w(v, v) < w(v, v \cup \{\lambda\})$$

is a unique sink orientation. Furthermore, the sink of a subcube $[I, J]$ is a basis of $[I, J]$.

Proof. We orient the cube in such a way that edges point towards their inclusion-minimal basis. Given a subcube $[I, J]$ of $\mathfrak{C}^d$ and a sink $o \in [I, J]$, then o is inclusion-minimal basis of all its incident edges in $[I, J]$. In particular, by Lemma 3.9, o is a basis of $[I, J]$. It is also inclusion-minimal as basis of $[I, J]$: If o was not inclusion-minimal we would find a basis $o' \subseteq o$ with $w(o', o') = w(o, o)$. By locality we can assume that o' and o differ in only one element. Hence, o would not be an inclusion-minimal basis of the edge $\{o', o\}$.

On the other hand, by Lemma 3.9, an inclusion-minimal basis v of $[I, J]$ is a basis of all its incident edges. If v is not inclusion-minimal on one edge, say $\{v \oplus \{\lambda\}, v\}$, then λ has to be in v and $w(v \oplus \{\lambda\}, v \oplus \{\lambda\}) = w(v, v)$. But then $v \oplus \{\lambda\}$ is also a basis of $[I, J]$ and v not inclusion-minimal.

In conclusion each sink in $[I, J]$ is a inclusion-minimal basis of $[I, J]$. By Lemma 3.11 $[I, J]$ has exactly one such basis, i.e., $[I, J]$ has a unique sink. $\square$

In order to find the orientation in a vertex we need to compute if $w(v, v) < w(v, v \oplus \{\lambda\})$ for all $\lambda \in [n]$. This can also be done by evaluating $w(v \oplus \{\lambda\}, v \oplus \{\lambda\})$. Thus, a vertex evaluation in the USO of Theorem 3.12 corresponds to at most $d + 1$ evaluations of the form $w(v, v)$.

For the remainder of this section, we study the reverse question: For which USO s can we find a reducible strong LP-type problem inducing s? Let s be a USO on $\mathfrak{C}^d$ induced by some reducible strong LP-type problem $(d, w, O, \leq)$. The w-value of a subcube $[I, J]$ of $\mathfrak{C}^d$ is given by the w-value of the sink of $[I, J]$. Denote with $\sigma(I, J)$ the sink of such a subcube $[I, J]$. Then s is induced by w if and only if

$$w(I, J) = w(\sigma(I, J), \sigma(I, J)). \tag{3.12}$$

We can interpret σ as a mapping from the pairs of sets $I \subseteq J \subseteq [d]$ to $\mathrm{V}(\mathfrak{C})$. What if we set $w = \sigma$? Can $(d, \sigma, \mathrm{V}(\mathfrak{C}), \subseteq)$ be a reducible strong LP-type problem?

For any s, the function σ is reducible. For a subcube $[I, J]$ and a label $\lambda \in [I, J]$, its sink $o = \sigma(I, J)$ is either in the lower or the upper λ-facet of $[I, J]$ and in this facet a sink. But then the σ-value of this facet is $o = \sigma(I, J)$.

Also σ is local. Let $[I_1, J_1]$ and $[I_2, J_2]$ be two subcubes which have the same sink $o = \sigma(I_1, J_1) = \sigma(I_2, J_2)$. This sink is vertex of both subcubes. In particular, $I_1 \cup I_2 \subseteq v \subseteq J_1 \cap J_2$ and o is vertex in $[I_1 \cap I_2, J_1 \cap J_2]$ as well as in $[I_1 \cup I_2, J_1 \cup J_2]$. We have to show that o is sink in both subcubes.

A label $\lambda \in (J_1 \cap J_2) \setminus (I_1 \cap I_2)$ is not in I_1 or not in I_2. In the first case λ is in $J_1 \setminus I_1$, thus the edge $\{o \oplus \{\lambda\}, o\}$ is directed towards o as $\sigma(I_1, J_1) = o$. The second case is analogous. This shows that $\sigma(I_1 \cap I_2, J_1 \cap J_2) = o$. Similarly, a label $\lambda \in (J_1 \cup J_2) \setminus (I_1 \cup I_2)$ is in J_1 or J_2. Hence, $\lambda \in J_1 \setminus I_1$ or $J_2 \setminus I_2$ and the λ-edge in o is incoming. In consequence, $\sigma(I_1 \cup I_2, J_1 \cup J_2) = o$.

As Figure 3.4 shows, σ does not have to be monotone. On the other hand, we will not find $I \subseteq J \subseteq [d]$ and $I' \subseteq J' \subseteq [d]$ with $(I, J) \subseteq (I', J')$ such that $\sigma(I', J')$ is proper subset of $\sigma(I, J)$. That is, σ is not directly contradicting monotonicity: Let $I \subseteq J \subseteq [d]$ and $I' \subseteq J' \subseteq [d]$ with $(I, J) \subseteq (I', J')$ and $\sigma(I', J') \subseteq \sigma(I, J)$, then $\sigma(I', J') = \sigma(I, J)$. Observe that

$$I \subseteq I' \subseteq \sigma(I', J') \subseteq \sigma(I, J) \subseteq J,$$

thus $o = \sigma(I, J)$ and $o' = \sigma(I', J')$ are both in the subcube $[I', J]$. Since $J \setminus I'$ is a subset of $J' \setminus I'$ as well as $J \setminus I$, for any $\lambda \in J \setminus I'$ the λ-edge incident to o as well as the λ-edge incident to o' are incoming. Hence, o and o' are sink of $[I', J]$ which can only be the case if $o = o'$.

We just argued that for any USO s the tuple $(d, \sigma, \mathrm{V}(\mathfrak{C}), \subseteq)$ is nearly a reducible strong LP-type problem. It fails to be monotone only because of some missing relations. Namely, for $I \subseteq J \subseteq [d]$ and $I' \subseteq J' \subseteq [d]$ with $(I, J) \subseteq (I', J')$ the sets $\sigma(I', J')$ and $\sigma(I, J)$ have to be comparable. If we add all resulting relations to the subset-relation and still have a poset, we get a reducible strong LP-type problem. Otherwise, we fail not only for σ, i.e., no other $(d, w, O, \leq)$ will have s as its USO.

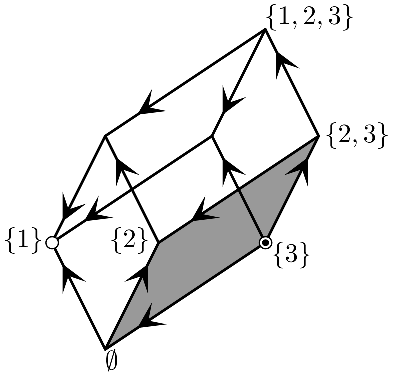


Figure 3.4: The map σ might be not monotone. The highlighted subcube $[\emptyset, \{2,3\}]$ has sink $\{2\}$ whereas the whole cube has sink $\{1\}$.

Proposition 3.13
A unique sink orientation on $\mathfrak{C}$ with outmap s is induced by a reducible strong LP-type problem if and only if the digraph on $\mathrm{V}(\mathfrak{C})$

$$u \rightsquigarrow v \iff u \setminus v \subseteq [d] \setminus (s(u) \cup s(v)) \tag{3.13}$$

is acyclic except for loops.

Proof. Let $\preceq$ be the transitive closure of $\rightsquigarrow$, i.e.

$$u \preceq v \iff \exists u_1, \dots, u_k : u \rightsquigarrow u_1 \rightsquigarrow \cdots \rightsquigarrow u_k \rightsquigarrow v.$$

Since for $u \subseteq v$ the set $u \setminus v$ is empty, $\rightsquigarrow$ and therefore $\preceq$ is a refinement of $\subseteq$. In particular, $\preceq$ is reflexive. It is transitive by definition. Finally, it is antisymmetric if and only if $(\mathrm{V}(\mathfrak{C}), \rightsquigarrow)$ has no other cycles except loops.

If the relation $\preceq$ is antisymmetric, $(d, \sigma, \mathrm{V}(\mathfrak{C}), \preceq)$ can be shown to be a reducible strong LP-type problem. As we argued above, σ is already reducible and local. For monotonicity, let $(I, J) \subseteq (I', J')$, $J' \subseteq [d]$ and $o = \sigma(I, J)$, $o' = \sigma(I', J')$. We have to show that $o \preceq o'$. Since o is the sink in $[I, J]$ for $\lambda \in J \setminus I$ the λ-edge incident to o is incoming, i.e., $\lambda \notin s(o)$. Hence, $J \setminus I \subseteq [d] \setminus s(o)$. Similarly, $J' \setminus I' \subseteq [d] \setminus s(o')$. Furthermore, since $o \subseteq J \subseteq J'$ and $I \subseteq I' \subseteq o'$, a $\lambda \in o \setminus o'$ is in J but not in I', we conclude

$$o \setminus o' \subseteq J \setminus I' \subseteq J \setminus I \cap J' \setminus I' \subseteq [d] \setminus s(o) \cap [d] \setminus s(o').$$

Thus, $o \rightsquigarrow o'$.

We just showed that $(d, \sigma, \mathrm{V}(\mathfrak{C}), \preceq)$ is a reducible strong LP-type problem if and only if $\preceq$ is a poset which is the case if and only if $\rightsquigarrow$ is acyclic besides loops. Furthermore, for an edge $\{v, v \cup \{\lambda\}\}$ the value $\sigma(v, v \cup \{\lambda\}) \neq v = \sigma(v, v)$ if and only if $v \to v \cup \{\lambda\}$. Hence, s is the orientation defined by $(d, \sigma, \mathrm{V}(\mathfrak{C}), \preceq)$.

On the other hand, let $(d, w, O, \leq)$ be a reducible strong LP-type problem which induces s as its USO. For the relation $\preceq$ defined by s let us first prove

$$u \preceq v \Rightarrow w(u, u) \leq w(v, v).$$

As $\preceq$ is the transitive closure of $\rightsquigarrow$, it is enough to show the statement for pairs $u \rightsquigarrow v$. For such a pair by definition $u \setminus v$ is a subset of $[d] \setminus s(u)$ and $[d] \setminus s(v)$. In particular, $u \setminus (u \cap v) = u \setminus v$ is disjoint from $s(u)$ and $(u \cup v) \setminus v = u \setminus v$ is disjoint from $s(v)$. Hence, u is a sink in $[u \cap v, u]$ and v is a sink in $[v, u \cup v]$. As s is the USO corresponding to w, by Theorem 3.12 and monotonicity of w we get

$$w(u, u) = w(u \cap v, u) \leq w(v, u \cup v) = w(v, v).$$

We will show that $\preceq$ is antisymmetric. Take three vertices u, v, v' with $u \rightsquigarrow v \preceq v' \preceq u$. As we just showed, for such u, v and v' the inequality

$$w(u, u) \leq w(v, v) \leq w(v', v') \leq w(u, u)$$

must hold. Therefore, $w(u, u) = w(v, v)$ and by locality the two values $w(u \cap v, u \cap v)$ and $w(u \cup v, u \cup v)$ are equal. Assume there is a $\lambda \in u \setminus v$. Since $u \rightsquigarrow v$, this label λ is not in $s(u)$. Thus, the edge $\{u \setminus \{\lambda\}, u\}$ is directed towards u. As the orientation s is defined by w the vertex u is a sink of $\{u \setminus \{\lambda\}, u\}$ if and only if $w(u \setminus \{\lambda\}, u \setminus \{\lambda\}) < w(u \setminus \{\lambda\})$. Hence

$$w(u \cap v, u \cap v) \quad \leq \quad w(u \setminus \{\lambda\}, u \setminus \{\lambda\})$$

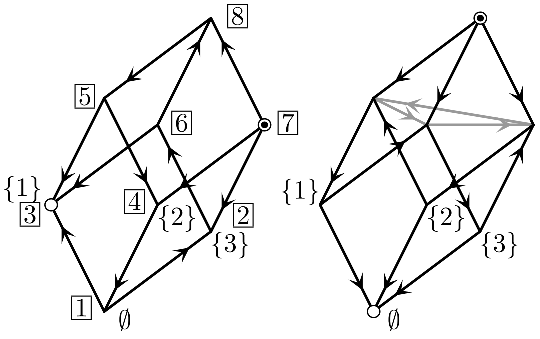


Figure 3.5: Being reducible strong LP-type problem is not preserved by isomorphism. Both pictures show (up to isomorphism) the same (cyclic) USO. In the picture to the left the boxed numbers define a strong LP-type problem via $w(I, J) = \boxed{\sigma(I, J)}$. In the picture to the right the shaded arrows form a cycle $\{1, 2\} \rightsquigarrow \{1, 3\} \rightsquigarrow \{2, 3\} \rightsquigarrow \{1, 2\}$.

$$\begin{aligned} &< w(u \setminus \{\lambda\}, u) \\ &\leq w(u \cup v, u \cup v) \\ &= w(u \cap v, u \cap v), \end{aligned}$$

a contradiction. Thus, $u \setminus v = \emptyset$ and because of symmetry $v \setminus u$. But then u and v have to be equal. This proves that $\preceq$ is antisymmetric. Hence, $\rightsquigarrow$ has to be acyclic besides loops. □

Figure 3.5 shows two examples. For the USO on the left side $\rightsquigarrow$ is acyclic. It therefore can be defined by a reducible strong LP-type problem. In the USO on the right side we have $\{1, 2\} \rightsquigarrow \{2, 3\} \rightsquigarrow \{1, 3\} \rightsquigarrow \{1, 2\}$. Hence, this USO cannot be achieved by a reducible strong LP-type problem. The two USOs are isomorphic, i.e., whether or not a USO comes from a reducible strong LP-type problem is *not* preserved under isomorphism.

3.4 Strictly Convex (Quadratic) Programming

A function $f : D \to \mathbb{R}$ over a convex set $D \subseteq \mathbb{R}^d$ is strictly convex if for any $x, y \in D$ and $\theta \in [0, 1]$ the inequality

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y)$$

holds with equality only for $\theta = 0, 1$. Laxly speaking, the line segment connecting two points on the curve $\{(x, f(x)) \mid x \in D\}$ is always above the curve. If a strictly convex function f attains its infimum over D, it does so in a unique minimizer $x^*(D)$. If there would be two minimizers x_1 and x_2 in D the line between these two would be above the curve of f and therefore either $f(x_1)$ or $f(x_2)$ is not minimal.

In the following we will restrict to continuously differentiable strictly convex functions f. For such f the gradient

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_d}\right)$$

not only exists but determines minimizers:

Lemma 3.14 ([35, Chapter 2, Exercise 10])
Let $D \subseteq \mathbb{R}^d$ be a convex set and $f : D \to \mathbb{R}$ a continuously differentiable strictly convex function. Then a point x^* minimizes f over D iff

$$\forall x \in D : \nabla f(x^*)(x - x^*) \geq 0. \tag{3.14}$$

For an inner point x^* the condition (3.14) is equivalent to the equation $\nabla f(x^*) = 0$. Thus, the statement is mainly interesting for boundary points. In particular, if D is not full-dimensional, every point in D is a boundary point.

Proof. For a unit vector $u \in \mathbb{R}^d$, $\|u\| = 1$, the (one-sided) directional derivative of f along u at a point $x \in D$ is

$$\nabla_u f(x) = \lim_{\epsilon \to 0^+} \frac{f(x + \epsilon u) - f(x)}{\epsilon}.$$

We consider only the limit from above, as the points we are interested in very well can be on the boundary of D. In particular, $\nabla_u f(x)$ is not defined for all u. The limit only exists if $x+\epsilon u \in D$ for sufficiently small ϵ.

By the chain rule applied to f and $g : t \mapsto x + tu$, the directional derivative and the gradient are connected by

$$\nabla_u f(x) = \nabla f(x) \cdot u.$$

We use directional derivatives to show that x^* is a minimizer of f. By scaling with $\|x - x^*\|$, (3.14) can be equivalently formulated as: $\nabla_u f(x^*) \geq 0$ for all unit vectors u for which it is defined. Here we need convexity of D. By convexity, $\nabla_u f(x^*)$ is defined if there exists some x with $x - x^* = \|x - x^*\|u$.

First assume there is a unit vector u with $\nabla_u f(x^*) < 0$. Then for sufficiently small ϵ, the vector $x^* + \epsilon u$ is in D and

$$\frac{f(x^* + \epsilon u) - f(x^*)}{\epsilon} < 0.$$

In particular, $x^* + \epsilon u$ proves that x^* is not minimal.

Now assume that x^* does not minimize f. Then there is a point $x \in D$ and some $\Delta > 0$ with $f(x) = f(x^*) - \Delta$. Since f and D are convex for any $\theta \in [0,1]$ the point $(1-\theta)x^* + \theta x$ is in D and has f-value

$$f((1-\theta)x^* + \theta x) \leq (1-\theta)f(x^*) + \theta f(x) = f(x^*) - \theta\Delta.$$

Hence, for all ϵ, $0 < \epsilon \leq \|x - x^*\|$ and $u = (x - x^*)/\|x - x^*\|$ we get

$$\frac{f(x^* + \epsilon u) - f(x^*)}{\epsilon} \leq -\frac{\Delta}{\|x - x^*\|} < 0,$$

which shows $\nabla_u f(x^*) < 0$. $\square$

The problem of strictly convex programming is to minimize a strictly convex function f over the non-negative orthant, i.e., we want to find x^* with

$$f(x^*) = \min\{f(x) \mid x \geq 0\} \tag{3.15}$$

which if exists, is unique. The main problem for such programs is caused by the non-negativity constraints. The unconstrained variant to minimize f (even over some linear subspace $\mathbb{R}^I$) is considerably easier and (for suitable f) can be solved using analytical methods.

Again we have the situation that the problem we are interested in is difficult, but modifying the constraints we end up with a solvable problem. This suggests to connect to strong LP-type problems. For $I \subseteq J \subseteq [d]$ consider the set

$$\mathcal{C}(I,J) := \left\{x \in \mathbb{R}^d \;\middle|\; x_i \geq 0 \text{ for } i \notin I,\, x_i = 0 \text{ for } i \notin J\right\}. \tag{3.16}$$

In other words, starting from the non-negative orthant we drop the conditions in I and strengthen the conditions outside J. Such a $\mathcal{C}(I,J)$ is a convex cone, i.e., it is a convex set such that for any $x \in \mathcal{C}(I,J)$ and a positive $\alpha \in \mathbb{R}$ the point $\alpha x \in \mathcal{C}(I,J)$. From that point of view $\mathcal{C}(I,J)$ is the smallest convex cone containing $\{-e_i \mid i \in I\} \cup \{e_i \mid i \in J\}$ and the origin (where e_i is the i-th unit vector). In particular, a cone $\mathcal{C}(I,J)$ is contained in $\mathcal{C}(I',J')$ if and only if $(I,J) \subseteq (I',J')$.

Theorem 3.15
Let $f : \mathbb{R}^d \to \mathbb{R}$ be a continuously differentiable strictly convex function, such that for any $I \subseteq J \subseteq [d]$, a unique minimizer $x^*(I,J)$ over $\mathcal{C}(I,J)$ exists. Then for $w : \{(I,J) \mid I \subseteq J \subseteq [d]\} \to \mathbb{R}^{d+1}$ with

$$w(I,J) = \Big(-f\big(x^*(I,J)\big), x^*(I,J)\Big) \tag{3.17}$$

the tuple $(d, w, \mathbb{R}^{d+1}, \leq_{\text{lex}})$ is a reducible strong LP-type problem.

Proof. For sets $I \subseteq J \subseteq [d]$ and $I' \subseteq J' \subseteq [d]$ with $(I,J) \subseteq (I',J')$ the cone $\mathcal{C}(I,J)$ is contained in $\mathcal{C}(I',J')$. As $x^*(I',J')$ is minimizing over a larger set than $x^*(I,J)$, for $y = f(x^*(I,J))$ and $y' = f(x^*(I',J'))$ the inequality $y' \leq y$ holds. Furthermore, if $y' = y$, uniqueness implies $x^*(I',J') = x^*(I,J)$, that is $w(I',J') = w(I,J)$. If on the other hand $y' < y$, then lexicographically $w(I,J) = (-y, \ldots)$ is smaller than $w(I',J') = (-y', \ldots)$. Hence, w is monotone.

For locality take sets $I_1 \subseteq J_1 \subseteq [d]$ and $I_2 \subseteq J_2 \subseteq [d]$ with $w(I_1,J_1) = w(I_2,J_2)$. In particular, the minimizer

$$x^*(I_1,J_1) = x^*(I_2,J_2) = x^*$$

is in both cones $\mathcal{C}(I_1,J_1)$ and $\mathcal{C}(I_2,J_2)$. But then $x^*_i \geq 0$ for $i \notin I_1 \cap I_2$ and $x^*_i = 0$ for $i \notin J_1 \cap J_2$, so $x^* \in \mathcal{C}(I_1 \cap I_2, J_1 \cap J_2)$. As x^* is minimizing the larger cone, $\mathcal{C}(I_1,J_1)$ this proves $x^*(I_1 \cap I_2, J_1 \cap J_2) = x^*$.

For $x^*(I_1 \cup I_2, J_1 \cup J_2) = x^*$ we use Lemma 3.14. Take some point $x \in \mathcal{C}(I_1 \cup I_2, J_1 \cup J_2)$. We decompose x into two parts $\pi_1 x \in \mathcal{C}(I_1, J_1)$ and $\pi_2 x \in \mathcal{C}(I_2, J_2)$. For such decomposition of x we get by Lemma 3.14

$$\begin{aligned}
\nabla f(x^*)(x - x^*) &= \nabla f(x^*)(\pi_1 x - x^* + \pi_2 x - x^* + x^*) \\
&= \nabla f(x^*)(\pi_1 x - x^*) + \nabla f(x^*)(\pi_2 x - x^*) + \\
& \nabla f(x^*)(2x^* - x^*) \\
&\geq 0
\end{aligned}$$

since x^* is optimal in $\mathcal{C}(I_i, J_i)$ and $2x^* \in \mathcal{C}(I_1, J_1)$. Such decomposition can be achieved by e.g. the orthogonal projection π_1 onto $\mathbb{R}^{(J_1 \setminus J_2) \cup I_1}$ and $\pi_2 = \mathrm{id} - \pi_1$.

It remains to show that w is reducible. For $I \subseteq J \subseteq [d]$ and $\lambda \in J \setminus I$ the coordinate $x^*(I, J)_\lambda$ is either 0 or positive. In the first case $x^*(I, J) \in \mathcal{C}(I, J \setminus \{\lambda\})$ and optimal in this cone (since $\mathcal{C}(I, J \setminus \{\lambda\}) \subseteq \mathcal{C}(I, J)$). In the second case $x^* = x^*(I, J)$ is optimal in $\mathcal{C}(I \cup \{\lambda\}, J)$: If there would be a better point $\hat{x} \in \mathcal{C}(I \cup \{\lambda\}, J)$ then for sufficiently small ϵ the point $x_\epsilon = (1 - \epsilon)x^* + \epsilon \hat{x}$ would be in $\mathcal{C}(I, J)$ and $f(x_\epsilon) < f(x^*)$ by convexity, which contradicts optimality of x^*. $\square$

The proof of Theorem 3.15 only once referred to the d-dimensional suffix of w. If we only require that two subcubes have the same optimal value, locality fails. We have to assure that subcubes with the same w-value have the same minimizer. In that spirit, the last d coordinates of w are a (huge) symbolic perturbation.

Theorem 3.15 allows us to apply Theorem 3.12 and reduce a strictly convex program to a USO. In the next three corollaries we will give reformulations of the orientation in Theorem 3.12 in terms more suitable for strictly convex programs. In particular, we want to eliminate references to $w(v, v \cup \{\lambda\})$.

Corollary 3.16
Let $f : \mathbb{R}^d \to \mathbb{R}$ be a continuously differentiable strictly convex function, such that for any $I \subseteq J \subseteq [d]$ a unique minimizer $x^*(I, J)$ over $\mathcal{C}(I, J)$ exists. Then the orientation

$$v \to v \cup \{\lambda\} \iff x^*(v, v) \textit{ is not optimal in } \mathcal{C}(v, v \cup \{\lambda\})$$

defines a unique sink orientation on the cube $\mathfrak{C}^d$. Furthermore, for the sink o the point $x^*(o, o)$ minimizes f over the non-negative orthant.

Proof. By Theorem 3.15 the tuple $(d, w, \mathbb{R}^{d+1}, \leq_{\text{lex}})$ with w defined as in (3.17) is a reducible strong LP-type problem. Thus, by Theorem 3.12 the orientation $v \to v \cup \{\lambda\} \iff w(v, v) < w(v, v \cup \{\lambda\})$ defines a USO on $\mathfrak{C}^d$.

As $x^*(v, v \cup \{\lambda\})$ is minimizing over a larger set than $x^*(v, v)$, for the f-values $f(x^*(v, v)) \geq f(x^*(v, v \cup \{\lambda\}))$ holds. In the case $f(x^*(v, v)) = f(x^*(v, v \cup \{\lambda\}))$ the points $x^*(v, v)$ and $x^*(v, v \cup \{\lambda\})$ have to be equal, as they both minimize $\mathcal{C}(v, v \cup \{\lambda\})$. But then $w(v, v) = w(v, v \cup \{\lambda\})$.

In particular, $w(v, v) < w(v, v \cup \{\lambda\})$ is equivalent to the inequality $f(x^*(v, v)) > f(x^*(v, v \cup \{\lambda\}))$. The latter can only happen if $x^*(v, v)$ is not optimal in $\mathcal{C}(v, v \cup \{\lambda\})$. □

Corollary 3.16 implicitly still refers to $w(v, v \cup \{\lambda\})$. The following two corollaries only consider the knowledge we have in v. Let us first describe how to derive the orientation of an edge to a larger set.

Corollary 3.17
Let $f : \mathbb{R}^d \to \mathbb{R}$ be a continuously differentiable strictly convex function, such that for any $I \subseteq J \subseteq [d]$ a unique minimizer $x^*(I, J)$ over $\mathcal{C}(I, J)$ exists. Then the orientation

$$v \to v \cup \{\lambda\} \iff (\nabla f(x^*(v, v)))_\lambda < 0$$

defines a unique sink orientation on the cube $\mathfrak{C}^d$. Furthermore, for the sink o the point $x^*(o, o)$ minimizes f over the non-negative orthant.

Proof. Let $v \subseteq [d]$ and $\lambda \in [d] \setminus v$.

If $v \to v \cup \{\lambda\}$ by Corollary 3.16 $x^* = x^*(v, v)$ is not optimal in $\mathcal{C}(v, v \cup \{\lambda\})$. Thus, by Lemma 3.14 we find some $x \in \mathcal{C}(v, v \cup \{\lambda\})$ with $\nabla f(x^*)(x - x^*) < 0$. Since $x \in \mathcal{C}(v, v \cup \{\lambda\})$ the λ-entry $\alpha = x_\lambda$ is non-negative and $x' = x - \alpha e_\lambda$ is in $\mathcal{C}(v, v)$. Thus

$$\nabla f(x^*)(x' - x^*) + \alpha \nabla f(x^*) e_\lambda = \nabla f(x^*)(x - x^*) < 0.$$

By Lemma 3.14 applied to $\mathcal{C}(v, v)$ the first expresion $\nabla f(x^*)(x' - x^*)$ is non-negative. Hence, $(\nabla f(x^*))_\lambda < 0$.

Now assume $(\nabla f(x^*))_\lambda < 0$ and consider $x^* + e_\lambda \in \mathcal{C}(v, v \cup \{\lambda\})$. We get

$$\nabla f(x^*)(x^* + e_\lambda - x^*) = \nabla f(x^*) e_\lambda < 0$$

and by Lemma 3.14 x^* is not optimal in $\mathcal{C}(v, v \cup \{\lambda\})$.

In summary, x^* is not optimal in $\mathcal{C}(v, v \cup \{\lambda\})$ if and only if the λ-coordinate of $\nabla f(x^*)$ is negative and

$$v \to v \cup \{\lambda\} \iff \big(\nabla f(x^*(v,v))\big)_\lambda < 0.$$

□

For the edges towards smaller vertices it suffices to know the corresponding entry of $x(v)$.

Corollary 3.18
Let $f : \mathbb{R}^d \to \mathbb{R}$ be a continuously differentiable strictly convex function, such that for any $I \subseteq J \subseteq [d]$ a unique minimizer $x^*(I, J)$ over $\mathcal{C}(I, J)$ exists. Then the orientation

$$v \setminus \{\lambda\} \to v \iff x^*(v,v)_\lambda > 0$$

defines a unique sink orientation on the cube $\mathfrak{C}^d$. Furthermore, for the sink o the point $x^*(o,o)$ minimizes f over the non-negative orthant.

Proof. Again we want to know whether or not $x^*(v \setminus \{\lambda\}, v \setminus \{\lambda\})$ is optimal in $\mathcal{C}(v \setminus \{\lambda\}, v)$. Since $\mathcal{C}(v \setminus \{\lambda\}, v \setminus \{\lambda\}) \subseteq \mathcal{C}(v \setminus \{\lambda\}, v) \subseteq \mathcal{C}(v,v)$, we will distinguish the three cases $x^*(v,v)_\lambda > 0$, $x^*(v,v)_\lambda = 0$ and $x^*(v,v)_\lambda < 0$.

If $x^*(v,v)_\lambda \geq 0$ then $x^*(v,v)$ is also optimal in the smaller domain $\mathcal{C}(v \setminus \{\lambda\}, v)$. For $x^*(v,v)_\lambda$ the point $x^*(v \setminus \{\lambda\}, v \setminus \{\lambda\})$ cannot be optimal in $\mathcal{C}(v \setminus \{\lambda\}, v)$.

In the case $x^*(v,v)_\lambda = 0$ all three optimizers $x^*(v,v)$, $x^*(v \setminus \{\lambda\}, v)$, and $x^*(v \setminus \{\lambda\}, v \setminus \{\lambda\})$ have to coincide. In particular, $x^*(v \setminus \{\lambda\}, v \setminus \{\lambda\})$ is optimal in $\mathcal{C}(v \setminus \{\lambda\}, v)$.

Finally, if $x^*(v,v)_\lambda$ is negative, for any point $x \in \mathcal{C}(v \setminus \{\lambda\}, v)$ the segment between x and $x^*(v,v)$ intersects $\mathcal{C}(v \setminus \{\lambda\}, v \setminus \{\lambda\})$, say in the point $x_\theta = (1-\theta)x^*(v,v) + \theta x$. The function value $f(x_\theta)$ is by convexity bounded by $(1-\theta) f(x^*(v,v)) + \theta f(x)$. As $f(x) \geq f(x^*(v,v))$ we can bound $f(x_\theta) \leq f(x)$. In particular, the minimizer for $\mathcal{C}(v \setminus \{\lambda\}, v)$ has to be in $\mathcal{C}(v \setminus \{\lambda\}, v \setminus \{\lambda\})$. Hence, $x^*(v \setminus \{\lambda\}, v \setminus \{\lambda\})$ minimizes $\mathcal{C}(v \setminus \{\lambda\}, v)$.

In summary, $x^*(v \setminus \{\lambda\}, v \setminus \{\lambda\})$ is not optimal in $\mathcal{C}(v \setminus \{\lambda\}, v)$ if and only if $x^*(v, v)_\lambda > 0$. This proves the claim. $\square$

As an example, let us consider *strictly convex quadratic functions.* Such a function f is given by a positive definite matrix $Q \in \mathbb{R}^{d \times d}$, a vector $u \in \mathbb{R}^d$ and a scalar $\alpha \in \mathbb{R}$, namely

$$f(x) = x^T Q x + u^T x + \alpha.$$

A *strictly convex quadratic program* is the problem of finding the minimizer of a strictly convex quadratic function f over the non-negative orthant, i.e finding the unique x^* with

$$f(x^*) = \min \{ f(x) \mid x \geq 0 \}. \tag{3.18}$$

As the name indicates such f is strictly convex. Also it is continuously differentiable with

$$\nabla f(x) = x^T (Q^T + Q) + u^T. \tag{3.19}$$

If every cone $\mathcal{C}(I, J)$ has a minimizer strict convex quadratic programming can be transformed to a reducible strong LP-type by Theorem 3.15 and to a USO by Corollary 3.17 and Corollary 3.18.

Lemma 3.19
Given a positive definite matrix $Q \in \mathbb{R}^{d \times d}$, a vector $u \in \mathbb{R}^d$, a scalar $\alpha \in \mathbb{R}$, and $I \subseteq J \subseteq [d]$ then the function $f(x) = x^T Q x + u^T x + \alpha$ has a unique minimizer $x^*(I, J)$ in the cone $\mathcal{C}(I, J)$.

Proof. As we already noted since f is convex it is enough to show that there exists a minimizer in $\mathcal{C}(I, J)$.

Define $\beta = \inf \left\{ u^T x \mid \|x\| = 1 \right\}$ and $\gamma = \inf \left\{ x^T Q x \mid \|x\| = 1 \right\}$. Since the sphere $\{x \mid \|x\| = 1\}$ is compact both β and γ are not infinite and since Q is positive definite $\gamma > 0$. Now for $r \in \mathbb{R}$ and $x \in \mathbb{R}^d$ of norm one

$$f(rx) = r^2 x^T Q x + r u^T x + \alpha \geq r^2 \gamma + r\beta + \alpha.$$

In particular, for any $K > 0$ there exists a ball L_K around the origin such that $f(x) > K$ for $x \notin L_K$. Thus, for a large enough ball L around the origin

$$\inf\{f(x) \mid x \in \mathcal{C}(I,J)\} = \inf\{f(x) \mid x \in \mathcal{C}(I,J) \cap L\}.$$

Since $\mathcal{C}(I,J) \cap L$ is compact the infimum is attained. □

The advantage of strictly convex *quadratic* functions is that the gradient is a linear function. In particular, we can test the conditions in Corollary 3.17 and Corollary 3.18 efficiently.

Theorem 3.20
Let $f : \mathbb{R}^d \to \mathbb{R}$ be a strictly convex quadratic function. For $v \subseteq [d]$ let $x^*(v)$ be the unique minimizer of f over $\mathbb{R}^v$ and

$$s(v) = \{\lambda \in v \mid x^*(v)_\lambda \leq 0\} \cup \left\{\lambda \in [d] \setminus v \;\middle|\; \left(\nabla f(x^*(v))\right)_\lambda < 0\right\}. \tag{3.20}$$

Then $s : 2^{[d]} \to 2^{[d]}$ defines a polynomial-time unique sink oracle for the strictly convex quadratic program defined by f.

Proof. By Corollary 3.17 and Corollary 3.18, s is the outmap of the strong LP-type problem defined by f. It remains to show that a vertex evaluation can be done in polynomial time. To calculate $s(v)$ it is enough to find $x^*(v)$ and $\nabla f(x^*(v))$. Since $\mathcal{C}(v,v) = \mathbb{R}^v$ is a linear subspace by Lemma 3.14 $x^*(v)$ is the solution to

$$\begin{aligned} x_\lambda &= 0 \quad \text{for } \lambda \notin v \\ (\nabla f(x))_\lambda &= 0 \quad \text{for } \lambda \in v \end{aligned}$$

As ∇f is a linear function this is a system of linear equations and can be solved in polynomial time. Therefore, $x^*(v)$ and $\nabla f(x^*(v))$ can be determined in polynomial time. □

3.5 Linear Programming Revisited

As we explained in section 3.1, every linear program LP(A,b,c) which maximizes $c^T x$ over the polyhedron $\mathcal{P} = \mathcal{P}(A,b) = \{x \geq 0 \mid Ax = b\}$

induces a unique sink orientation on $\mathcal{P}$ as long as $\mathcal{P}$ is bounded. But only if $\mathcal{P}$ is a cube we are in the original setup of USOs on cubes. By now we collected the ingredients to show that we can define a USO capturing $\mathrm{LP}(A, b, c)$ for arbitrary $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$. We will do so by viewing a linear program as a limit of quadratic programs. The USO we define has nothing in common with the orientation defined in section 3.1. In particular, if we start with an LP over a combinatorial cube, the USO we get will be of much higher dimension than the original cube.

For the sake of simplicity fix some notation. In this section, A will be a matrix in $\mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. Furthermore, $\mathcal{P} := \mathcal{P}(A, b)$ is the polyhedron defined by $\{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$. It is the intersection of the affine space $\mathcal{H} := \mathcal{H}(A, b) = \{x \in \mathbb{R}^n \mid Ax = b\}$ with the non-negative orthant $\mathcal{C}^+ := \mathcal{C}(\emptyset, [n])$. Any face $\mathcal{F}$ of $\mathcal{P}$ can be described by

$$\mathcal{F} = \mathcal{F}(v) = \mathcal{P} \cap \mathbb{R}^v = \mathcal{H} \cap \mathbb{R}^v \cap \mathcal{C}^+$$

for a $v \subseteq [n]$. As we start with a general matrix A, all of the above sets are potentially empty. In the normal setting this is not a problem as $\emptyset$ is a polyhedron and therefore a face of any polyhedron. Only for $\mathcal{H} = \emptyset$ we would have lied calling $\mathcal{H}$ an affine subspace.

One might already guess that we construct a USO by setting up a correspondence between a vertex v of $\mathfrak{C}^n$ and $\mathcal{F}(v)$. For this to work it is crucial that for any $v \in \mathrm{V}(\mathfrak{C}^n)$ we assign $\mathcal{F}(v)$ some meaningful object. In other words, for all the vertices v for which we do not find a corresponding face in $\mathcal{P}$ we have to fake a face.

Consider the Gram matrix $A^T A$. It is a symmetric matrix and for any $x, y \in \mathbb{R}^n$ we know $\langle Ax, Ay \rangle = \langle A^T Ax, y \rangle$. The latter fact is enough to show

$$\ker A^T A = \ker A \tag{3.21}$$
$$\operatorname{Im} A^T A = \operatorname{Im} A^T \tag{3.22}$$
$$\ker A^T A \oplus \operatorname{Im} A^T A = \mathbb{R}^n. \tag{3.23}$$

We will use Gram matrices to fake faces. Given a vertex $v \subseteq [n]$ of $\mathfrak{C}^n$ we want to construct an affine subspace $\mathcal{H}(v)$ of $\mathbb{R}^v$ which expresses somehow the behavior of $\mathcal{P}$ relative to $\mathbb{R}^v$. If $\mathcal{P} \cap \mathbb{R}^v$ is non-empty, i.e.,

$\mathcal{F}(v)$ is a proper face, we want to capture $\mathcal{F}(v)$. In this case $\mathcal{H}(v)$ is best set to $\mathcal{H}(v) = \mathcal{H} \cap \mathbb{R}^v$. But what if $\mathcal{H} \cap \mathbb{R}^v$ is empty?

The set $\mathcal{H} \cap \mathbb{R}^v$ is given by the system of linear equations $Ax = b$ together with $x_i = 0$ for $i \notin v$. Using the notation from (3.6) for $x \in \mathbb{R}_v$ we get $Ax = Ax + 0x = (A \mid 0)_v x$. Thus, if we define

$$A_v := (A \mid 0)_v \tag{3.24}$$

we can write $\mathcal{H} \cap \mathbb{R}^v = \{x \in \mathbb{R}^v \mid A_v x = b\}$. So far, we just reformulated the problem. But for the Gram matrix $A_v^T A_v$ of A_v the set

$$\mathcal{H}(v) := \left\{x \in \mathbb{R}^v \;\middle|\; A_v^T A_v x = A_v^T b\right\} \tag{3.25}$$

is always an affine subspace of $\mathbb{R}^v$: By (3.22) $A_v^T b \in \operatorname{Im} A_v^T A_v$. Thus, we find some $x \in \mathbb{R}^d$ with $A_v^T A_v x = A_v^T b$. Since the non-v-entries of x do not affect the value of $A_v x$, we even find some $x \in \mathbb{R}^v$.

Furthermore, if $\mathcal{H} \cap \mathbb{R}^v \neq \emptyset$ we can write $\mathcal{H} \cap \mathbb{R}^v$ as $x_0 + \ker A_v$ for some $x_0 \in \mathcal{H} \cap \mathbb{R}^v$. This x_0 satisfies $A_v^T A_v x_0 = A_v^T b$. Thus, $\mathcal{H}(v) = x_o + \ker A_v^T A_v$. Since $\ker A_v^T A_v = \ker A_v$, in this case the two spaces $\mathcal{H}(v)$ and $\mathcal{H} \cap \mathbb{R}^v$ coincide.

In other words, for all $v \subseteq [n]$ for which $\mathcal{P} \cap \mathbb{R}^v$ is a non-empty face $\mathcal{F}(v) = \mathcal{H}(v) \cap \mathcal{C}^+$ holds. We call such v *feasible*. Whenever v is infeasible, i.e., $\mathcal{P} \cap \mathbb{R}^v = \emptyset$, the set

$$\mathcal{F}(v) := \mathcal{H}(v) \cap \mathcal{C}^+$$

will be called *fake face*. Even fake faces can be empty if $\mathcal{H}(v)$ does not contain a non-negative point. But $\mathcal{H}(v)$ is always a well-behaved affine subspace.

Definition 3.21
Let $A \subseteq \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. For $v \subseteq [n]$ set

$$\begin{aligned} \mathcal{H}(v) &:= \left\{x \in \mathbb{R}^v \;\middle|\; A_v^T A_v x = A_v^T b\right\} &(3.26)\\ \mathcal{F}(v) &:= \mathcal{H}(v) \cap \mathcal{C}^+. &(3.27) \end{aligned}$$

Furthermore, let $x(v)$ be the orthogonal projection of the origin to $\mathcal{H}(v)$ and $c(v)$ the orthogonal projection of c to $\ker A \cap \mathbb{R}^v$.

The vector $c(v)$ captures the influence of the objective function given by c inside $\mathcal{H}(v)$. That is, for two vectors $x_1, x_2 \in \mathcal{H}(v)$ the difference in the objective value according to c and according to $c(v)$ agrees:

$$\begin{aligned} cx_1 - cx_2 = c(x_1 - x_2) &= (c - c(v))(x_1 - x_2) + c(v)(x_1 - x_2) \\ &= c(v)x_1 - c(v)x_2. \end{aligned}$$

(Remember that $c - c(v)$ is orthogonal to $x_1 - x_2 \in \ker A_v^T A_v$.) In particular, for a face $\mathcal{F}(v)$ an optimal solution with respect to c can be found by considering $c(v)$ instead.

The point $x(v)$ is a canonical reference point for $\mathcal{H}(v)$. In particular, we can write

$$\mathcal{H}(v) = x(v) + \ker A \cap \mathbb{R}^v.$$

Both $x(v)$ and $c(v)$ can be computed efficiently. As $\operatorname{Im} A_v^T A_v$ is orthogonal to $\ker A_v^T A_v$ and thus orthogonal to $\mathcal{H}(v)$, $x(v)$ is the unique vector in $\operatorname{Im} A_v^T A_v \cap \mathcal{H}(v)$. Expressed by a system of linear equations a solution (x, y) to

$$\begin{pmatrix} A_v^T A_v & 0 \\ \mathbb{I} & -A_v^T A_v \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} A_v^T b \\ 0 \end{pmatrix}$$

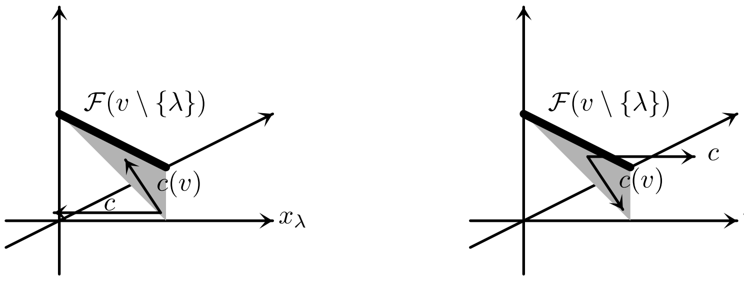
determines $x(v) = x$. Similarly, one can calculate $c(v)$. As $c(v)$ is the projection of c to $\ker A^T A$ a solution (x, y) to

$$\begin{pmatrix} A_v^T A_v & 0 \\ 0 & (A_v^T A_v)^2 \\ \mathbb{I} & A_v^T A_v \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ A_v^T c \\ c \end{pmatrix}.$$

yields $c(v) = x$.

Definition 3.22
For $A \subseteq \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^m$ define an orientation $\phi(A, b, c)$ on $\mathfrak{C}^n$ by directing the edge between $v \subseteq [n]$ and $v \setminus \{\lambda\}$, $\lambda \in v$ the following way:

$$v \setminus \{\lambda\} \to v \iff c(v)_\lambda > 0 \text{ or } \big(c(v)_\lambda = 0 \wedge x(v)_\lambda > 0\big).$$



$\mathcal{F}(v \setminus \{\lambda\})$
$c(v)$
c
x_λ
$\mathcal{F}(v \setminus \{\lambda\})$
c
$c(v)$
x_λ

Figure 3.6: The situation above illustrates the influence of $c(v)$. The gray area is $\mathcal{F}(v)$ and we look at the edge $\{v \setminus \{\lambda\}, v\}$. On the left side we direct $v \to v \setminus \{\lambda\}$ and on the right side $v \setminus \{\lambda\} \to v$

We direct $v \to v \setminus \{\lambda\}$ if and only if $c(v)_\lambda \leq 0$ and $c(v)_\lambda \neq 0 \vee x(v)_\lambda \leq 0$ which can be equivalently written as

$$v \setminus \{\lambda\} \leftarrow v \iff c(v)_\lambda < 0 \text{ or } \big(c(v)_\lambda = 0 \wedge x(v)_\lambda \leq 0\big).$$

The intuition behind this orientation is the following: Assume we have two feasible faces $\mathcal{F}(v \setminus \{\lambda\})$ and $\mathcal{F}(v)$. If $c(v)_\lambda > 0$ then no point in $\mathcal{F}(v \setminus \{\lambda\})$ can be optimal in $\mathcal{F}(v)$. On the other hand, if $c(v) < 0$ and $x \in \mathcal{F}(v)$, the intersection point of $\mathcal{F}(v \setminus \{\lambda\})$ with the half-line $\{x + \alpha c(v) \mid \alpha \geq 0\}$ exists and has a better objective value than x. Thus, optimality of $\mathcal{F}(v)$ is already attained in $\mathcal{F}(v \setminus \{\lambda\})$. See Figure 3.6.

A feasible v with $c(v) = 0$ potentially is optimal. If it is not optimal one of the $v \cup \{\lambda\}$ will reveal this. For instance, in the picture on the right side of Figure 3.6 $\mathcal{F}(v \setminus \{\lambda\})$ is not optimal, since $c(v)$ has a positive λ-coordinate. But c does not distinguish between optimal faces. A USO capturing the LP has to tag a special face among the set of all optimal faces and mark the corresponding v as sink. The orientation proposed in Definition 3.22 does so by taking $x(v)$ into account. The reasoning behind this is the following: First of all $x(v)$ is a generic point in $\mathcal{H}(v)$, in particular, it can be computed efficiently. But even more important,

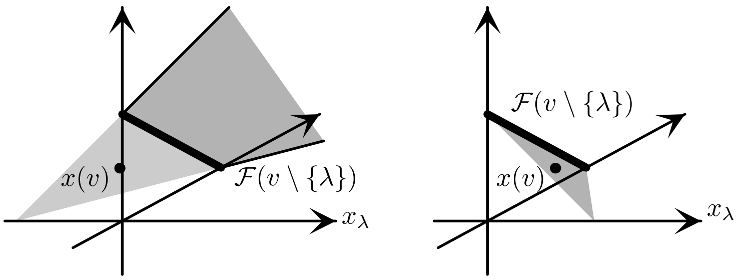

$x(v)$
$\mathcal{F}(v \setminus \{\lambda\})$
x_λ
$\mathcal{F}(v \setminus \{\lambda\})$
$x(v)$
x_λ

Figure 3.7: The situation above illustrates the influence of $x(v)$. The gray area is $\mathcal{F}(v)$. The vector c is assumed to be orthogonal to $\mathcal{H}(v)$. On the left side $x(v)$ has a negative λ-coordinate. Thus, $x(v)$ is not a feasible solution, but $x(v \setminus \lambda)$ is. We direct $v \to v \setminus \{\lambda\}$. On the right side $x(v)$ is feasible. In particular, the λ-entry of $x(v)$ is positive and we direct $v \setminus \{\lambda\} \to v$.

$x(v)$ is a potential optimizer. If we search for the face for which $x(v)$ is a non-negative point, the sink will not only reveal an optimal face but also a solution in the optimal face. See Figure 3.7.

Let us now state the main result of this section:

Theorem 3.23
For $A \subseteq \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^m$ the orientation $\phi(A, b, c)$ defines a unique sink orientation on $\mathfrak{C}^n$.

We do not claim that this orientation is a unique sink oracle! A minor obstacle is, that a vertex evaluation is only polynomial in n *and* m rather than in n alone. But the core problem lies in the fact, that the orientation has the unique sink property for arbitrary A, b and c, even if the original problem LP(A, b, c) is infeasible. Especially in the case of infeasible LP's we cannot provide a unique sink oracle. It is not clear to what "optimal" solution the sink should correspond to.

The main idea for the proof of Theorem 3.23 is to approximate the linear program by a sequence of strictly convex quadratic programs. Remember that for strictly convex quadratic programs the outmap of an edge is determined by the solution of a system of linear equations. In consequence, at some point we will need to consider sequences of systems of linear equations. The technical lemma in the background is the following:

Lemma 3.24
Let M be a symmetric matrix over $\mathbb{R}^d$, κ the orthogonal projection onto $\ker M$ and ι the orthogonal projection onto $\operatorname{Im} M$. Furthermore, let $y, z \in \mathbb{R}^d$. For $\epsilon > 0$ consider the system of linear equations

$$Mx + \epsilon^2 x = My + \frac{\epsilon}{2} z. \tag{3.28}$$

Then there is an $\epsilon_0 = \epsilon_0(M, y, z)$, such that for all ϵ, $0 < \epsilon < \epsilon_0$ there exists a unique solution $x^{(\epsilon)}$ to (3.28) and the sign pattern of $x^{(\epsilon)}$ at coordinate i is

$$\operatorname{sgn} x_i^{(\epsilon)} = \begin{cases} \operatorname{sgn}(\iota y)_i & \text{for } (\kappa z)_i = 0 \\ \operatorname{sgn}(\kappa z)_i & \text{otherwise} \end{cases} .$$

For the proof we will only need that $\mathbb{R}^d = \ker M \oplus \operatorname{Im} M$. This is the case for symmetric M.

Proof. As $\det(M + \epsilon^2 \mathbb{I})$ is a non-zero polynomial in ϵ there exists some $\epsilon' = \epsilon'(M)$, such that for ϵ with $\epsilon' > \epsilon > 0$ the matrix $M + \epsilon^2 \mathbb{I}$ is invertible. For these ϵ there exists a unique solution $x^{(\epsilon)}$ to

$$Mx + \epsilon^2 x = My + \frac{\epsilon}{2} z.$$

We will consider $\iota(x^{(\epsilon)})$ and $\kappa(x^{(\epsilon)})$ separately.

$$Mx^{(\epsilon)} + \epsilon^2 x^{(\epsilon)} = \underbrace{\epsilon^2 \kappa(x^{(\epsilon)})}_{\in \ker M} + \underbrace{M\iota(x^{(\epsilon)}) + \epsilon^2 x^{(\epsilon)}}_{\in \operatorname{Im} M}$$

For the κ-component since $\kappa M y = 0$ the equation is rewritten as

$$\epsilon^2 \kappa(x^{(\epsilon)}) = \kappa(My + \frac{\epsilon}{2} z) = \frac{\epsilon}{2} \kappa(z).$$

Hence, $\kappa(x^{(\epsilon)}) = \frac{\kappa(z)}{2\epsilon}$.

The ι-component case is a bit more involved. The corresponding equation reads:

$$M\iota(x^{(\epsilon)}) + \epsilon^2\iota(x^{(\epsilon)}) = \iota(My + \frac{\epsilon}{2}z) = M\iota(y) + \frac{\epsilon}{2}\iota(z)$$

(Since $\iota(My) = My = M(\kappa(y) + \iota(y)) = M\iota(y)$.) Basically, we have the same equation for $\iota(x^{(\epsilon)})$ as before projecting to $\operatorname{Im} M$ for $x^{(\epsilon)}$. The only difference is that now all vectors are in $\operatorname{Im} M$ which allows us to invert M. More precicely, consider the linear functions $g : \operatorname{Im} M \to \operatorname{Im} M$, $g(x) = Mx$ and $g_\epsilon : \operatorname{Im} M \to \operatorname{Im} M$, $g_\epsilon(x) = Mx + \epsilon^2 x$. Since $\ker M$ and $\operatorname{Im} M$ are orthogonal, g has a trivial kernel. Furthermore, the g_ϵ's are restrictions of bijections. Thus, all maps are invertible. Furthermore, $g_\epsilon \to g$ (as elements of the normed algebra of automorphisms on $\operatorname{Im} M$), thus $g_\epsilon^{-1} \to g^{-1}$. Applying this convergence to the equation above we get

$$\begin{aligned}\iota x^{(\epsilon)} &= g_\epsilon^{-1}(g_\epsilon(x^{(\epsilon)}))\\ &= g_\epsilon^{-1}(g(\iota y)) + \frac{\epsilon}{2} g_\epsilon^{-1}(\iota(z))\\ &\to g^{-1}(g(\iota(y))) + 0 = \iota(y).\end{aligned}$$

For a coordinate $i \in [n]$ with $\kappa(z)_i = 0$ the sign pattern of $x_i^{(\epsilon)}$ only depends on $\iota(x^{(\epsilon)})$. Since $\iota(x^{(\epsilon)}) \to \iota(y)$ we find an $\epsilon_i = \epsilon_i(M, y)$, such that for $\epsilon \in (0, \epsilon_i)$ the point $\iota(x^{(\epsilon)})$ has at most distance $|\iota(y)_i|/2$ from $\iota(y)$. In particular, $\operatorname{sgn} x_i^{(\epsilon)} = \operatorname{sgn} \iota(y)_i$.

For $i \in [n]$ with $\kappa(z)_i \neq 0$ we find an $\epsilon_i = \epsilon_i(M, y, z)$, such that for $\epsilon < \epsilon_i$ on the one hand $\|\iota(x^{(\epsilon)}) - \iota(y)\| < \frac{1}{2}$ and on the other hand $|(\frac{\kappa(z)}{2\epsilon})_i| > 1$. For such ϵ the sign pattern of $x_i^{(\epsilon)}$ is determined by $(\frac{\kappa(z)}{2\epsilon})_i$.

Collecting all ϵ_i and ϵ' for $\epsilon_0 = \min\{\epsilon', \epsilon_1, \dots, \epsilon_d\}$ the statement of the lemma holds. □

Proof of Theorem 3.23. As already indicated we will reduce the LP to a sequence of strictly convex quadratic programs. For $\epsilon > 0$ let $Q_\epsilon = A^T A + \epsilon^2 \mathbb{I}$. This matrix is positive definite since for $x \in \mathbb{R}^n$

$$x^T Q_\epsilon x = x^T A^T A x + \epsilon^2 x^T x = \|Ax\| + \epsilon^2\|x\|$$

which is non-negative and equals 0 only for the 0-vector. Therefore, the function

$$\begin{aligned} f_\epsilon(x) &= (Ax-b)^T(Ax-b) - \epsilon c^T x + \epsilon^2 x^T x \\ &= x^T(A^TA + \epsilon^2 \mathbb{I})x - (2b^TA + c^T)x + b^Tb \end{aligned}$$

is a strictly convex quadratic function. The idea behind f_ϵ is that for $\epsilon \to 0$ the part $(Ax-b)^T(Ax-b)$ dominates $\epsilon c^T x$ and the latter dominates $\epsilon^2 x^T x$. Thus, minimizing f_ϵ over $\mathcal{C}^+$ is similar to maximizing $c^T x$ over $\mathcal{P}$. In fact, we will show that for small ϵ the USO defined by f_ϵ equals $\phi(A,b,c)$.

To distinguish the USO defined by f_ϵ and $\phi(A,b,c)$ let us write $u \to_\epsilon v$ if u is directed to v according to f_ϵ. For $u = v \setminus \{\lambda\}$ according to Theorem 3.20

$$v \setminus \{\lambda\} \to_\epsilon v \iff (x^{(\epsilon)}(v))_\lambda > 0$$

where $x^{(\epsilon)}(v)$ is the unique minimizer of f_ϵ over $\mathbb{R}^v$, i.e., the solution to

$$\begin{aligned} x_i &= 0 \quad \text{for } i \notin v \\ (\nabla f_\epsilon(x))_i &= 0 \quad \text{for } i \in v. \end{aligned}$$

As the gradient is $\nabla f_\epsilon(x) = 2x^T(A^TA + \epsilon^2\mathbb{I}) - (2b^TA + \epsilon c^T)$ the equation system is equivalent to

$$A_v^T A_v x + \epsilon^2 x = A_v^T b + \frac{\epsilon}{2} c_v \tag{3.29}$$

over $\mathbb{R}^v$ (where $(c_v)_i = c_i$ for $i \in v$ and $(c_v)_i = 0$ otherwise).

Let $M = A_v^T A_v$, $y = x(v)$ and $z = c_v$. As $x^{(\epsilon)}(v)$ is the solution to $Mx + \epsilon^2 x = My + \epsilon/2z$, Lemma 3.24 yields some $\epsilon_0 = \epsilon_0(M,y,z) = \epsilon_0(A,v,c)$, such that for $\epsilon < \epsilon_0$ and $\lambda \in [n]$ the sign of $x^{(\epsilon)}(v)_\lambda$ is positive if and only if $(\kappa c_v)_\lambda > 0$, or $(\kappa c_v)_\lambda = 0$ and $(\iota x(v))_\lambda > 0$.

For the rest of the prove let $\epsilon < \epsilon_0$. Per definition $v \setminus \{\lambda\} \to_\epsilon v$ if and only if $x^{(\epsilon)}(v)_\lambda > 0$ for $\lambda \in v$. Hence,

$$v \setminus \{\lambda\} \to_\epsilon v \iff (\kappa c_v)_\lambda > 0 \text{ or } ((\kappa c_v)_\lambda = 0 \wedge (\iota x(v))_\lambda > 0).$$

It remains to show that $\kappa(c_v) = c(v)$ and $\iota(x(v)) = x(v)$. Since $x(v)$ is the minimal norm point in $\mathcal{H}(v)$, it is orthogonal to $\ker A_v^T A_v$, i.e., in $(\ker A_v^T A_v)^\perp = \operatorname{Im} A_v^T A_v$. This establishes $\iota(x(v)) = x(v)$.

For $\kappa(c_v) = c(v)$ it is enough to show that $c - \kappa(c_v)$ is orthogonal to $\ker A_v^T A_v$. In other words, we have to show that for all $x \in \ker A_v^T A_v$ the scalar products $\langle x, c\rangle$ and $\langle x, \kappa(c_v)\rangle$ are equal. We already know that $\langle x, c_v\rangle = \langle x, \kappa(c_v)\rangle$ (as $\kappa(c_v)$ is the orthogonal projection of c_v onto $\ker A_v^T A_v$) and $\langle x, c\rangle = \langle c, c_v\rangle$ (since c_v is the orthogonal projection of c onto $\mathbb{R}^v$ and $x \in \mathbb{R}^v$). Hence, the desired equality is shown. $\square$

With Theorem 3.23 we established a reduction from any $\mathrm{LP}(A, b, c)$ with n variables and m constraints to a USO of dimension n. Furthermore, the reduction is polynomial in n and m. But as long as we cannot relate the sink of the USO to an optimal solution of $\mathrm{LP}(A, b, c)$ the reduction is useless.

Proposition 3.25
If the linear program $\mathrm{LP}(A, b, c)$ *has an optimal solution then it attains its optimum in* $x(o)$ *of the unique sink* o *of* $\phi(A, b, c)$*.*

Proof. If the linear program has an optimal solution the set of all optimal solutions forms a face $\mathcal{F} = \mathcal{F}(v)$. Let x^* be the minimal norm point in $\mathcal{F}$ and $o := \{\lambda \mid x^*_\lambda \neq 0\}$. We prove that o is a (and therefore the) sink of $\phi(A, b, c)$ and $x^* = x(o)$.

Since $x^* \in \mathcal{H}(o) \cap \mathcal{P}$ we are in a feasible vertex. In particular, $\mathcal{F}(o)$ is a nonempty face of $\mathcal{P}$. Furthermore, $o \subseteq v$ and in consequence $\mathcal{F}(o)$ is a subface of $\mathcal{F} = \mathcal{F}(v)$.

Since x^* as an element of $\mathbb{R}^o$ is strictly positive for small ϵ the point $x^* + \epsilon c(o)$ is still in $\mathcal{F}(o)$. As a point in $\mathcal{F}(o) \subseteq \mathcal{F}$ the objective value of $x^* + \epsilon c(o)$ equals the value of x^*, so

$$0 = c^T(x^* + \epsilon c(o)) - c^T x^* = \epsilon c^T c(o) = \epsilon \|c(o)\|^2,$$

which shows $c(o) = 0$.

We use Lemma 3.14 to show that $x^* = x(o)$. Consider the norm function $g : x \mapsto \|x\|$. The gradient of g is $\nabla g = 2x^T$. For $x \in \mathcal{H}(o)$ we thus have to show that $2x^T(x - x^*) \geq 0$. Since x^* is strictly positive in $\mathbb{R}^o$ and $x \in \mathbb{R}^o$ we find some $\theta > 0$, such that $\theta(x - x^*)$ is strictly positive in $\mathbb{R}^o$. This $\theta(x - x^*)$ is in $\mathcal{F}(o)$. In $\mathcal{F}(o)$ we know that x^* is minimal norm point. Hence, $2\theta x^T(x - x^*) \geq 0$.

In particular, for all $\lambda \in o$ we know that $c(o)_\lambda = 0$ and $x(o)_\lambda > 0$, i.e., $o \setminus \{\lambda\} \to o$.

It remains to show that for $\lambda \in [n] \setminus o$ we have $o \cup \{\lambda\} \to o$. For this to happen either $c(o \cup \{\lambda\})_\lambda < 0$, or $c(o \cup \{\lambda\})_\lambda = 0$ and $x(o \cup \{\lambda\})_\lambda \leq 0$ has to hold. We will rule out all other sign patterns for $c(o \cup \{\lambda\})_\lambda$ and $x(o \cup \{\lambda\})_\lambda$.

First assume $c(o \cup \{\lambda\})_\lambda > 0$. Choose $\epsilon > 0$ small enough, such that $x^*_\mu + \epsilon c(o \cup \{\lambda\})_\mu$ is still non-negative for $\mu \in o$. For such ϵ the point $x^* + \epsilon c(o \cup \{\lambda\})$ is in $\mathcal{P}$. Furthermore

$$\begin{aligned} c^T(x^* + \epsilon c(o \cup \{\lambda\})) &= c^T x^* + \epsilon c^T c(o \cup \{\lambda\}) \\ &= c^T x^* + \epsilon \|c(o \cup \{\lambda\})\|^2 > c^T x^* \end{aligned}$$

in contradiction to x^* being optimal. Hence, $c(o \cup \{\lambda\})_\lambda \leq 0$.

Now assume $c(o \cup \{\lambda\})_\lambda = 0$ and $x(o \cup \{\lambda\})_\lambda > 0$. In this case $c(o \cup \{\lambda\})$ is not only in $\ker A \cap \mathbb{R}^{v \cup \{\lambda\}}$ but in $\ker A \cap \mathbb{R}^v$. Furthermore, $c - c(o \cup \{\lambda\})$ is orthogonal to $\ker A \cap \mathbb{R}^v$ as well as to $\ker A \cap \mathbb{R}^{o \cup \{\lambda\}}$. Thus, $c(o \cup \{\lambda\}) = c(o) = 0$. But then $\mathcal{F}(o \cup \{\lambda\})$ is an optimal face, i.e., subface of F. In particular, $x(v \cup \{\lambda\}) \in \mathcal{H}(o \cup \{\lambda\}) \subseteq \mathcal{H}(v)$. Since x^* is strictly positive in $\mathbb{R}^v$ and we assumed $x(o \cup \{\lambda\})_\lambda > 0$, for some $\theta \in [0,1]$ the convex combination $(1-\theta)x^* + \theta x(o \cup \{\lambda\})$ is still non-negative. Hence, $(1-\theta)x^* + \theta x(o \cup \{\lambda\}) \in \mathcal{F}$. Furthermore, $x(o \cup \{\lambda\})$ is of smaller norm than x^*, as $x^* \in \mathcal{H}(o \cup \{\lambda\})$ but $x^*_\lambda = 0$. In consequence, by convexity of the norm function $(1-\theta)x^* + \theta x(o \cup \{\lambda\})$ is a point of smaller norm than x^* in $\mathcal{F}$. This contradicts the choice of x^*. $\square$

In contrast to the first reduction from linear programming to USOs in this more general approach cycles can arise. See for example Figure 3.8. The LP in Figure 3.8 is highly degenerate since the equation $Ax = b$ has no positive solution. Ergo all faces are fake. In particular, the cycle is found among the fake faces. So far, no example is known with a cycle in the feasible region.

$$\begin{aligned} & \max x_1 && \text{such that} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 3 & -1 \\ 1 & -1 & 20 \end{pmatrix} x &= && \begin{pmatrix} 47 \\ -33 \\ 1 \end{pmatrix} \\ x &\geq && 0 \end{aligned}$$

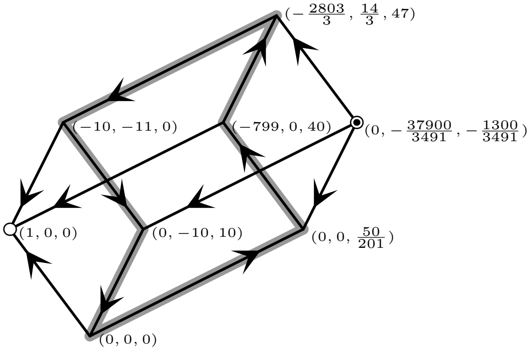


Figure 3.8: An LP with a cyclic USO. The matrix has full rank. Thus, all $\mathcal{H}(v)$ are points and $c(v) = 0$. In particular, $x(v)$ alone decides the orientation. The vertices are labeled by their $x(v)$. The highlighted edges form a cycle.

3.6 Remarks

Abstract objective functions were first introduced by Adler and Saigal [1]. They provide a general framework for the RANDOMFACET algorithm. How exactly linear programming relates to the larger class of abstract objective function is unknown up to now. In general, it is hard to decide whether or not an abstract objective function can be realized as linear program. The Holt-Klee condition is only a weak indication.

Recently, Develin [8] proved that the number of orientations induced by linear programs compared to the orientations fulfilling the Holt-Klee condition is vanishing as the dimension grows. His techniques can be applied to all reductions to USOs where the orientation is decided according to sign patterns of polynomials. In particular, all reductions in this thesis are of such type.

The reduction from linear complementarity problems is historically the first appearance of unique sink orientations. It was introduced by Stickney and Watson [40].

The concept of strong LP-type problems was introduced by Gärtner [13, 11] to capture the combinatorial structure of quadratic programs. It is called strong LP-type problem as by fixing one of the two parameters one gets an LP-type problem. (For LP-type problems see [15]). The definition of strong LP-type problems here is slightly more general than Gärtner's approach, as here we allow arbitrary posets $(O, \leq)$ whereas originally $O = \mathbb{R}$ was assumed.

As it turns out this generalization allows us to capture strictly convex quadratic programming as strong LP-type problem and thus is the more natural definition.

The idea to view a linear program as a limit process of strictly convex programs originates from Gärtner [13]. The reduction presented here is particularly nice as the resulting USO can be described in terms of the LP only.

The hope is that $\phi(A, b, c)$ can be shown to be USO without the detour via quadratic programming. In classical LP-theory one tends to assume that an LP has a unique solution. The orientation $\phi(A, b, c)$ has to deal with degenerate cases and assigns some solution even to unbounded or infeasible LP's. The key to fully understand $\phi(A, b, c)$ is a geometric view to these solutions.

In the unbounded case $\phi(A, b, c)$ answers a feasible ray as witness for the unboundedness. Which ray? And how can one distinguish between infeasible and unbounded cases? Both questions remain unanswered.

4 Structure

Viertes Geschoss: Hier wohnt der Architekt. Er geht auf in seinem Plan. Dieses Gebäude steckt voller Ideen! Es reicht von Funda- bis Firmament, Und vom Fundament bis zur Firma.

(Einstürzende Neubauten)

4.1 Outmaps

As we saw in the previous chapter, in the context of the optimization problems that give rise to unique sink orientations of cubes, it is often more adequate to look at an orientation in terms of its outmap

In the following, we will mostly take this latter point of view. However, the difference is linguistic, rather than structural and we do not strictly distinguish between an actual unique sink orientation and its outmap. In particular, we use the convenient shorthand USO for both.

The question then arises how one could discriminate unique sink outmaps from ordinary maps. Simply using the definition for any subcube $[I, J]$ one needs to check wether or not there is a vertex $I \subseteq v \subseteq J$ which is a sink in $[I, J]$, i.e., for which the outmap satisfies $s(v) \cap J \backslash I = \emptyset$. But as existence is in general difficult to verify or (even worse) falsify it would be nice to find a more handy characterization of USOs.

The following basic but crucial observation provides us with such a characterization: For a USO s on a cube $\mathfrak{C}$ and a label $\lambda \in \operatorname{carr} \mathfrak{C}$, consider the two λ-facets $\mathfrak{C}_1$ and $\mathfrak{C}_2$. Since s is a unique sink orientation, $\mathfrak{C}_1$ and $\mathfrak{C}_2$ have unique sinks o_1 and o_2, respectively. Only these two vertices have the potential to be the global sink, say o_1. Then vertex o_2 has to have its λ-edge outgoing. After flipping all λ-edges o_1 is no longer the global sink and o_2 takes over.

Lemma 4.1 ([42, Lemma 2.1])
Given a unique sink outmap s on a cube $\mathfrak{C}$ and $\Lambda \subseteq \operatorname{carr} \mathfrak{C}$. Then the map

$$\Lambda \oplus s : \mathrm{V}(\mathfrak{C}) \to 2^{\operatorname{carr} \mathfrak{C}}, \quad v \mapsto \Lambda \oplus s(v) \tag{4.1}$$

is a unique sink outmap on $\mathfrak{C}$.

The orientation $\Lambda \oplus s$ differs from s in exactly the Λ-edges. Thus, every edge with label $\lambda \in \Lambda$ is flipped. We will call $\Lambda \oplus s$ the Λ-*reorientation* of s.

Proof. It is enough to prove the statement for $\Lambda = \{\lambda\}$, as for $\Lambda' = \{\lambda_1, \dots, \lambda_k\}$ one gets

$$\Lambda' \oplus s = \{\lambda_1\} \oplus \cdots \oplus \{\lambda_k\} \oplus s.$$

Split $\mathfrak{C}$ into the two λ-facets $\mathfrak{C}_1$ and $\mathfrak{C}_2$. A subcube $\mathfrak{C}'$ is either completely in one of the λ-facets or divided into two faces. For the first case the orientation in $\mathfrak{C}'$ is not affected by relabeling λ (as $\mathfrak{C}'$ does not contain λ-edges), and thus has a unique sink.

In the second case $\mathfrak{C}'_i = \mathfrak{C}' \cap \mathfrak{C}_i$ are facets of $\mathfrak{C}'$. Let o_i be the sink of $\mathfrak{C}'_i$. Without loss of generality o_1 is the sink of $\mathfrak{C}'$ with respect to s. With respect to $\{\lambda\} \oplus s$ the λ-edge in o_1 is outgoing, so o_1 is no longer the sink. But now o_2 is a sink. All other vertices in $\mathfrak{C}'$ have an outgoing edge of label $\neq \lambda$. Thus, $\mathfrak{C}'$ has a unique sink. $\square$

As a consequence of Lemma 4.1 an outmap s of a USO is a bijection: For a USO s consider a vertex $v \in \mathrm{V}(\mathfrak{C})$. It has outmap $\Lambda = s(v)$. By Lemma 4.1 the map $\Lambda \oplus s$ is a unique sink orientation. Its unique sink is v (as $\Lambda \oplus s(v) = \emptyset$). Hence, in the original s no other vertex can have the same outmap $s(v)$, i.e., s is injective and therefore bijective.

Corollary 4.2 ([42, Lemma 2.2])
A unique sink outmap s on a cube $\mathfrak{C}$ is a bijection from $\mathrm{V}(\mathfrak{C})$ to $2^{\operatorname{carr}\mathfrak{C}}$.

Since a USO induces USOs on all subcubes not only is s bijective but all its restrictions to subcubes are bijective as well. In particular, for two vertices u, v their outmap values relative to the smallest subcube containing u and v differ. The smallest subcube containing u and v has carrier $u \oplus v$. The outmap value in u relative to this subcube is $s(u) \cap (u \oplus v)$ and the outmap value in v is $s(v) \cap (u \oplus v)$. Hence, in a USO for each pair of vertices u, v the set $(s(u) \oplus s(v)) \cap (u \oplus v)$ has to be non-empty. This turns out to be a complete characterization for unique sink outmaps.

Proposition 4.3 ([42, Lemma 2.3])
Given a cube $\mathfrak{C}$, a map $s : \mathrm{V}(\mathfrak{C}) \to 2^{\operatorname{carr}\mathfrak{C}}$ is a unique sink outmap on $\mathfrak{C}$ if and only if for all $u, v \in \mathrm{V}(\mathfrak{C})$ we have

$$(u \oplus v) \cap (s(u) \oplus s(v)) \neq \emptyset. \tag{4.2}$$

Proof. First assume s has property (4.2) and fix a subcube $\mathfrak{C}^{[I,J]}$ of $\mathfrak{C}$. We have to show that $\mathfrak{C}^{[I,J]}$ has a unique sink.

Consider the restricted map $\tilde{s} : [I, J] \to 2^{J \setminus I}$,

$$\tilde{s}(u) = s(u) \cap J \setminus I,$$

i.e., the outmap of the orientation on $[I, J]$ induced by s. If $\tilde{s}$ is not injective there are $u, v \in [I, J]$, $u \neq v$, with $\tilde{s}(u) = \tilde{s}(v)$. For these vertices we get

$$(s(u) \oplus s(v)) \cap J \setminus I = \emptyset$$

and since $u \oplus v \subseteq J \setminus I$ this contradicts the assumption (4.2). Ergo $\tilde{s}$ is injective, i.e., bijective. In particular there is exactly one sink $o \in [I, J]$.

Now assume s does not have the property (4.2) and let u, v witness its failure. But then, since $(u \oplus v) \cap (s(u) \oplus s(v)) = \emptyset$, in the subcube $[u \cap v, u \cup v]$ both u and v have the same outmap $\Lambda = s(u) \cap (u \oplus v) = s(v) \cap (u \oplus v)$. Thus, $\Lambda \oplus s$ has two sinks in $[u \cap v, u \cup v]$, i.e., is not a USO. By Lemma 4.1 s is not a USO either. □

Given a cube $\mathfrak{C}$ and $a \in \mathrm{V}(\mathfrak{C})$ consider the map

$$\oplus_a : \mathrm{V}(\mathfrak{C}) \to 2^{\operatorname{carr} \mathfrak{C}}, \quad v \mapsto v \oplus a.$$

By (4.2) $\oplus_a$ is a USO. In a vertex v an edge is outgoing if the Hamming-distance to a is decreasing along this edge. The orientation $\oplus_a$ is called *uniform orientation* towards a. See Figure 4.1 for an example of a uniform orientation.

A uniform orientation is fully described by a and $\operatorname{carr} \mathfrak{C}$. Furthermore, up to isomorphism there is exactly one d-dimensional uniform orientations: The map $\oplus_a$ itself is an isomorphism between the cube $\mathfrak{C}$ with the orientation $\mathfrak{C}_a$ and the cube $2^{\operatorname{carr} \mathfrak{C}}$ with the orientation $\mathrm{id} = \oplus_\emptyset$. Whenever we write *the* uniform orientation we refer to $\oplus_\emptyset$.

Using uniform orientations one can nicely write down isomorphisms between USOs. Let s_1 and s_2 be two USOs on $\mathfrak{C}_1$ and $\mathfrak{C}_2$, respectively. A (digraph-)isomorphism $\Psi : \mathrm{V}(\mathfrak{C}_1) \to \mathrm{V}(\mathfrak{C}_2)$ has to map the sink o_1 of s_1 to the sink o_2 of s_2. Furthermore, the neighbors $o_1 \oplus \{\lambda\}$, $\lambda \in \operatorname{carr} \mathfrak{C}_1$ map to neighbors $o_2 \oplus \{\mu\}$, $\mu \in \operatorname{carr} \mathfrak{C}_2$. Thus, Ψ induces a bijection $\tau : \operatorname{carr} \mathfrak{C}_1 \to \operatorname{carr} \mathfrak{C}_2$ with $\Psi(o_1 \oplus \{\lambda\}) = o_2 \oplus \{\tau(\lambda)\}$. But then for $\Lambda \subseteq \operatorname{carr} \mathfrak{C}_1$ the equation

$$\Psi(o_1 \oplus \Lambda) = o_2 \oplus \tau[\Lambda]$$

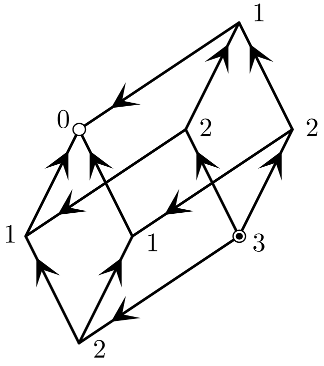


Figure 4.1: A uniform orientation. The vertices are labeled by their Hamming-distance to the sink.

holds. Here $\tau[\Lambda] = \{\tau(\lambda) \mid \lambda \in \Lambda\}$ is the image of Λ under τ.

As Ψ is a digraph-isomorphism the edge $\{o_1 \oplus \Lambda, o_1 \oplus \Lambda \oplus \{\lambda\}\}$ in $\mathfrak{C}_1$ and its corresponding edge $\{o_2 \oplus \tau[\Lambda], o_2 \oplus \tau[\Lambda] \oplus \{\tau(\lambda)\}\}$ are oriented the same way. For the outmaps this implies

$$\tau[s_1(o_1 \oplus \Lambda)] = s_2(o_2 \oplus \tau[\Lambda]).$$

On the other hand, if we find such a τ the corresponding Ψ is a digraph-isomorphism.

Lemma 4.4
Two outmaps s_1 on $\mathfrak{C}_1$ and s_2 on $\mathfrak{C}_2$ describe isomorphic orientations if and only if there exist $a_1 \in \mathrm{V}(\mathfrak{C}_1)$, $a_2 \in \mathrm{V}(\mathfrak{C}_2)$ and a bijection $\tau : \operatorname{carr} \mathfrak{C}_1 \to \operatorname{carr} \mathfrak{C}_2$, such that for all $v \in \mathrm{V}(\mathfrak{C}_1)$

$$\tau[s_1(v \oplus a_1)] = s_2\big(\tau[v] \oplus a_2\big) \tag{4.3}$$

Given two isomorphic USOs s_1 and s_2, we can rewrite the above equation, such that s_2 is expressed in terms of s_1. Set $a = \tau[a_1] \oplus a_2$. Then for all $v \in \mathrm{V}(\mathfrak{C}_2)$ we have $s_2(v) = \tau[s_1(\tau^{-1}[v \oplus a])]$. If we denote the image-map $\Lambda \mapsto \tau[\Lambda]$ by τ' then

$$s_2 = \tau' \circ s_1 \circ (\tau')^{-1} \circ \oplus_a.$$

Given a USO s on $\mathfrak{C}^d$ the set of isomorphic USOs on $\mathfrak{C}^d$ has at least 2^d members. Let o be the sink of s and $a \in \mathrm{V}(\mathfrak{C}^d)$. The map $s \circ \oplus_a$ has its sink in $o \oplus a$. Furthermore, as there are $d!$ possible choices for τ there are at most $2^d d!$ isomorphic USOs. Both upper and lower bound are tight.

4.2 Algebraic Excursion

In the following we will mainly consider USOs on the standard cube $2^{[d]}$, since an arbitrary cube $\mathfrak{C}$ is isomorphic to $2^{[\dim \mathfrak{C}]}$. By Corollary 4.2 a USO s on $\mathfrak{C}^d$ is a permutation of $2^{[d]}$. The inverse s^{-1} is a map from $2^{\operatorname{carr} \mathfrak{C}^d} = \mathrm{V}(\mathfrak{C}^d)$ to $\mathrm{V}(\mathfrak{C}^d) = 2^{\operatorname{carr} \mathfrak{C}^d}$. For $v_1, v_2 \in 2^{[d]}$ and $\hat{v}_i = s(v_i)$ we get

$$(\hat{v}_1 \oplus \hat{v}_2) \cap (s^{-1}(\hat{v}_1) \oplus s^{-1}(\hat{v}_2)) = (s(v_1) \oplus s(v_2)) \cap (v_1 \oplus v_2) \neq \emptyset.$$

Hence, s^{-1} satisfies (4.2) and thus is a USO. This suggests to study the set of all USOs on $\mathfrak{C}^d$

$$\mathrm{USO}(d) = \left\{ s : 2^{[d]} \to 2^{[d]} \;\middle|\; s \text{ is a unique sink orientation} \right\} \tag{4.4}$$

as a subset of the symmetric group $S_{2^{[d]}}$. In particular, one might ask how USOs behave under the group operation $\circ$ on the symmetric group. We will study the following two questions:

1. Which subgroups of $S_{2^{[d]}}$ operate on $\mathrm{USO}(d)$?
2. What is the subgroup generated by $\mathrm{USO}(d)$?

An easy example regarding the first question is given by the subgroup $(\left\{ \oplus_a \;\middle|\; a \in 2^{[d]} \right\}, \circ) \cong (2^{[d]}, \oplus)$. This group operates on $\mathrm{USO}(d)$ from the left as well as from the right: for any $s \in \mathrm{USO}(d)$ the maps $s \circ \oplus_a$ and $\oplus_a \circ s$ are USOs (which is a direct consequence of (4.2)). Both maps we already saw, $s \circ \oplus_a$ is the isomorphic image of s with sink in $o \oplus a$ (where o is the sink of s). And $\oplus_a \circ s = a \oplus s$ is the a-reorientation of s. Figure 4.2 shows how both operations perform on a USO.

Except for this rather simple group (and its subgroups) no other group operates on $\mathrm{USO}(d)$.

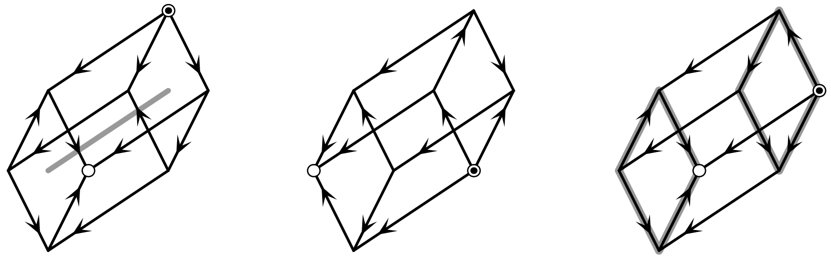


Figure 4.2: A USO s (in the middle), $s \circ \oplus_{\{1,2\}}$ (left) and $\oplus_{\{1,2\}} \circ s$. The left orientation is the mirror image of the middle one along the gray axis. In the right orientation the flipped edges are highlighted.

Proposition 4.5
For a map $g : 2^{[d]} \to 2^{[d]}$ the following statements are equivalent:

(i) $g = \oplus_a$ for some $a \subseteq [d]$,

(ii) for all unique sink orientations s, $s \circ g$ is a unique sink orientation and

(iii) for all unique sink orientations s, $g \circ s$ is a unique sink orientation.

Proof. We just argued that $s \circ \oplus_a$ and $\oplus_a \circ s$ is a USO for all USOs s and all a. This establishes (i)⇒(ii) and (i)⇒(iii).

In the following we proof (ii)⇒(i) and then (iii)⇒(i) using (ii)⇒(i). By Corollary 4.2, g has to be a bijection for (ii) or (iii) to hold. For (ii)⇒(i) we argue that for any $u \in \mathrm{V}(\mathfrak{C})$ the set $g^{-1}(u) \oplus g^{-1}(\emptyset)$ equals u. In fact, the main ingredient is that $u \subseteq g^{-1}(u) \oplus g^{-1}(\emptyset)$.

Assume $s \circ g$ is a USO for all USOs s. Fix $u \subseteq [d]$. For $\lambda \in u$ choose a USO $s = s_{u,\lambda}$, such that $s(u) = \{\lambda\}$ and $s(\emptyset) = \emptyset$. Such an s reveals $\lambda \in g^{-1}(u) \oplus g^{-1}(\emptyset)$:

$$\begin{aligned}
\emptyset &\neq (s \circ g(g^{-1}(u)) \oplus s \circ g(g^{-1}(\emptyset))) \cap (g^{-1}(u) \oplus g^{-1}(\emptyset)) \\
&= (s(u) \oplus s(\emptyset)) \cap (g^{-1}(u) \oplus g^{-1}(\emptyset)) \\
&= \{\lambda\} \cap (g^{-1}(u) \oplus g^{-1}(\emptyset)).
\end{aligned}$$

If we find $s_{u,\lambda}$ for all $\lambda \in u$, we get $u \subseteq g^{-1}(u) \oplus g^{-1}(\emptyset)$. Such $s_{u,\lambda}$ exists, e.g.

$$s_{u,\lambda}(v) = \begin{cases} v & \lambda \notin v \\ v \oplus (u \setminus \{\lambda\}) & \lambda \in v \end{cases},$$

is USO, as one can easily verify by checking the USO-property.

Thus, the mapping $h : u \mapsto g^{-1}(u) \oplus g^{-1}(\emptyset)$ is bijective and for all $u \in \mathrm{V}(\mathfrak{C})$ we have $u \subseteq h(u)$. By induction on $|u|$ we can conclude that u and $g^{-1}(u) \oplus g^{-1}(\emptyset)$ have to be equal. Hence

$$g^{-1} = \oplus_{g^{-1}(\emptyset)}$$

and $g = g^{-1}$. This proves (ii)⇒(i).

For (iii)⇒(i) observe that if $g \circ s$ is a USO then $s^{-1} \circ g^{-1}$ is a USO. In particular, if (iii) holds for a map g, then (ii) holds for g^{-1}. Since we already know that (ii) implies (i), we find an a with $g^{-1} = \oplus_a$. But then $g = g^{-1} = \oplus_a$. □

The answer for the second question is that the smallest group containing USO(d) is the symmetric group. Even worse, already the transpositions in USO(d) generate $S_{2^{[d]}}$.

Proposition 4.6
The group generated by all d-dimensional unique sink outmaps is the symmetric group $S_{2^{[d]}}$.

Proof. As a first step we characterize the transpositions of $S_{2^{[d]}}$ which are also in USO(d). For $a \subseteq [d]$ and $\lambda \in [d]$ consider the uniform orientation towards $\emptyset$ and in this orientation flip the edge between a and $a \oplus \{\lambda\}$. In every vertex $v \subseteq [d]$, $v \neq a, a \oplus \{\lambda\}$ the corresponding outmap is still $s(v) = v$. The outmap value of a in the uniform orientation is a, now since the λ-edge changed, it is $s(a) = a \oplus \{\lambda\}$. Similarly, $s(a \oplus \{\lambda\}) = a$. Therefore, s is the transposition that exchanges a and $a \oplus \{\lambda\}$. We will prove that this is the only type of transposition in USO(d).

Consider a transposition $s = (a, b) \in S_{2^{[d]}}$. Then for all $u, v \in 2^{[d]}$, $u, v \notin \{a, b\}$ (4.2) is satisfied trivially and so is for the pair a, b. Only the pairs a, u and b, u for $u \in 2^{[d]} \setminus \{a, b\}$ are interesting. Choose $\lambda \in a \oplus b$. If $b \neq a \oplus \{\lambda\}$ then

$$(a \oplus a \oplus \{\lambda\}) \cap (s(a) \oplus s(a \oplus \lambda)) = \{\lambda\} \cap (b \oplus a \oplus \{\lambda\}) = \emptyset$$

and s is not a USO. On the other hand, for $b = a \oplus \{\lambda\}$ and $u \neq a, b$

$$(a \oplus u) \cap (s(a) \oplus s(u)) = (a \oplus u) \cap (a \oplus \{\lambda\} \oplus u) \neq \emptyset$$

and symmetrically $(b \oplus u) \cap (s(b) \oplus s(u)) \neq \emptyset$, hence s is a USO. Thus, a transposition is in USO(d) if and only if the two transposed sets have Hamming-distance 1.

To show that USO(d) generates $S_{2^{[d]}}$ it is enough to show how to generate a general transposition (a, b) by transpositions flipping two sets of Hamming-distance 1. The core observation is that for a, b, c we have

$$(a, b)^{(b,c)} = (b, c)(a, b)(b, c) = (a, c).$$

Enumerating $a \oplus b = \{\lambda_1, \ldots, \lambda_k\}$ and setting $a_1 = a \oplus \{\lambda_1\}$, $a_{i+1} = a_i \oplus \{\lambda_{i+1}\}$ we get

$$(a, b) = (a, a_1)^{\prod (a_i, a_{i+1})}.$$

□

4.3 Phases

Given a USO s and a label λ, how can one modify the λ-edges of s? As shown in Lemma 4.1 flipping all λ-edges simultaneously preserves the USO-property. But what if we want to flip only a few edges? For instance, the transposition $(a, a \oplus \{\lambda\})$ differs from the uniform orientation in exactly the edge $\{a, a \oplus \{\lambda\}\}$. Why was this edge flippable?

Let e be a λ-edge incident with the sink o of a USO s. Flipping e makes o a non-sink. To maintain the USO-property a new sink has to be found. The only possible candidate is the sink o' in the λ-facet antipodal to o. Thus, the λ-edge adjacent to o' must be flipped too. As s is a bijection the same argument goes through for all outmap-patterns. Extending this further to subcubes leads to the following definition:

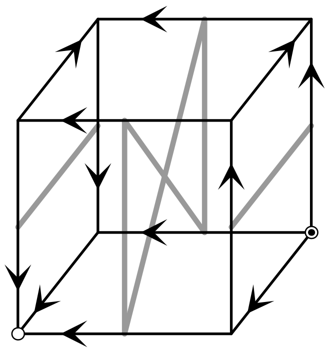


Figure 4.3: A USO and its phases. Edges in phase are connected by a shaded line.

Definition 4.7
Let s be a unique sink orientation of dimension d and $\lambda \in [d]$. Two λ-edges e_1 and e_2 are in direct phase (which we denote by $e_1 \parallel e_2$) if there are $v_1 \in e_1$, $v_2 \in e_2$, such that

$$(v_1 \oplus v_2) \cap (s(v_1) \oplus s(v_2)) = \{\lambda\}. \tag{4.5}$$

Let e_1 and e_2 be two λ-edges in direct phase. In particular, for the witnesses $v_1 \in e_1$ and $v_2 \in e_2$ the label λ has to be in $v_1 \oplus v_2$. Hence, v_1 and v_2 are in different λ-facets. Furthermore,e_1 and e_2 have to be directed towards the same λ-facet: Since $\lambda \in s(v_1) \oplus s(v_2)$, say $\lambda \in s(v_1)$, the edge e_1 is leaving v_1, i.e., pointing towards the λ-facet containing v_2. The edge e_2 is entering v_2, i.e., also pointing towards this facet. See Figure 4.3 for an example.

Consider a USO s with λ-edges e_1 and e_2 in direct phase. Let $v_1 \in e_1$ and $v_2 \in e_2$ witness that they are in phase. If one flips e_1 but not e_2 the resulting orientation s' has a new outmap $s'(v_1) = s(v_1) \oplus \{\lambda\}$ in v_1 whereas in v_2 nothing changed, i.e., $s'(v_2) = s(v_2)$. But then

$$(v_1 \oplus v_2) \cap (s'(v_1) \oplus s'(v_2)) = (v_1 \oplus v_2) \cap (s(v_1) \oplus s(v_2)) \oplus \{\lambda\} = \emptyset$$

and s' is not a USO.

In consequence, to preserve the USO-property one has to flip edges in phase simultaneously. As an edge is directly in phase with itself and being directly in phase is symmetric, the transitive closure is an equivalence relation.

Definition 4.8
Given a unique sink orientation s, let $|||$ be the transitive closure of $||$. The phase of an edge e is the set of all edges in phase to e, $\{e' \mid e \; ||| \; e'\}$. A phase of λ-edges is called a λ-phase.

As $|||$ is an equivalence relation the set of edges is partitioned into phases. Every phase is a λ-phase for some λ since edges in phase are of the same label.

Proposition 4.9
Let L be a set of non-adjacent edges in a unique sink orientation s. The orientation s' one obtains by flipping all edges in L is a unique sink orientation if and only if L is a union of phases.

Proof. The set L is a union of phases if and only if it is closed under $|||$.

First assume L is closed under $|||$. We will verify that s' is a USO by checking (4.2), i.e., for pairs u, v of vertices we show that the set $(u \oplus v) \cap (s'(u) \oplus s'(v))$ is not empty. The sets $s'(u)$ and $s(u)$ differ at most in one element as u is incident to at most one edge in L.

If neither of u and v are incident to edges in L, $s(u) = s'(u)$ and $s(v) = s'(v)$ and there is nothing to prove.

If exactly one of the vertices is incident to an edge in L, say $e = \{u, u \oplus \{\lambda\}\} \in L$, then $s'(u) = s(u) \oplus \{\lambda\}$ and $s'(v) = s(v)$. Furthermore, since the λ-edge in v is not in L, the two edges e and $\{v, v \oplus \{\lambda\}\}$ are not in λ-phase. In particular, $(u \oplus v) \cap (s(u) \oplus s(v)) \neq \{\lambda\}$. Hence, $(u \oplus v) \cap (s'(u) \oplus s'(v))$ is not empty.

If u and v are both incident to λ-edges in L we flip both edges. That is, $s'(u) \oplus s'(v) = (s(u) \oplus \{\lambda\}) \oplus (s(v) \oplus \{\lambda\}) = s(u) \oplus s(v)$ and this case is also good.

Finally, consider two vertices u and v incident to edges in L of different label, say $e_u = \{u, u \oplus \{\lambda\}\}, e_v = \{v, v \oplus \{\mu\}\} \in L$. If $\{\lambda, \mu\} \cap u \oplus v = \emptyset$, the values $(u \oplus v) \cap (s(u) \oplus s(v))$ and $(u \oplus v) \cap (s'(u) \oplus s'(v))$ are equal, hence nonempty.

For $\lambda \in u \oplus v$ and $\mu \notin u \oplus v$ the edge e_u and the λ-edges incident to v cannot be in phase. If they were, the λ-edge incident to v would be in L. But then L would contain two adjacent edges. Thus, the set $(u \oplus v) \cap (s(u) \oplus s(v))$ contains a label different from λ. This label is also in the set $(u \oplus v) \cap (s'(u) \oplus s'(v))$. The case $\lambda \notin u \oplus v$ and $\mu \in u \oplus v$ is resolved symmetrically.

If $\lambda, \mu \in u \oplus v$ consider the vertex w in $[u \cap v, u \cup v]$ with outmap $s(w) \cap (u \oplus v) = (s(u) \oplus \{\lambda\}) \cap (u \oplus v)$. Such a vertex exists since s induces a bijection on $[u \cap v, u \cup v]$. The λ-edge incident to w is in phase with e_u.

Assume $(u \oplus v) \cap (s'(u) \oplus s'(v)) = \emptyset$. This can happen only if the set $(u \oplus v) \cap (s(u) \oplus s(v))$ is equal to $\{\lambda, \mu\}$. But then the μ-edge incident to w is in phase to e_v with respect to s. This cannot be since then both the λ- and the μ-edge incident to w would be in L.

For the back-direction assume L is not closed under $|||$. We will show that then s' is not a USO. In this case one finds edges e and e' being in phase and $e \in L$ but $e' \notin L$. As $|||$ is the transitive closure of $\|$ we find a sequence of edges $e_1, e_2, \ldots, e_k$ with

$$e = e_1 \parallel e_2 \parallel \cdots \parallel e_k = e'.$$

Along this sequence at some point we will leave L, i.e., for some i the edge e_i is in L but e_{i+1} is not. Let $v_i \in e_i$ and $v_{i+1} \in e_{i+1}$ witness $e_i \parallel e_{i+1}$. Then $(v_i \oplus v_{i+1}) \cap (s(v_i) \oplus s(v_{i+1})) = \{\lambda\}$. Since $e_i \in L$ we have $s'(v_i) = s(v_i) \oplus \{\lambda\}$ and since $e_{i+1} \notin L$ the outmap values $s'(v_{i+1})$ and $s(v_{i+1})$ agree. But then $(v_i \oplus v_{i+1}) \cap (s'(v_i) \oplus s'(v_{i+1}))$ is empty, i.e., s' is not USO. $\square$

In the uniform orientation every edge is only in phase to itself. If one chooses two edges incident to the sink then flipping the edges in L gives a 2-face with two sinks. This shows that we cannot drop the condition that L contain only non-adjacent edges.

The simplest way to find a set L without adjacent edges is to take only edges of the same label. In particular, Proposition 4.9 is applicable to the set of all λ-edges. Thus, Lemma 4.1 can be seen as a simple corollary of Proposition 4.9. (We heavily used Proposition 4.3, i.e., Lemma 4.1

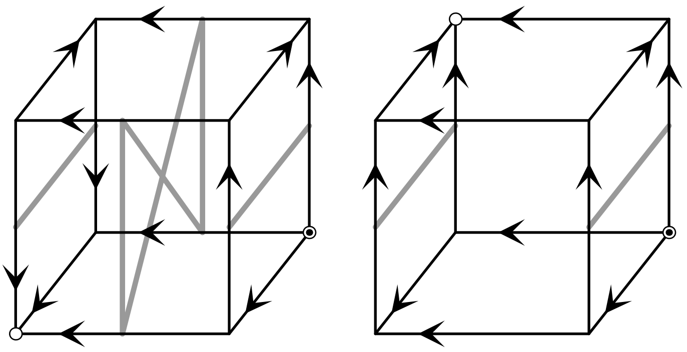


Figure 4.4: Flipping a phase affects other phases. Both USOs differ in one phase.

in the proof of Proposition 4.9.) Flipping a set of λ-edges will not affect the λ-phases. But it highly affects the phases of other labels. See for example Figure 4.4.

Proposition 4.10
Every unique sink orientation can be obtained from the uniform orientation by successively flipping at most d sets of edges.

Proof. We will argue backwards, i.e., starting from a USO s we describe how to flip edges, such that after d flips we get the uniform orientation.

Let $s_1 = s$. Let L_λ be the set of all λ-edges in s_λ pointing towards the upper λ-facet. The orientation $s_{\lambda+1}$ is obtained from s_λ by flipping L_λ. Since edges in phase are directed the same way L_λ is a union of phases and can be flipped.

After at most d flips all edges point towards the corresponding lower facet and therefore s_d is the uniform orientation. □

The construction in Proposition 4.10 gives rise to a random process. For a USO s, first choose a label λ uniformly at random. Now choose a set L uniformly at random from the set of all unions of λ-phases and flip L. Call this USO s_L. As s_L differs from s exactly in the edges in L, the λ-phases of s and s_L agree and $(s_L)_L = s$.

For a d-dimensional USO s let $\mathrm{ph}_\lambda(s)$ be the number of λ-phases. The above random process is described by the following transition-probability:

$$p_{s,s'} = \frac{1}{d} \begin{cases} \sum_{\lambda=1}^{d} 2^{-\mathrm{ph}_\lambda(s)} & \text{for } s = s' \\ 2^{-\mathrm{ph}_\lambda(s)} & \text{for } s' = s_L,\ L \text{ collection of } \lambda\text{-phases} \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

The set L is the unique set of edges in which s and s' differ and therefore $p_{s,s'}$ is well-defined. These probabilities define a *Markov chain.*

We recall the definition of a Markov chain. For a detailed discussion see [5]. Let $(X_t)_{t\in\mathbb{N}}$ be a random process, i.e., a sequence of random variables. For the sake of simplicity we assume that the range of all X_t is finite. We interpret t as time and X_t as a random experiment conducted at time t. Suppose now that the experiments at time $0, \ldots, r-1$ have been conducted with outcomes $X_0 = x_0, \ldots, X_{t-1} = x_{t-1}$. These events may influence the next experiment, i.e., for each possible outcome x_t, the probability

$$\Pr[X_t = x_t \mid X_0 = x_0, X_1 = x_1, \ldots, X_{t-1} = x_{t-1}]$$

may depend heavily on the past results $x_0, \ldots, x_{t-1}$.

The random process $(X_t)_{t\in\mathbb{N}}$ is called a *Markov chain* if the outcome only depends on the last result, i.e.,

$$\Pr[X_t = x_t \mid X_0 = x_0, \ldots, X_{t-1} = x_{t-1} = \Pr[X_t = x_t \mid X_{t-1} = x_{t-1}]$$

for all t and all $x_0, x_1, \ldots, x_t$. A Markov chain is homogeneous if all X_t have the same range of positive values and for all x, y the probabilities $\Pr[X_t = x \mid X_{t-1} = y]$ do not depend on t. For a homogeneous Markov chain the (common) range is called the *state space* and its elements are the *states* of the Markov chain. Furthermore, a homogeneous Markov chain is fully described by its state space, the *transition probabilities*

$$p_{x,y} = \Pr[X_t = y \mid X_{t-1} = x],$$

and $\Pr[X_0 = x]$.

The main fact we are interested in, is that a Markov chain can simulate the uniform distribution. For this we need three more definitions. Consider the digraph G on all possible states of the Markov chain. A state x has a directed edge to y if the transition probability $p_{x,y} > 0$. The Markov chain is called symmetric if $p_{x,y} = p_{y,x}$ for all states x and y. The chain is irreducible, if the digraph G is strongly connected, i.e., for all states x and y we find directed paths from x to y and from y to x. Finally, the chain is aperiodic if we do not find a state x and $k \in \mathbb{N}$, $k > 1$, for which all cycles in G containing x have length congruent to 0 modulo k.

A Markov chain is said to converge to the uniform distribution on the state space S if the k-th powers of the transition matrix $(p_{x,y})_{x,y \in S}$ converges to $(1/|S|)_{x,y \in S}$. In other words, a Markov chain converges to the uniform distributions if the outcome of an experiment is nearly uniform distributed over S as $t \to \infty$. For aperiodic, symmetric and irreducible homogeneous Markov chains the following is known:

Theorem 4.11
Any aperiodic, symmetric and irreducible homogeneous Markov chain converges to the uniform distribution on the state space.

A Markov chain is aperiodic if every $p_{x,x} > 0$. Furthermore, for symmetric Markov chains the digraph G can be simplified to a graph and strongly connected can be replaced by connected.

Applied to the situation of d-dimensional USOs the Markov chain described in (4.6) is a random walk on the graph G_d with

$$\mathrm{V}(G_d) = \mathrm{USO}(d) \quad \text{and} \quad \{s_1, s_2\} \in \mathrm{E}(G_d) \iff p_{s_1,s_2} > 0. \qquad (4.7)$$

Proposition 4.12 ([30])
The homogeneous Markov chain defined by the transition probability in (4.6) is aperiodic, symmetric and irreducible.

Proof. As $p_{s,s} > 0$ the Markov chain is aperiodic.

For symmetry let $s \neq s'$ be two USOs and L the set of edges in which s and s' differ. If L contains edges of different labels then $p_{s,s'} = 0 = p_{s',s}$ since flipping only edges of one label will not destroy all differences in L. For the case that L consists only of λ-edges L has to be a collection of λ-phases, since flipping all edges in L transforms s into s' and s' into s. Furthermore, as flipping one λ-phase does not change the set of all λ-phases the numbers $\mathrm{ph}_\lambda(s)$ and $\mathrm{ph}_\lambda(s')$ agree. Thus, $p_{s,s'} = p_{s',s}$.

By Proposition 4.10, there is a flip-sequence transforming a given USO s to the uniform USO. Each flip in the sequence has positive probability. Thus, there is a path in G_d from s to the uniform USO. Hence, G_d is connected. Since the Markov chain is symmetric this implies that the chain is irreducible. $\square$

In a more concrete form Proposition 4.12 says that we can generate a random USO (drawn uniformly at random from $\mathrm{USO}(d)$) the following way: Choose an arbitrary d-dimensional USO. Then repeatedly choose a random λ together with a random set of λ-phases and flip this set. After some while the resulting orientation will be "sufficiently random". The core problem here is that we do not know how often we need to repeat. So far, we have not been able to resolve this problem.

Nevertheless in the following we start a discussion about the graph G_d. The results we are able to achieve are a (weak) indication that the Markov chain has a reasonable mixing rate. The diameter of G_d is at most $2d$ by Proposition 4.10. Furthermore, we will show that its connectivity is not too small. Obviously, G_d is closely related to the numbers $\mathrm{ph}_\lambda(s)$.

The easiest example for a d-dimensional USO with exactly one d-phase is the following: take some $(d-1)$-dimensional USO s and $\bar{s} = \oplus_{[d-1]} \circ s$ the orientation differing from s in every edge. Then there are only two USOs having s in the lower d-facet and $\bar{s}$ in the upper d-facet, namely the one with all d-edges directed downwards and the one with all d-edges directed upwards. If there were two d-edges with different orientation, there would be a 2-face containing a d-edge pointing upwards and a d-edge pointing downwards. In this 2-face the other two edges would be in phase, but s and $\bar{s}$ differ in them.

Similarly, to have 2^{d-1} d-phases we take some $(d-1)$-dimensional USO and place a copy of it in the lower and upper d-facet of a d-cube. Then the d-edges in this d-cube can be directed arbitrarily. This is the only construction achieving 2^{d-1} d-phases. Call an edge which is in phase with no other edge *flippable*. The next lemma shows that an edge e is flippable if and only if the outmaps of the two incident vertices differ in the label of e only. Consequently, if all d-edges are flippable we must have the same orientation in the upper and lower d-facet.

Lemma 4.13
For a unique sink outmap s an edge $\{v, v \oplus \{\lambda\}\}$ is in phase with no other edge if and only if

$$s(v \oplus \{\lambda\}) = s(v) \oplus \{\lambda\}.$$

PROOF. If $\{v, v \oplus \{\lambda\}\}$ is only in phase with itself, then in particular no vertex other than $v \oplus \{\lambda\}$ has the outmap value $s(v) \oplus \{\lambda\}$. (Otherwise, this vertex would be in phase with v). But since s is a bijection $v \oplus \{\lambda\}$ takes the value $s(v) \oplus \{\lambda\}$.

Now if there is an edge in phase with $\{v, v \oplus \{\lambda\}\}$ there has to be a vertex v' with $(v \oplus v') \cap (s(v) \oplus s(v')) = \{\lambda\}$. Thus

$$(v \oplus \{\lambda\} \oplus v') \cap (s(v) \oplus \{\lambda\} \oplus s(v')) = \emptyset,$$

and s is not a USO. $\square$

Let us now consider the overall number of phases of a USO s, i.e., $\mathrm{ph}(s) = \sum_{\lambda \in [d]} \mathrm{ph}_\lambda(s)$. By the observations above $\mathrm{ph}(s)$ is bounded from below by d and from above by $d2^{d-1}$. The upper bound cannot be improved, since by Lemma 4.13, in the uniform orientation every edge is flippable. In dimension 2 the eye has 4 phases and the bow has 3 phases. In particular, the lower bound of d is not tight even in dimension 2.

Lemma 4.14
For $d > 2$ a d-dimensional unique sink orientation has at least $2d$ phases.

PROOF. Let s be a counterexample of minimal dimension $d > 2$. More precisely, s is a d-dimensional USO with less than $2d$ phases and, if $d > 3$, all USOs of dimension $d - 1$ have at least $2d - 2$ phases.

In particular, we find a label λ for which s has only one λ-phase. For this λ all edges have to be directed towards the same λ-facet (as they are in phase). Now consider two μ-edges e_1 and e_2, $\mu \neq \lambda$, in the different λ-facets. Assume e_1 and e_2 are in phase. The witnessing vertices v_1 and v_2 then agree in the orientation of their incident λ-edge. Hence, one of the λ-edges points towards the upper λ-facet, whereas the other points towards the lower λ-facet. But then there is more than one λ-phase.

In other words, for such s we have one λ-phase and all other phases are phases of one of the λ-facet. For the overall number of phases we therefore can count the number of phases in the lower and in the upper λ-facet independently.

For $d = 3$ both λ-facets have at least 3 phases. Hence, the number of phases in s is at least $3 + 3 + 1 = 7 \geq 2d$, which contradicts the assumption.

For $d > 3$, by assumption the two λ-facets have $2d - 2$ phases each. Altogether this would give $2d - 2 + 2d - 2 + 1 = 4d - 3 \geq 2d$ phases for s, a contradiction. □

The numbers $\mathrm{ph}_\lambda(s)$ are closely related to the degree of s in the graph G_d defined in (4.7). The λ-edges of s can be oriented in $2^{\mathrm{ph}_\lambda(s)}$ ways. One of the orientations is s itself, the others are neighbors in G_d. Hence, the degree of s in G_d is

$$\deg(s) = \sum_{\lambda \in [d]} 2^{\mathrm{ph}_\lambda(s)} - 1.$$

Let $\mathrm{ph}(d) = \min \{\mathrm{ph}(s) \mid s \in \mathrm{USO}(d)\}$. In terms of $\mathrm{ph}(d)$ Lemma 4.14 proves $2d \leq \mathrm{ph}(d)$. We will now relate $\mathrm{ph}(d)$ to the minimal degree $\delta(d)$ of G_d.

Lemma 4.15
$d(2^{\mathrm{ph}(d)/d} - 1) \leq \delta(d) \leq 2^{\mathrm{ph}(d)} - 1.$

Thus, by Lemma 4.14 the minimal degree of G_d has to be at least $d(2^{2d/d} - 1) = 3d$.

Proof. First consider a USO s which is tight in the number of phases. For this s we get

$$\begin{aligned} \sum_{\lambda \in [d]} \mathrm{ph}_\lambda(s) &= \mathrm{ph}(d) \\ \sum_{\lambda \in [d]} 2^{\mathrm{ph}_\lambda(s)} - 1 &\geq \delta(d). \end{aligned}$$

Without knowing how $\mathrm{ph}(d)$ distributes over all labels, we want to bound the latter sum from above. In general, the following holds:

$$\max\left\{\sum_{i\in[d]} 2^{k_i}-1 \;\middle|\; k_i \geq 0, \sum_{i\in[d]} k_i = K\right\} = 2^K - 1.$$

We prove this by induction on d. The statement is clearly true for $d = 1$. Now assume it is true for $d-1$. Then for a fix k_d the sum $\sum_{i\in[d-1]} 2^{k_i}-1$ has maximal value $2^{K-k_d}-1$. Thus, we need to maximize the function $k_d \mapsto 2^{K-k_d} - 1 + 2^{k_d}$ over $0 \leq k_d \leq K$. The maximum is attained at $k_d = 0$. Applied to our situation this proves the upper bound on $\delta(d)$.

For the lower bound on $\delta(d)$ consider a USO s' with minimal degree in G_d. For such s' we know

$$\begin{aligned}\sum_{\lambda\in[d]} 2^{\mathrm{ph}_\lambda(s')}-1 &= \delta(d)\\ \sum_{\lambda\in[d]} \mathrm{ph}_\lambda(s') &\geq \mathrm{ph}(d).\end{aligned}$$

Again we prove a upper bound for the latter sum:

$$\max\left\{\sum_{i\in[d]} k_i \;\middle|\; k_i \geq 0, \sum_{i\in[d]} 2^{k_i}-1 = L\right\} = d\log(L+d) - d\log d.$$

The statement is true for $d = 1$. Now fix k_d and assume the statement is true for $d-1$. Thus, we aim to maximize

$$k_d \mapsto k_d + (d-1)\log(L - 2^{k_d} + 1 + d - 1) - (d-1)\log(d-1).$$

The first derivative of this function is $k_d \mapsto 1-(d-1)2^{k_d}(L-2^{k_d}+d)^{-1}$, which is 0 for $k_d = \log(L+d) - \log d$. In fact, this is the maximum for $k_d \geq 0$ and for this value of k_d we get

$$k_d+(d-1)\log(L-2^{k_d}+1+d-1)-(d-1)\log(d-1) = d\log(L+d)-d\log d.$$

□

The (edge-)connectivity of G_d is trivially bounded by $\delta(d)$ from above. For a lower bound the number $\mathrm{ph}(d)$ is of much more use.

Lemma 4.16
The graph G_d is $2(d+1)$-connected for $d > 2$.

Proof. Since there are 2^d uniform orientations, after deleting $2d+1$ orientations there is still one uniform orientation left. Without loss of generality, it is the orientation towards the empty set. We will show that even if $2d+1$ USOs are forbidden we can find a path from a given USO to the uniform orientation. For the rest of the proof fix a USO s.

We want to successively comb edges until everything is uniform. Define for a label λ the set

$$L_\lambda = \{\{v, v \cup \{\lambda\}\} \mid \lambda \notin v,\, v \to v \cup \{\lambda\}\},$$

which is the set of all λ-edges differing from the uniform orientation. Along a path we want more and more labels to agree with the uniform orientation, i.e., $L_\lambda = \emptyset$ for progressively more labels λ. Call such λ combed (as all edges are "combed downwards"). Let k be the number of combed labels, i.e., without loss of generality $L_1, \ldots, L_k$ are empty.

The goal is to comb one more label (and then proceed inductively). One way to achieve this is to flip a L_λ. The resulting orientation s_λ is combed in $1, \ldots, k$ and λ. (As already argued, L_λ is closed under phases and therefore flippable.)

For $k = d-1$, flipping L_d will end in the uniform orientation (which by assumption has not been deleted). For the rest of the proof let $k < d-1$ and assume that for $k+1$ there is a path to the uniform orientation. In particular, it is enough to show that there is a path from s to a USO with $k+1$ combed labels.

Every s_λ, $\lambda \in \{k+1, \ldots, d\}$ is combed in $k+1$ labels and connected to s by an edge. Thus, if we can choose one of these orientations we are done. For the rest of the proof assume that all these $d-k$ direct paths via the s_λ's are deleted.

We proceed in an indirect way. First we move "horizontally" to a new USO with k combed labels and from there "go up" to a USO with $k+1$ combed labels. If we find more than $d-k+1$ such detours not all of them can be blocked.

Decompose s into 2^k subcubes $0 \cdots 0 * \cdots *$ to $1 \cdots 1 * \cdots *$. As s is combed in the labels $1, \dots, k$ no two edges in different subcubes can be directly in λ-phase, $\lambda = k+1, \dots, d$. (The outmap values of the incident vertices differ at least in one of the labels $1, \dots, k$.) For $k < d-2$, Lemma 4.14 applied to each subcube yields $2(d-k)$ phases per subcube. For $k = d-2$ the subcubes are 2-dimensional and have at least 3 phases. Hence, the $\{k+1, \dots, d\}$-edges split into at least $2^k 2(d-k)$ phases for $k < d-2$ and $3 \cdot 2^{d-2}$ for $k = d-2$.

The first term $2^k 2(d-k)$ is at least $2d$ if and only if $d \geq k \frac{2^k}{2^k - 1}$ or $k = 0$, which is the case for $k < d-2$. The second term $3 \cdot 2^{d-1}$ is at least $2d$ for $d > 2$. In particular, the $\{k+1, \dots, d\}$-edges partition into at least $2d$ phases. For such a phase P define $\hat{P}$: If $P \neq L_\lambda$ set $\hat{P} = P$ otherwise let $\hat{P}$ be the set of all λ-edges. By this choice no two $\hat{P}$ are equal and all $\hat{P}$ can be flipped without ending in an s_λ.

For any such $\hat{P}$ one can first flip $\hat{P}$. Choose afterwards some $\lambda \in \{k+1, \dots, d\}$ different from the edge-labels in $\hat{P}$ and flip L_λ. This results in a USO s_P which has $k+1$ combed edges.

Altogether there are $d-k$ direct routes via s_λ's and at least $2d$ routes via s_P's. Hence, one has to delete $2d + d - k \geq 2d + d - d + 2 = 2d + 2$ vertices from G to kill all these routes. In particular, G is still connected if one deletes only $2d+1$ vertices. □

Virtually the same arguments go through for all bounds $f(d) \leq \mathrm{ph}(d)$, as long as $f(d)$ has the following property:

$$2 \geq \frac{f(d)}{f(d-1)}.$$

In particular, every polynomial lower bound on $\mathrm{ph}(d)$ yields a polynomial bound on the connectivity.

4.4 Local Changes

To be in phase is more a global than a local property. Even edges which are far away potentially can be in the same phase. This makes the concept hard to handle in general. In contrast, it can be decided locally whether a given edge is in phase with any other edges at all by Lemma 4.13. In the following we want to generalize Lemma 4.13.

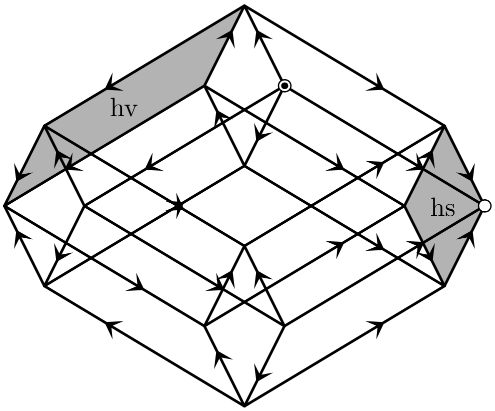


Figure 4.5: A USO with a hypersink (marked with hs) and a hypervertex (marked with hv).

According to Lemma 4.13 an edge $\{o, o \oplus \{\lambda\}\}$ in the sink o is flippable if and only if $s(o \oplus \lambda) = s(o) \oplus \{\lambda\} = \{\lambda\}$. In other words, the 1-dimensional subcube $\{o, o \oplus \{\lambda\}\}$ is flippable if and only if all edges incident to (a vertex of) the subcube are incoming.

Definition 4.17
For a cube $\mathfrak{C}$ and a subcube $\mathfrak{C}_0$ an edge $\{u, v\}$ is called incident *to $\mathfrak{C}_0$ if $u \in \mathrm{V}(\mathfrak{C}_0)$ and $v \notin \mathrm{V}(\mathfrak{C}_0)$. The subcube $\mathfrak{C}_0$ is a* hypersink *in a unique sink orientation s if all incident edges are incoming, i.e., directed towards the vertex in $\mathfrak{C}_0$. It is a* hypervertex *if there is a $\Lambda \subseteq \operatorname{carr} \mathfrak{C}_0$, such that $\mathfrak{C}_0$ is a hypersink in $\Lambda \oplus s$.*

In other words, for a USO s on a cube $\mathfrak{C}$ a subcube $\mathfrak{C}_0$ is a hypervertex if and only if all incident edges of the same label are oriented the same way. For an example see Figure 4.5. With this new notation Lemma 4.13 claims that every 1-dimensional hypervertex can be oriented arbitrarily.

Lemma 4.18 ([39, Lemma 3])
Let s be a unique sink outmap on a cube $\mathfrak{C}$ and $\mathfrak{C}_0$ be a hypersink. For

a unique sink outmap s_0 on $\mathfrak{C}_0$, define $s' : \mathrm{V}(\mathfrak{C}) \to 2^{\operatorname{carr}\mathfrak{C}}$ by

$$s'(v) = \begin{cases} s(v) & \text{for } v \notin \mathrm{V}(\mathfrak{C}_0) \\ s_0(v) & \text{for } v \in \mathrm{V}(\mathfrak{C}_0). \end{cases}$$

Then s' is a unique sink outmap on $\mathfrak{C}$. Furthermore, if s and s_0 are acyclic then s' is acyclic.

Proof. The map s' is well defined since $\operatorname{carr}\mathfrak{C}_0 \subseteq \operatorname{carr}\mathfrak{C}$. Furthermore, since an edge incident to $\mathfrak{C}_0$ is directed towards $\mathfrak{C}_0$ the outmap s' orients every edge only once.

If a subcube $\mathfrak{C}'$ is a subcube of $\mathfrak{C}_0$ then its orientation is determined by s_0. As s_0 is a USO there is a unique sink in $\mathfrak{C}'$. If $\mathfrak{C}'$ is disjoint from $\mathfrak{C}_0$ the orientation is determined by s and again there is a unique sink.

Now assume $V = \mathrm{V}(\mathfrak{C}') \cap \mathrm{V}(\mathfrak{C}_0) \neq \emptyset$ and $\mathfrak{C}'$ is not a subcube of $\mathfrak{C}_0$. Then V spans a subcube $\mathfrak{C}_0'$ in $\mathfrak{C}'$. With respect to s the unique sink of $\mathfrak{C}'$ has to be in $\mathfrak{C}_0'$ as $\mathfrak{C}_0'$ is a hypersink. For s' the vertices outside V do not change their outmap. In particular, if $\mathfrak{C}'$ has a sink with respect to s' it lies in $\mathfrak{C}_0'$. As s' on $\mathfrak{C}_0'$ is given by s_0 and s_0 is a USO there is exactly one unique sink in $\mathfrak{C}_0'$ which is then the unique sink in $\mathfrak{C}'$.

Now let s and s_0 be acyclic. No directed path in $\mathfrak{C}$ can leave $\mathfrak{C}_0$. Therefore, a potential cycle of s' is contained either in $\mathfrak{C}$ and therefore is a cycle w.r.t. s_0 or in $\mathfrak{C} \setminus \mathfrak{C}_0$, in which case it is a cycle w.r.t. s. Consequently, if s_0 and s are acyclic, s' is acyclic as well. $\square$

At first sight it seems that Lemma 4.13 is of more generality than Lemma 4.18 as we could apply it not only to 1-dimensional hypersinks but to all 1-dimensional hypervertices. By the definition of a hypervertex it is rather obvious how to extend Lemma 4.18:

Corollary 4.19 ([39, Corollary 6])
Let s be a unique sink outmap on a cube $\mathfrak{C}$, $\mathfrak{C}_0$ a hypervertex of s and s_0 a unique sink orientation on $\mathfrak{C}_0$. Let Λ be the set of labels of edges leaving $\mathfrak{C}_0$. Then the orientation

$$s'(v) = \begin{cases} s(v) & \text{for } v \notin \mathrm{V}(\mathfrak{C}_0) \\ s_0(v) \cup \Lambda & \text{for } v \in \mathrm{V}(\mathfrak{C}_0) \end{cases}$$

is a unique sink orientation.

Proof. The label set Λ equals $s(v)\setminus\operatorname{carr}\mathfrak{C}_0$ for any v in $\mathfrak{C}_0$. Consider $\hat{s} = \Lambda\oplus s$. In $\hat{s}$ the subcube $\mathfrak{C}_0$ is a hypersink and by Lemma 4.18 can be replaced by s_0. Call the resulting USO $\hat{s}'$. Finally, the USO $s' = \Lambda \oplus \hat{s}'$ has outmap $s'(v) = \Lambda \oplus \Lambda \oplus s(v)$ for $v \notin \mathrm{V}(\mathfrak{C}_0)$ and $s'(v) = \Lambda \oplus s_0(v) = \Lambda \cup s_0(v)$ for $v \in \mathrm{V}(\mathfrak{C})_0$.

By Lemma 4.18 one can replace $\Lambda \oplus s$ in $\mathfrak{C}_0$ by $\Lambda \oplus s_0$. Relabeling the resulting USO with Λ yields s'. □

Unlike in Lemma 4.18, here acyclicity does not necessarily carry over to s'. Only in a hypersource the arguments for acyclicity go through as well.

Furthermore, Corollary 4.19 is best possible in the following sense. Assume that for a USO s, a subcube $\mathfrak{C}_0$ and a label $\lambda \notin \operatorname{carr}\mathfrak{C}_0$, the λ-edges incident to $\mathfrak{C}_0$ are not homogeneously oriented, in particular, we find two neighboring edges $\{u, u \oplus \{\lambda\}\}$ and $\{u \oplus \{\mu\}, u \oplus \{\lambda, \mu\}\}$ with opposite orientation. Then the edges $\{u, u \oplus \{\mu\}\}$ has to be directed towards the same facet as $\{u \oplus \{\lambda\}, u \oplus \{\lambda, \mu\}\}$. In particular, the orientation in $\mathfrak{C}_0$ cannot be chosen arbitrarily.

4.5 Products

So far, we have discussed ways to modify a USO. In the following section we describe how to generate new USOs from smaller ones. Given 2^{d_1} USOs of dimension d_2 the idea is to combine these to a $(d_1 + d_2)$-dimensional USO. In other words, given a d_1-dimensional USO we want to blow up the vertices to d_2-dimensional subcubes.

For cubes such a product is easily defined. For two cubes $\mathfrak{C}_1$ and $\mathfrak{C}_2$ with disjoint label sets the set $\{u \cup v \mid u \in \mathrm{V}(\mathfrak{C}_2), v \in \mathrm{V}(\mathfrak{C}_2)\}$ is the vertex set of a cube with label set $\operatorname{carr}\mathfrak{C}_1 \cup \operatorname{carr}\mathfrak{C}_2$. Call this cube $\mathfrak{C}_1 \times \mathfrak{C}_2$.

Lemma 4.20 ([39, Lemma 5])
Let $\mathfrak{C}_F$ and $\mathfrak{C}_H$ be two cubes with disjoint label sets. Then for a unique sink orientation $\bar{s}$ on $\mathfrak{C}_F$ and unique sink orientations s_v on $\mathfrak{C}_H$, $v \in \mathrm{V}(\mathfrak{C}_F)$, the map s on $\mathfrak{C}_F \times \mathfrak{C}_H$ defined by

$$s(u \cup v) = s_v(u) \cup \bar{s}(v)$$

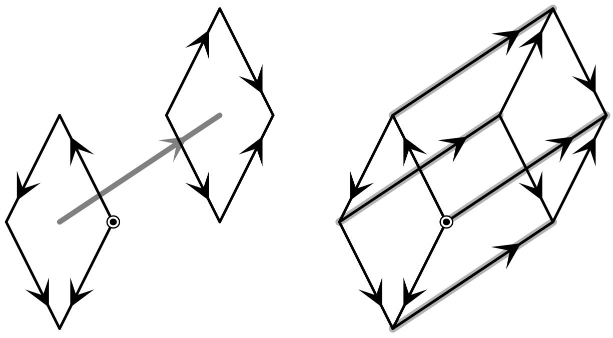


Figure 4.6: A 1-dimensional frame and 2-dimensional hypervertices.

is a unique sink orientation. Furthermore, if $\bar{s}$ and all s_u are acyclic, then so is s.

Figures 4.6, 4.7, and 4.8 illustrate the construction. In particular, Figure 4.6 and Figure 4.7 suggest two different readings of Lemma 4.20. In Figure 4.6$2^{d_1}$ USOs of dimension d_2 are glued together with a d_1-dimensional frame. On the other hand, in Figure 4.7 we replace the vertices of a d_1-dimensional USO by d_2-dimensional hypervertices.

Proof. Any vertex u in $\mathfrak{C}_F \times \mathfrak{C}_H$ is a disjoint union $u = u_H \cup u_F$, $u_H \in \mathfrak{C}_H$ and $u_F \in \mathfrak{C}_F$. Thus, it is enough to show that for $u_H, v_H \in \mathrm{V}(\mathfrak{C}_H)$ and $u_F, v_F \in \mathrm{V}(\mathfrak{C}_F)$ we have

$$\big((u_H \cup u_F) \oplus (v_H \cup v_F)\big) \cap \big((s_{u_F}(u_H) \cup \bar{s}(u_F)) \oplus (s_{v_F}(v_H) \cup \bar{s}(v_F))\big) \neq \emptyset. \tag{4.8}$$

Taking into account that all unions are unions of disjoint sets the left-hand side simplifies to

$$\big((u_H \oplus v_H) \cap (s_{u_F}(u_H) \oplus s_{v_F}(v_H))\big) \cup \big((u_F \oplus v_F) \cap (\bar{s}(u_F) \oplus \bar{s}(v_F))\big).$$

For $w = u_F = v_F$ the set $(u_H \oplus v_H) \cap (s_w(u_H) \oplus s_w(v_H)) \neq \emptyset$ since s_w is a USO. If $u_F \neq v_F$ the second set $(u_F \oplus v_F) \cap (\bar{s}(u_F) \oplus \bar{s}(v_F))$ is non-empty because $\bar{s}$ is a USO. In both cases (4.8) holds.

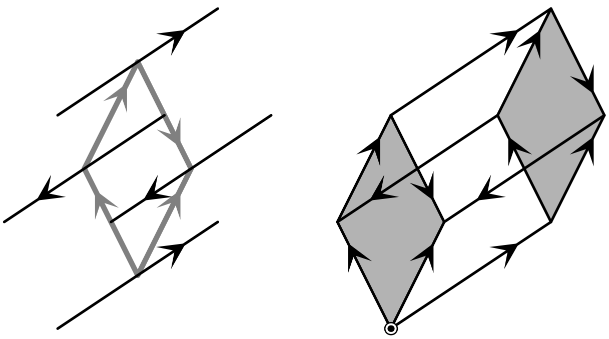


Figure 4.7: A 2-dimensional frame and 1-dimensional hypervertices.

For the second part of the lemma observe that a path $u_1 \cup v_1, \ldots, u_k \cup v_k$ in $\mathfrak{C}_F \times \mathfrak{C}_H$ w.r.t. s induces a walk $v_1, \ldots, v_k$ in $\mathfrak{C}_F$ w.r.t. $\bar{s}$. If the vertices $u_1 \cup v_1, \ldots, u_k \cup v_k$ formed a cycle, i.e. $u_1 \cup v_1 = u_k \cup v_k$, then the induced walk $v_1, \ldots, v_k$ either would contain a cycle in $\bar{s}$ or $v_1 = v_2 = \cdots = v_k = v$. Since $\bar{s}$ is acyclic, we must have the second case. But then $u_1, \ldots, u_k$ forms a cycle in $\mathfrak{C}_H$ w.r.t. s_v. This is a contradiction to s_v being acyclic, i.e., s has to be acyclic. □

For $\dim \mathfrak{C}_F = 1$, Lemma 4.20 says that we can combine two arbitrary $(d-1)$-dimensional USOs to a d-dimensional USO by placing them in two disjoint facets and directing all edges in between in the same direction. An example for this case can be seen in Figure 4.6.

The other extreme case $\dim \mathfrak{C}_H = 1$ shows that if a cube contains two opposite facets with the same $(d-1)$-dimensional USO s, the edges between these facets can be directed arbitrarily. Figure 4.7 illustrates this case. By Lemma 4.13 we already know that such USOs are the only ones in which all edges of one label are flippable. But in addition Lemma 4.20 guarantees them to be acyclic if s was acyclic.

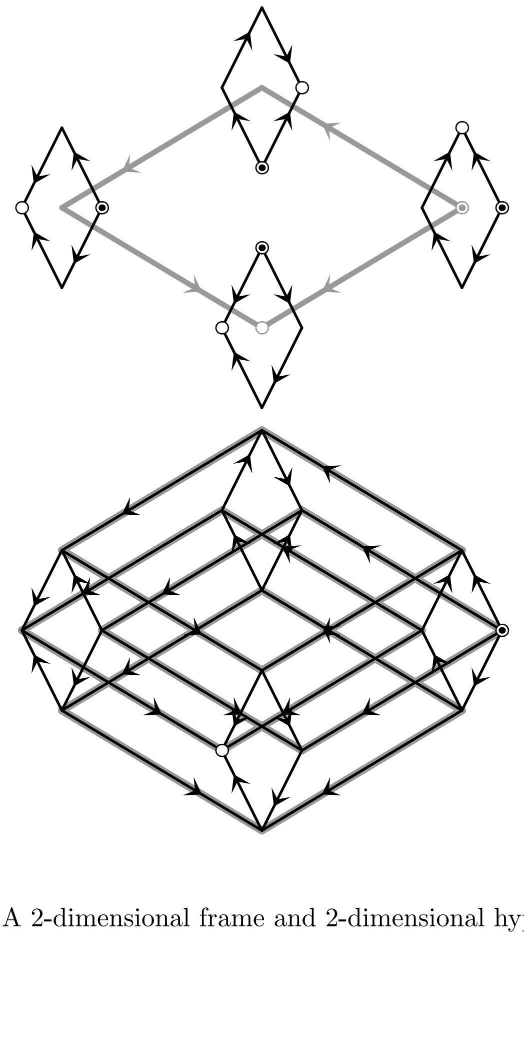


Figure 4.8: A 2-dimensional frame and 2-dimensional hypervertices.

4.6 Examples

In the following we collect USOs that can be obtained by local changes and/or a product construction. Even if the examples are rather simple and straight-forward applications of Corollary 4.19 or Lemma 4.20 they provide us with a good base. In fact, most concrete USOs used in the literature belong to one of the following types.

4.6.1 Partial Unique Sink Orientations

Given a partial orientation on a cube $\mathfrak{C}$, under what circumstances can this orientation be extended to a USO? We address this question in two different ways. In the spirit of this chapter a USO is given by its outmap. Hence, a partial USO would be a partial outmap. We will consider such partial outmaps first. Afterwards we widen our view and consider partial orientations. In difference to partial outmaps such orientations do not have to determine all edges in a vertex.

Let $s_0 : \mathrm{V}(\mathfrak{C}) \supset D \to \operatorname{carr} \mathfrak{C}$ be a partial outmap. If s_0 can be extended to a USO s, this s must satisfy (4.2). In particular, for all $u, v \in D$ we need

$$(u \oplus v) \cap (s_0(u) \oplus s_0(v)) \neq \emptyset.$$

It is not surprising that this condition is not sufficient for the existence of s. Figure 4.9 shows an example of an orientation which is partially good but cannot be extended.

The set D for which s_0 from Figure 4.9 is defined consists of four vertices. Hence, for $|D| \geq 4$ the map s_0 might not be extendible. We will show that this is best possible. For $|D| = 2$ there is always an acyclic USO extending s_0 and for $|D| = 3$ one can extend but might get a cycle. (Since s_0 already can have a cycle, see Figure 4.10).

Lemma 4.21 ([39, Corollary 4])
For two vertices v_1, v_2 in a cube $\mathfrak{C}$ and sets $\Lambda_1, \Lambda_2 \subseteq \operatorname{carr} \mathfrak{C}$ with

$$(v_1 \oplus v_2) \cap (\Lambda_1 \oplus \Lambda_2) \neq \emptyset$$

there exists an acyclic unique sink orientation s with $s(v_i) = \Lambda_i$, $i = 1, 2$.

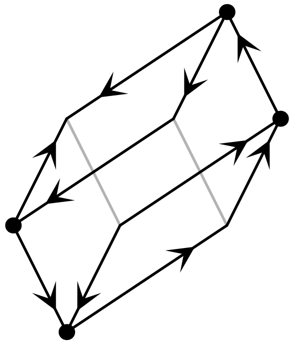


Figure 4.9: A non-extendible partial USO. None of the given vertices (marked with a black dot) contradict (4.2). But the gray edges cannot be oriented without creating a 4-cycle.

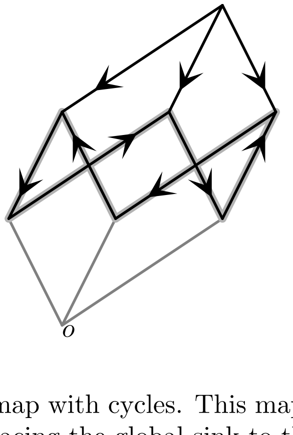


Figure 4.10: A partial map with cycles. This map can be extended to a USO by placing the global sink to the vertex o.

Proof. Choose $\lambda \in (v_1 \oplus v_2) \cap (\Lambda_1 \oplus \Lambda_2)$. Without restriction we can assume $\lambda \in \Lambda_1$. Let $\mathfrak{C}_i$ be the λ-facet containing v_i. On $\mathfrak{C}_i$ let s_i be the uniform orientation towards $v_i \oplus (\Lambda_i \setminus \{\lambda\})$. According to Lemma 4.20 if we orient all λ-edges towards $\mathfrak{C}_1$ the resulting orientation s is a USO.

In s the vertex v_1 has the outmap $s(v_1) = s_1(v_1) = v_1 \oplus v_1 \oplus \Lambda_1 = \Lambda_1$ and v_2 has $s(v_2) = s_2(v_2) \oplus \{\lambda\} = \Lambda_2$. As s_1 and s_2 are acyclic s is acyclic. $\square$

Lemma 4.22
For three vertices v_1, v_2, v_3 in a cube $\mathfrak{C}$ and sets $\Lambda_1, \Lambda_2, \Lambda_3 \subseteq \operatorname{carr} \mathfrak{C}$ with

$$(v_i \oplus v_j) \cap (\Lambda_i \oplus \Lambda_j) \neq \emptyset,$$

for $i, j = 1, 2, 3$, $i \neq j$ there exists a unique sink orientation s with $s(v_i) = \Lambda_i$, $i = 1, 2, 3$.

Proof. Choose $\lambda \in (v_1 \oplus v_3) \cap (\Lambda_1 \oplus \Lambda_3)$. Without loss of generality v_1 and v_2 are in the lower λ-facet $\mathfrak{C}_1$ and v_3 is in the upper λ-facet $\mathfrak{C}_2$. By the previous lemma we find a USO s_1 on $\mathfrak{C}_1$ with $s_1(v_1) = \Lambda_1 \setminus \{\lambda\}$ and $s_1(v_2) = \Lambda_2 \setminus \{\lambda\}$. On $\mathfrak{C}_2$ there is a USO s_2 with $s_2(v_2 \oplus \{\lambda\}) = \Lambda_2 \setminus \{\lambda\}$ and $s_2(v_3) = \Lambda_2 \setminus \{\lambda\}$. By Lemma 4.20 one can furthermore orient all λ-edges according to the λ-edge in v_1 and get a USO s'.

By construction $s'(v_1) = \Lambda_1$ and $s'(v_3) = \Lambda_3$. Furthermore, for v_2 only the λ-edge might be oriented in the wrong direction. But this edge is flippable, since $s'(v_2) \oplus \{\lambda\} = s'(v_2 \oplus \{\lambda\})$. After possibly flipping this edge we found the desired orientation. $\square$

We now consider general partial orientations. That is, we drop the restriction that the orientation is given by a partial outmap. In particular, we are interested in sets of edges, such that *any* partial orientation on this set can be extended to a USO.

Definition 4.23
A set of edges X in a cube $\mathfrak{C}$ is called extendible *if for any orientation ϕ on X there is a unique sink orientation extending ϕ.*

Let $x(d)$ be the maximal cardinality of an extendible edge set in a d-dimensional cube.

For instance, in dimension 2 any edge set of size 3 is extendible. Such set X consists of two edges of the same label, say label 2, and one edge of different label, say label 1. No matter how the two edges of label 2 are oriented, if we choose for the 1-edge not in X the same orientation as for the 1-edge in X, the resulting orientation is USO. Since not every orientation of $\mathfrak{C}^2$ is USO, the set of all edges is not extendible. Hence, $x(2) = 3$.

Lemma 4.24
$x(d) \geq 2^d - 1$.

Proof. The claim is trivially true for dimension 1. Now assume X_d is an extendible edge set in $\mathfrak{C}^d$ with $2^d - 1$ edges. In $\mathfrak{C}^{d+1}$ consider

$$X_{d+1} = X_d \cup \{\{u \oplus \{d\}, v \oplus \{d\}\} \mid \{u, v\} \in X_d\} \cup \{\{\emptyset, \{d\}\}\}.$$

The set X_{d+1} consists of a copy of X_d in the lower d-facet, a copy of X_d in the upper d-facet, and one d-edge.

Any orientation of X_{d+1} can be extended to a USO with the help of Lemma 4.20. Since X_d is extendible we find USOs s_1 and s_2 on the lower and upper d-facet extending the two copies of X_d. Now we connect s_1 and s_2 by combing all d-edges according to the orientation of $\{\emptyset, \{d\}\}$. $\square$

The bound is tight in dimension 3. It is also tight for $d > 3$ if we only allow edge sets for which $(\mathrm{V}(\mathfrak{C}), X)$ is connected.

Lemma 4.25
Let X be an extendible edge set in $\mathfrak{C}^d$, such that $(\mathrm{V}(\mathfrak{C}), X)$ is connected. Then $|X| \leq 2^d - 1$.

Proof. Let X be a set of at least 2^d edges for which $(\mathrm{V}(\mathfrak{C}), X)$ is connected. We will show that such X is not extendible. A spanning tree of $(\mathrm{V}(\mathfrak{C}), X)$ has $2^d - 1$ edges. If we add an additional edge from X to a spanning tree, the resulting subgraph X' contains exactly one cycle C. On C place a cyclic orientation. Edges not in C orient towards C. The resulting orientation ϕ cannot be extended to a USO. For instance, see Figure 4.11

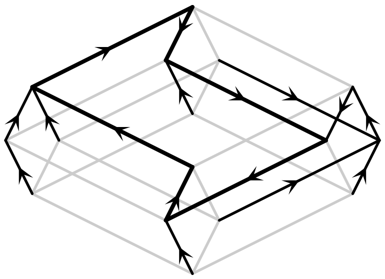


Figure 4.11: An edge set with 2^d edges. The cycle is depicted by bold lines. The orientation drawn in the picture is not extendible, since every vertex has an outgoing edge.

If a vertex $v \in \mathrm{V}(\mathfrak{C})$ is in C it has an outgoing edge in C. If v is not in C, choose some u in C. Since X' is connected, there is a path from v to u. We are only interested in the first edge of such a path. This edge is directed towards C, hence it is outgoing from v. In particular, every vertex in $\mathrm{V}(\mathfrak{C})$ has at least one outgoing edge according to ψ. Hence, no extension of ϕ has a sink. □

We will use Lemma 4.25 as the main ingredient to prove that $x(3) = 7$. Nevertheless, for dimension 3 an exhaustive case distinction over all possible X can still be done, either by computer or by a smart way of enumerating the possible X. For dimension 4 the number of configurations is already too high.

Lemma 4.26
$x(3) = 7$.

Proof. Let X be a set of 8 edges and assume that X is extendible. Since $x(2) = 3$, every facet has at most 3 edges. Let $x_\lambda^{(1)}$ be the number of edges in X and the upper λ-facet. Correspondingly, $x_\lambda^{(0)}$ is the number of edges in X and the lower λ-facet. Double-counting yields

$$\sum_{\lambda=1}^{3} x_\lambda^{(1)} + x_\lambda^{(0)} = 2|X| = 16.$$

This can only be achieved if four of the six facets have 3 edges. In particular, for some λ both λ-facets each have 3 edges in X. Thus, the vertices of the upper λ-facet are in the same connected component of $(\mathrm{V}(\mathfrak{C}), X)$, as well as the vertices of the lower λ-facet. Furthermore, two λ-edges are in X and connect the vertices in the two λ-facets. But then $(\mathrm{V}(\mathfrak{C}), X)$ is connected in contradiction to Lemma 4.25. $\square$

The arguments in the proof fail for dimension 4. The double counting from above yields

$$\sum_{\lambda=1}^{4} x_\lambda^{(1)} + x_\lambda^{(0)} = 32,$$

which can be achieved by $x_\lambda^{(1)} = x_\lambda^{0} = 4$.

4.6.2 Combed and Decomposable Orientations

Definition 4.27
A unique sink orientation s on a cube $\mathfrak{C}$ is λ-combed *if all λ-edges are directed the same way, i.e.*

$$\forall u, v \in \mathrm{V}(\mathfrak{C}) : \lambda \in s(u) \oplus s(v) \iff \lambda \in u \oplus v.$$

An orientation is combed *if it is λ-combed for some $\lambda \in [d]$.*

By flipping all λ-edges pointing upwards (downwards) we can comb any label λ in a USO downwards (upwards). On the other hand, a λ-combed USO is product of its two λ-facets with a 1-dimensional frame. It can be decomposed into two USOs of one dimension lower.

Definition 4.28
A unique sink orientation is decomposable *if every subcube is combed.*

See Figure 4.12 for a decomposable USO and Figure 4.13 for a combed but not decomposable USO.

4.6.3 Klee-Minty Cubes

The most famous representative of the class of decomposable cubes is the so-called Klee-Minty cube. Every one-dimensional USO is a Klee-Minty cube. A d-dimensional cube is a Klee-Minty cube if it is combed

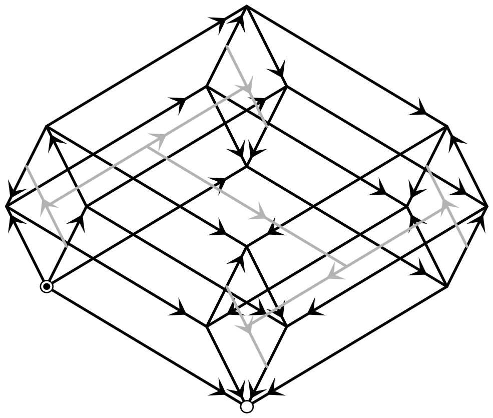


Figure 4.12: A decomposable 4-dimensional cube together with its decomposition.

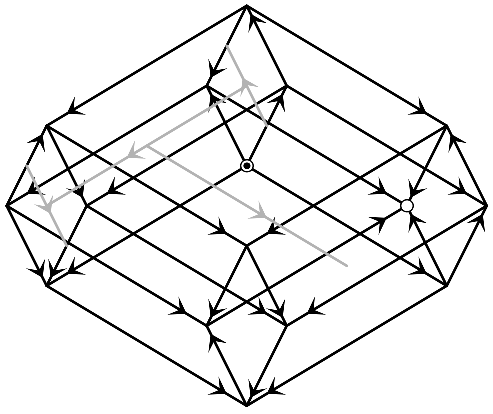


Figure 4.13: A combed 4-cube which is not decomposable.

along a label λ and the two λ-facets are opposite $d-1$-dimensional Klee-Minty cubes. More precisely a USO s on $\mathfrak{C}^d$ is a Klee-Minty cube if we find a $d-1$-dimensional Klee-minty cube s' and a label λ, such that s' defines the orientation in the lower λ-facet, the complete reorientation $\bar{s}'$ of s' defines the orientation in the upper λ-facet and all λ-edges are directed towards the same facet.

Up to isomorphism there is exactly one cube satisfying the definition. A Klee-Minty cube cannot be combed in two labels, since on the opposed facets along a combed label the orientations differ in every edge. We therefore can relabel, such that the Klee-Minty cube of dimension d is d-combed.

Based on this standard labeling one can explicitly write down the outmap km_d of the d-dimensional Klee-Minty cube:

$$\mathrm{km}_d(v) = \{\lambda \mid |\{\lambda, \dots, d\} \cap v| \equiv 1 \pmod 2\}.$$

For two vertices u, v and $\lambda = \max u \oplus v$ the sets $u \cap \{\lambda, \dots, d\}$ and $v \cap \{\lambda, \dots, d\}$ differ in λ only. Thus, exactly one of the two sets is even and $\lambda \in \mathrm{km}_d(u) \oplus \mathrm{km}_d(v)$. This shows that km_d is a USO.

We will show that km_d is a Klee-Minty cube. Trivially, km_1 is a Klee-Minty cube. Now assume km_{d-1} is a Klee-Minty cube. We want to show that km_d is a Klee-Minty cube. For km_d the label d is combed: $d \in \mathrm{km}_d(v)$ if and only if $d \in v$, i.e., all d-edges leave the upper d-facet. Now for a vertex v in the lower d-facet the sets $\{\lambda, \dots, d\} \cap v$ and $\{\lambda, \dots, d-1\} \cap v$ are equal. In particular, $\mathrm{km}_d(v) = \mathrm{km}_{d-1}(v)$. On the other hand, for a vertex u in the upper d-facet the two sets $\{\lambda, \dots, d\} \cap u$ and $\{\lambda, \dots, d-1\} \cap u$ differ exactly in the element d. Thus, the parity changes by one for any λ and $\mathrm{km}_d(u) = [d] \setminus \mathrm{km}_{d-1}(u \setminus \{d\})$.

Since by assumption km_{d-1} is a Klee-Minty cube, km_d is combed in d, has a Klee-Minty cube in the lower d-facet and the opposite orientation in the upper d-facet. Hence, km_d is a Klee-Minty cube.

An alternative way of constructing the d-dimensional Klee-Minty cube is to place the $d-1$-dimensional Klee-Minty cube in both 1-facets (after relabeling $i \to i+1$) and then orient the 1-edges according to the parity of their lower vertex. It is easy to check that this yields km_d again.

The prominence of the Klee-Minty cubes is based on the simplex algorithm. In a vertex v choose an outgoing edge follow it. This describes a simplex algorithm as soon as we give a pivot rule, i.e., a decision rule for which outgoing edge to choose. Such simplex algorithms are normally studied in the set-up of linear programming. Although they are fast in practice, for nearly all deterministic pivot rules examples are known, on which the rule needs exponential many queries to find the optimum. (See [3] for a survey.)

We will discuss only a combinatorial simplex rule. Let Simplex be the simplex algorithm which chooses the edge with minimal label. Formally, Simplex in a vertex v will proceed with $v \oplus \{\min s(v)\}$.

Proposition 4.29
The Simplex *algorithm needs 2^d queries to find the sink of the Klee-Minty cube* km_d *if it starts in its source.*

Proof. The statement is trivially true for dimension 1.

The crucial observation is that the source of the lower d-facet is directly below the sink of the upper d-facet. The global source is the source in the upper d-facet. By induction Simplex needs 2^{d-1} queries until it queries the sink in the upper d-facet. Only then it will choose a d-edge and proceed to the source of the lower d-facet. Again it needs 2^{d-1} queries to finally reach the global sink. Overall this adds up to 2^d queries. □

4.6.4 Matching Flip Orientations

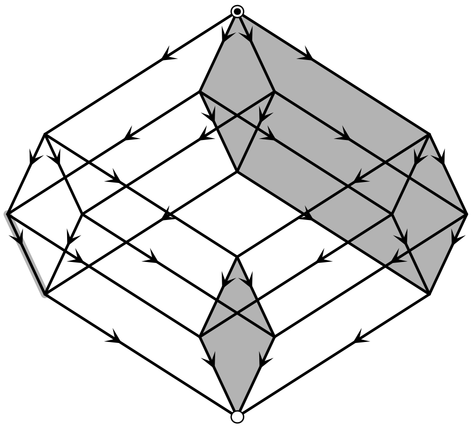
In the uniform orientation on a cube $\mathfrak{C}$ every subcube $\mathfrak{C}'$ is a hypervertex and by Corollary 4.19 can be replaced by an arbitrary USO s. If afterwards one chooses a subcube $\mathfrak{C}''$ disjoint from $\mathfrak{C}'$ $\mathfrak{C}''$ is a hypervertex again and can be replaced. In general, this yields the following:

Lemma 4.30
Let s be the uniform orientation on $\mathfrak{C}$ towards $a \in \mathrm{V}(\mathfrak{C})$. For a family of pairwise disjoint subcubes $(\mathfrak{C}_i)_{i=1,\dots,k}$ of $\mathfrak{C}$ and unique sink orientations s_i on $\mathfrak{C}_i$, $i = 1, \dots, k$ the orientation one obtains by replacing s on $\mathfrak{C}_i$ with s_i is a unique sink orientation.

Proof. As already mentioned by Corollary 4.19 we only have to show that every $\mathfrak{C}_i$ is a hypervertex. But as the $\mathfrak{C}_i$ are mutually disjoint the edges incident to a $\mathfrak{C}_i$ are determined by s. As s is combed in all labels $\mathfrak{C}_i$ is a hypervertex. $\Box$

A family of disjoint cubes of dimension 1, i.e., a set of disjoint edges is a matching on $\mathfrak{C}$. In particular, every matching on $\mathfrak{C}$ gives a USO on $\mathfrak{C}$ by flipping the matching edges in the uniform orientation. Furthermore, all these orientations are mutually different. Such an orientation is called *matching flip orientation* for obvious reasons. As there are $d^{\Omega(2^d)}$ perfect matchings on a d-dimensional cube this shows:

Proposition 4.31 ([27])
The number of unique sink orientations in dimension d is bounded from below by $d^{\Omega(2^d)}$.



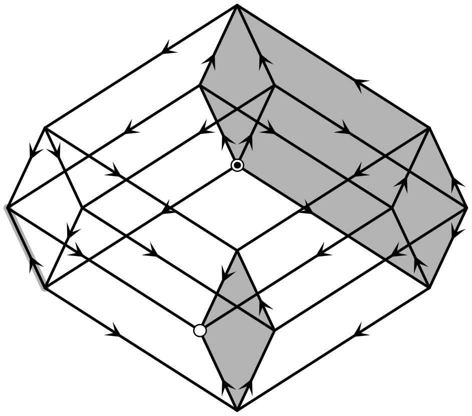


Figure 4.14: The upper orientation is uniform. In the lower orientation the gray subcubes are flipped.

4.7 Remarks

The lower bound in Proposition 4.31 is sharp in the sense that the number of USOs in dimension d is $2^{\theta(2^d \log d)}$ (cf. page 16 and [27]). Thus, the simple proof of Proposition 4.31 is rather surprising. We will see in the next chapter that matching flip orientations are very special.

In connection with isomorphisms the flippable edges of a USO play an important role. Obviously, one has to check all flippable edges in order to know if two USOs are isomorphic. In particular, whether a USO is uniform cannot be checked without knowing all edges. From this point of view the number of flippable edges is an interesting invariant for checking isomorphy. Nevertheless, in high dimension nearly no two USOs are isomorphic. The size of an isomorphism class is at most $d!2^d$ whereas the overall number of USOs $2^{\theta(2^d \log d)}$.

The question arises if there are USOs without flippable edges. The answer to this question is closely related to a conjecture by Keller [22]: A tiling of $\mathbb{R}^d$ by unit cubes is a collection of (geometric) unit cubes, such that each point x in $\mathbb{R}^d$ is in one of these cubes and if x is in more than one cube it is on the boundary of all cubes containing x. Keller conjectured that in any such tiling we find two cubes, such that their intersection is a facet of both of them. Szabó [41] showed that Keller's conjecture can be decided on a much smaller class of tilings. The class introduced by Szabó is in one-to-one correspondence to d-dimensional USOs and a tiling falsifies Keller's conjecture if and only if the corresponding USO has no flippable edge. For $d \leq 6$ Keller's conjecture is true [36], but it gets false starting in dimension 8. (For a counterexample in dimension 8 see [26], for dimensions 10 and 12 see [25].) In particular, each USO of dimension less or equal 6 has a flippable edge, whereas starting in dimension 8 there are USOs with no flippable edge. The question remains open for dimension 7.

A flippable edge in a hypervertex is globally flippable. In particular, neither Corollary 4.19 nor Lemma 4.20 can construct a USO without flippable edges from USOs with flippable edges. Thus, we cannot construct all d-dimensional USOs from lower-dimensional USOs using these two construction schemes only.

The phase-concept is general enough to generate all USOs. Unfortunately, it is too general to be used for concrete constructions. In the following we will study algorithms for finding the sink. To achieve lower bounds one often needs to extend a partial outmap to a USO. As new vertices can produce new phase-dependencies such constructions are hard to obtain using phases. In contrast local changes and products turn out to be very useful for this purpose.

The Markov chain was first suggested by Valtr and Matoušek [30]. Unfortunately, its mixing rate is unknown. Still, it is the only known method to produce a random USO with approximate uniform distribution.

5 Algorithms

Sink him!

(Stubb)

5.1 Complexity Model

The main interest in USOs is to solve Sink or SinkOrFalsify: By accessing the outmap the goal is to find the sink (i.e., query it) or to give a certificate that the underlying orientation is not a USO. The characterization of unique sink outmaps (4.2) suggests certificates of the form u, v with $(u \oplus v) \cap (s(u) \oplus s(v)) = \emptyset$. Nevertheless we still allow for (much) longer certificates. Consider an algorithm $\mathcal{A}$ for which it is known that $\mathcal{A}$ needs at most $t_{\mathcal{A}}(d)$ queries on a d-dimensional USO to find the sink. Then, for an outmap s, a sequence of queries longer than $t_{\mathcal{A}}(d)$ is a certificate for s not being USO.

Since in the complexity model for USOs we only count the number of vertex evaluations, the problem does not directly relate to classical complexity theory. As seen in Chapter 3, we use USOs to solve other problems via a well-behaved outmap. In the following we reformulate this connection in the setting of formal languages.

Definition 5.1
For a language $L \subseteq \Sigma^* \times \Sigma^*$ over the alphabet Σ, the problem $F(L)$ is to find for any $x \in \Sigma^*$ a $y \in \Sigma^*$ such that $(x, y) \in L$, or to decide that no such y exists.

The class of languages for which $(x, y) \in L$ can be decided in polynomial time is called FNP. *The subclass of $L \in$ FNP for which $F(L)$ can be solved in polynomial time is called* FP.

The names FNP and FP are not chosen arbitrarily. In fact, every problem in NP has a variant in FNP. For instance, Sat is equivalent to

$$\{(x, y) \mid y \text{ satisfying assignment for formula } x\}.$$

On the other hand, for $F \in$ FNP the language

$$\pi_1 L := \{x \mid \exists y : (x, y) \in L\}$$

is in NP. Thus, P = NP iff FP = FNP. The notions of polynomial reducibility, completeness and hardness carry over to the functional variants. For a more detailed description, see [34, Chapter 10].

In the problems studied in Chapter 3 we always had a parameter which fixes the dimension for the corresponding USO. In the setting of general languages it is more convenient to describe USOs in a way which does not depend on the dimension.

For the rest of the chapter, we assume without loss of generality $\Sigma = \{0, 1, *\}$. In particular, we can describe cubes and subcubes as words in Σ^*. Let

$$2^{<\infty} = \{v \subseteq \mathbb{N} \mid v \text{ finite}\}.$$

As before, we embed $2^{<\infty}$ into $\{0,1\}^*$ except now we do not fix the dimension. More formally, a set $u \in 2^{<\infty}$ is represented by the word w of length $\max u$ with $w_\lambda = 1 \iff \lambda \in u$. In particular, w ends with 1.

A map $s : 2^{<\infty} \to 2^{<\infty}$ is a unique sink outmap iff for all $u, v \in 2^{<\infty}$ condition (4.2) holds. We say that such an outmap s has dimension d if it is uniform outside $2^{[d]}$, i.e., $s(v) = v$ for all $v \in 2^{<\infty} \setminus 2^{[d]}$. In particular, for a d-dimensional USO the cube $\mathfrak{C}^d$ is a hypersink of s in $2^{<\infty}$. This enhanced definition captures the original definition of USOs: By Corollary 4.19 we can embed any USO into the uniform orientation on $2^{<\infty}$.

Definition 5.2
For a functional language $L \subseteq \Sigma^* \times \Sigma^*$, a unique sink oracle is a tuple $(\mathcal{S}, \mathcal{T})$ of functions

$$\begin{aligned} \mathcal{S} &: \Sigma^* \to (2^{<\infty} \to 2^{<\infty}) \\ \mathcal{T} &: \Sigma^* \times 2^{<\infty} \to \Sigma^* \end{aligned}$$

(called oracle $\mathcal{S}$ and interpreter $\mathcal{T}$), such that for any $x \in \pi_1 L$, the map $\mathcal{S}(x)$ is a unique sink orientation, and for the sink o of this orientation, the interpretation $y = \mathcal{T}(x, o)$ satisfies $(x, y) \in L$.

A unique sink oracle is called polynomial if $\mathcal{S}(x)(v)$ and $\mathcal{T}(x, v)$ can be calculated in polynomial time in $|x|$ and $|v|$, and the dimension of $\mathcal{S}(x)$ is polynomial in $|x|$.

For instance, let L be the language, such that the first argument encodes a feasible, bounded linear program $\mathrm{LP}(A, b, c)$, and the second argument encodes a solution to $\mathrm{LP}(A, b, c)$. Then Definition 3.22 defines an oracle $\mathcal{S}$ by Theorem 3.23. Furthermore, Proposition 3.25 yields an interpreter $\mathcal{T}$. In particular, $(\mathcal{S}, \mathcal{T})$ is a polynomial unique sink oracle for linear programming.

Oracle and interpreter are allowed to do anything on $x \notin \pi_1 L$. In most cases, oracles are only meaningful on $\pi_1 L$. It might even be that an algorithm for $\mathcal{S}$ or $\mathcal{T}$ does not terminate outside $\pi_1 L$.

Assume for a moment that there is an algorithm for SinkOrFalsify which needs polynomial time to find the sink. Furthermore, assume that there is a polynomial unique sink oracle for FSat. Given some $x \in \pi_1$FSat, by assumption, the sink o of the corresponding USO can be found in polynomial time. The interpretation y of o solves FSat x. Hence, such an oracle for FSat implies that SinkOrFalsify is not polynomial provided P $\neq$ NP.

Theorem 5.3
Given an NP*-complete problem L and its functional variant FL. If there exists a polynomial unique sink oracle for FL, then* NP = coNP.

Proof. We will construct a polynomial time non-deterministic algorithm which decides $x \notin L$. This proves $L \in$ coNP. As a consequence NP = coNP.

Let $(\mathcal{S}, \mathcal{T})$ be the polynomial oracle for FL. Consider the language

$$\begin{aligned} L_{\mathcal{S}} \;=\; & \{(x, o) \mid \mathcal{S}(x)(o) = \emptyset\} \cup \\ & \{(x, u\#v) \mid (u \oplus v) \cap (\mathcal{S}(x)(u) \oplus \mathcal{S}(x)(v)) = \emptyset\}. \end{aligned}$$

With the conventions above, $L_{\mathcal{S}}$ is a language over Σ. Furthermore, for any $x \in \Sigma^*$ the outmap $\mathcal{S}(x)$ is either a USO (and thus we find x, o with $\mathcal{S}(x)(o) = \emptyset$) or has two vertices failing (4.2). Thus, the projection $\pi_1 L_{\mathcal{S}}$ equals Σ^*. In other words, $L_{\mathcal{S}}$ is total.

The relation $(x, y) \in L_{\mathcal{S}}$ can be checked in polynomial time. The dimension of $\mathcal{S}$ is polynomially bounded in $|x|$, say by $p(|x|)$. After scanning the first $2p(|x|) + 1$ symbols of y, we can distinguish three cases:

(i) y is too long,

(ii) y is of the form $y = o$, $o \in 2^{[p(|x|)]}$, or

(iii) y is of the form $y = u\#v$, $u, v \in 2^{[p(|x|)]}$.

In the first case, (x, y) cannot be in $L_{\mathcal{S}}$. In the second case, we have to check if $\mathcal{S}(x)(o)$ is empty. And in the third case, we have to check $\mathcal{S}(x)(u) \oplus \mathcal{S}(x)(v) = \emptyset$. Both can be done in polynomial time.

Let $x \notin L$. Consider the following algorithm: For $y \in \Sigma^*$, first check if # appears more than once in y. If so, y is rejected. If # appears exactly once in y, say $y = u\#v$, we can check in polynomial time whether $\mathcal{S}(x)(u) \oplus \mathcal{S}(x)(v) = \emptyset$. If not, reject y, otherwise y is a certificate for $\mathcal{S}(x)$ not being a USO. Thus, y certifies $x \notin L$. If $\# \notin y$ and $\mathcal{S}(x)(y) \neq \emptyset$, reject y. Otherwise, calculate $\hat{y} = \mathcal{T}(x, y)$. If $(x, \hat{y}) \in FL$ then $x \in L$, otherwise $x \notin L$. All steps can be done in polynomial time by assumption.

Thus, the procedure just described decides in polynomial time if y is a certificate for $x \notin L$. This establishes $L \in \text{coNP}$. □

5.2 A Lower Bound on Deterministic Algorithms

The setup for this section is the following: We are given an arbitrary deterministic algorithm $\mathcal{A}$ for Sink and want to construct a USO on which $\mathcal{A}$ needs "many" queries. Rather than exploiting structural aspects of such algorithms we do this in a game theoretic fashion. We play a game against $\mathcal{A}$. In every move the algorithm queries a vertex and we reveal the orientation in this vertex. We have to answer according to the outmap of a USO s. This outmap will be constructed in an on-line fashion depending on the queries of the algorithm. At the end of the game we have a concrete (acyclic) USO s for which $\mathcal{A}$ needs nearly a quadratic number of queries.

Given a deterministic algorithm $\mathcal{A}$ for finding the sink, we construct an acyclic unique sink orientation of dimension d for which the algorithm needs an almost-quadratic number of queries. For the first $d - \lceil \log_2 d \rceil$ inquiries, we maintain a partial outmap $s : W \to \operatorname{carr} \mathfrak{C}^d$ on the set $W \subseteq \mathrm{V}(\mathfrak{C}^d)$ of queried vertices, containing the answers we gave so far. We also maintain a set Λ of labels and an acyclic unique sink outmap $\tilde{s}$ on $\mathfrak{C}^\Lambda$. 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 $\Lambda = W = \emptyset$. After each inquiry we answer by revealing the value of the outmap s at the requested vertex.

Then we update Λ and $\tilde{s}$, such that the following conditions hold.

(a) $|\Lambda| \leq |W|$,

(b) $w' \cap \Lambda \neq w'' \cap \Lambda$ for every $w' \neq w'' \in W$ and

(c) $s(w) = \tilde{s}(w \cap \Lambda) \cup [d] \setminus \Lambda$ for every $w \in W$.

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

Suppose now that the algorithm 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 $u \cap \Lambda = w \cap \Lambda$ we have to distinguish two cases.

Case 1. For every $w \in W$, we have $w \cap \Lambda \neq u \cap \Lambda$.

Then we answer $s(u) = \tilde{s}(u \cap \Lambda) \cup [d] \setminus \Lambda$ and leave Λ 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 $u \cap \Lambda = v \cap \Lambda$.

By condition (b), there is exactly one such $v \in W$. The assumption of Case 2 and $u \neq v$ imply that we can fix a label $\lambda \notin \Lambda$ such that $\lambda \in u \oplus v$.

We answer $s(u) = s(v) \setminus \{\lambda\}$. As $\lambda \notin \Lambda$, the label λ is in $s(v)$.

Now we have to update Λ and $\tilde{s}$. We get our new Λ by adding λ. To define the new orientation $\tilde{s}$, we take two copies of the old $\tilde{s}$ on the two facets determined by λ. Then, by Lemma 4.20, we can define the orientation of the edges going across arbitrarily, so we make them such that the condition (c) is satisfied. More formally, let

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

Next, we have to check conditions (a)–(c) for our new W, Λ and $\tilde{s}$. Condition (a) still holds, because we added one element to each of W and Λ. Since Λ 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 λ was chosen from $u \oplus v$ and now is included in Λ, $u \cap \Lambda \neq v \cap \Lambda$. Condition (c) is straightforward from the definitions.

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

Our plan is to apply a local change with $\mathfrak{C}_0$ and an orientation s, which is consistent with the outmaps of the vertices evaluated so far. Corollary 4.19 then will enable us to reveal s on $\mathrm{V}(\mathfrak{C}^d) \setminus \mathrm{V}(\mathfrak{C}_0)$ to the algorithm and still be able to start a completely “new game” on $\mathfrak{C}_0$, a cube of relatively large dimension. To construct s satisfying the conditions of Lemma 4.18, we use the product construction of Lemma 4.20 twice.

First we define an orientation $\bar{s}$ of $\mathfrak{C}^{\Lambda'}$ using the product construction for $\tilde{s}$ on the frame $\mathfrak{C}^{\Lambda}$ and outmaps s_v on the hypervertices along $\mathfrak{C}^{\Lambda' \setminus \Lambda}$ with the property that for every $w \in W$, the map $s_{w \cap \Lambda}$ has its source at $w \cap (\Lambda' \setminus \Lambda)$. This last requirement can be satisfied because of condition (b). For $v \notin W$ the map s_v can be arbitrary.

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

Next, we use again the product construction with $\bar{s}$ on the frame $\mathfrak{C}^{\Lambda'}$, so we have to construct USOs s_v of $\mathfrak{C}^{[d] \setminus \Lambda'}$ for every $v \in \mathrm{V}(\mathfrak{C}^{\Lambda'})$. In doing so, we only take care that the sinks of all these orientations are in $\mathfrak{C}_0$, and if $v = w \cap \Lambda'$ for some $w \in W$, then $w \setminus \Lambda'$ is the source of s_v. (By condition (b), there can be at most one such vertex w for each v.) The appropriate s_v exists according to Lemma 4.21. Now apply the product construction with frame $\bar{s}$ and the hypervertices s_v. This provides us with an orientation s which agrees with our answers given for the evaluated vertices. Furthermore, $\mathfrak{C}_0$ is a hypersink w.r.t. s.

We can reveal s on $\mathrm{V}(\mathfrak{C}^d) \setminus \mathrm{V}(\mathfrak{C}_0)$ and still be able to place *any* orientation on $\mathfrak{C}_0$, a hypersink of dimension $d - \lceil \log_2 d \rceil$. Therefore, we just proved

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

Theorem 5.4 ([39, Theorem 9])
Any deterministic algorithm needs $\Omega(\frac{d^2}{\log d})$ many vertex evaluations to find the sink of an acyclic unique sink orientation on a d-dimensional cube.

Proof. We prove by induction for $d \geq 2$

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

Since $t_{acyc}(2) \geq 1 = \frac{4}{2} - \frac{2}{2}$ and $t_{acyc}(3) \geq 1 \geq \frac{9}{4} - \frac{3}{2}$, the inequality holds for $d = 2, 3$.

Now let $d \geq 4$. By induction for $2 \leq k < d$ we have $t_{acyc}(k) \geq \frac{k^2}{2\lceil \log_2 k \rceil} - \frac{k}{2}$. Since $d - \lceil \log_2 d \rceil \geq 2$ we get:

$$\begin{array}{rcl} t_{acyc}(d) & \geq & d - \lceil \log_2 d \rceil + t_{acyc}(d - \lceil \log_2 d \rceil) \\ & \geq & d - \lceil \log_2 d \rceil + \\ & & + \dfrac{(d - \lceil \log_2 d \rceil)^2}{2\lceil \log_2 (d - \lceil \log_2 d \rceil) \rceil} - \dfrac{1}{2}(d - \lceil \log_2 d \rceil) \\ & \geq & \dfrac{1}{2}d - \dfrac{1}{2}\lceil \log_2 d \rceil + \\ & & + \dfrac{d^2 - 2d\lceil \log_2 d \rceil + \lceil \log_2 d \rceil^2}{2\lceil \log_2 d \rceil} \\ & = & \dfrac{d^2}{2\lceil \log_2 d \rceil} - \dfrac{d}{2}, \end{array}$$

and the inequality also holds for d. □

5.3 Small Dimensions

For dimension 0, $t(0) = 1$, since there is no edge. This is a good opportunity to point out once more that we require the sink to be queried, even if (as in this case) we know where it is. For dimension 1, obviously $t(1) = 2$.

After querying the vertices of even cardinality in a USO s, the position of the sink is known. In fact, then we know the whole orientation. In particular, this shows that $t(2) \leq 2 + 1 = 3$ and $t(3) \leq 4 + 1 = 5$. In the following, we not only show that this simple algorithm is best possible. In both dimensions, this bound is sharp even on the set of acyclic orientations. Furthermore, we can forbid a vertex in the beginning.

Proposition 5.5 ([39, Proposition 10])
Every deterministic algorithm needs three queries to find the sink of a two-dimensional unique sink orientation, even if there is a fixed vertex which is known not to be the sink. In particular, $t(2) = 3$.

Proof. The strategy answers with the source for the first query of an algorithm. Now, all remaining three unqueried vertices are still potential sinks by Lemma 4.21. Even if one of them is known not to be the sink, there are two possibilities left. Therefore, no deterministic algorithm can evaluate the sink in two steps. □

As in the general lower bound, the main tool in the following is provided by Lemma 4.18: We will construct a partial outmap with a hypersink. Since the sink has to be located in this hypersink, we can then proceed recursively, using Lemma 4.18.

Proposition 5.6 ([39, Proposition 11])
Every deterministic algorithm needs five queries to find the sink of a three-dimensional acyclic unique sink orientation, even if there is a fixed vertex which is known not to be the sink. In particular, $t(3) = 5$.

Proof. Let $\mathcal{A}$ be some algorithm for Sink and u be the vertex which is known not to be the sink. We construct an example s on $\mathfrak{C}^3$ for which $\mathcal{A}$ needs 5 steps at least.

The first query v_1 will be answered with the source, $s(v_1) = [3]$. If the second query v_2 is not antipodal to v_1, both vertices are in a common facet, say $\lambda \notin v_1 \oplus v_2$. In this case set $s(v_2) = \{\lambda\}$. Combing all λ-edges towards the λ-facet $\mathfrak{C}_0$ not containing v_1 and v_2 makes $\mathfrak{C}_0$ a

2-dimensional hypersink. Any algorithm needs 3 more queries to find the sink in $\mathfrak{C}_0$, even if the non-sink u is in $\mathfrak{C}_0$.

If the second query v_2 is antipodal to v_1, choose some $\lambda \in v_2 \oplus u$ if $u \neq v_2$ or $\lambda = 1$ otherwise. Without loss of generality, $v_1 = 000$, $v_2 = 111$ and $\lambda = 1$. For all four edges $*10$, $1{*}0$, $10*$ and $*01$, there is an acyclic USO having the edge as hypersink (cf. Figure 5.1). At most one of the edges contains u (as $u = v_2$ or $u \in 0{*}{*}$) and at most two edges are blocked by the third query. Therefore, after 3 queries there is a 1-dimensional hypervertex for which any algorithm needs 2 additional queries to find the sink. $\square$

One can prove even stronger statements. For instance, it is true that even the knowledge about two fixed points of distance either three or one that neither is the sink would not help an algorithm to find the sink: it would still need to evaluate 5 vertices. Curiously, if it is known about two vertices of *distance two* that neither is the sink, an algorithm finding the sink in 4 steps exists.

Proposition 5.7 ([39, Proposition 12])
Every deterministic algorithm needs at least seven steps to find the sink of a four-dimensional acyclic unique sink orientation.

Proof. The first query v_1 of an algorithm $\mathcal{A}$ will be answered by the source, $s(v_1) = [4]$. For the second query v_2, choose a $\lambda \in v_1 \oplus v_2$ and answer $s(v_2) = [4] \setminus \{\lambda\}$. Without loss of generality $\lambda = 1$ and $v_1 = 0000$.

If v_1 and v_2 are not antipodal, we comb label 4. Thus, the facet $\mathfrak{C}'$ along label 4 not containing v_1 and v_2 is a hypersink. Since any algorithm needs 5 steps to find the sink in $\mathfrak{C}'$ we need 7 steps altogether.

For the rest of the proof assume v_1 and v_2 are antipodal, i.e., $v_1 = 0000$ and $v_2 = 1111$. Our goal for the next two queries is to keep a 2-dimensional hypersink unqueried. Since then we can force $\mathcal{A}$ into 3 more queries. In the following we fix two USOs s_1 and s_2. Our answers up to the fourth query rely either on s_1 or on s_2.

We now construct s_1. Let $\bar{s}$ be the bow on $\mathfrak{C}^{00**}$ with sink in 0010 and source in 0011. Construct a USO s_0 with frame $\bar{s}$ using Lemma 4.20. The hypervertices are $s_{**00} = s_{**01} = s_{**10} = \oplus_{11}$ and s_{**11} is the bow with source in 0011 and sink in 1011 (cf. Figure 5.2).

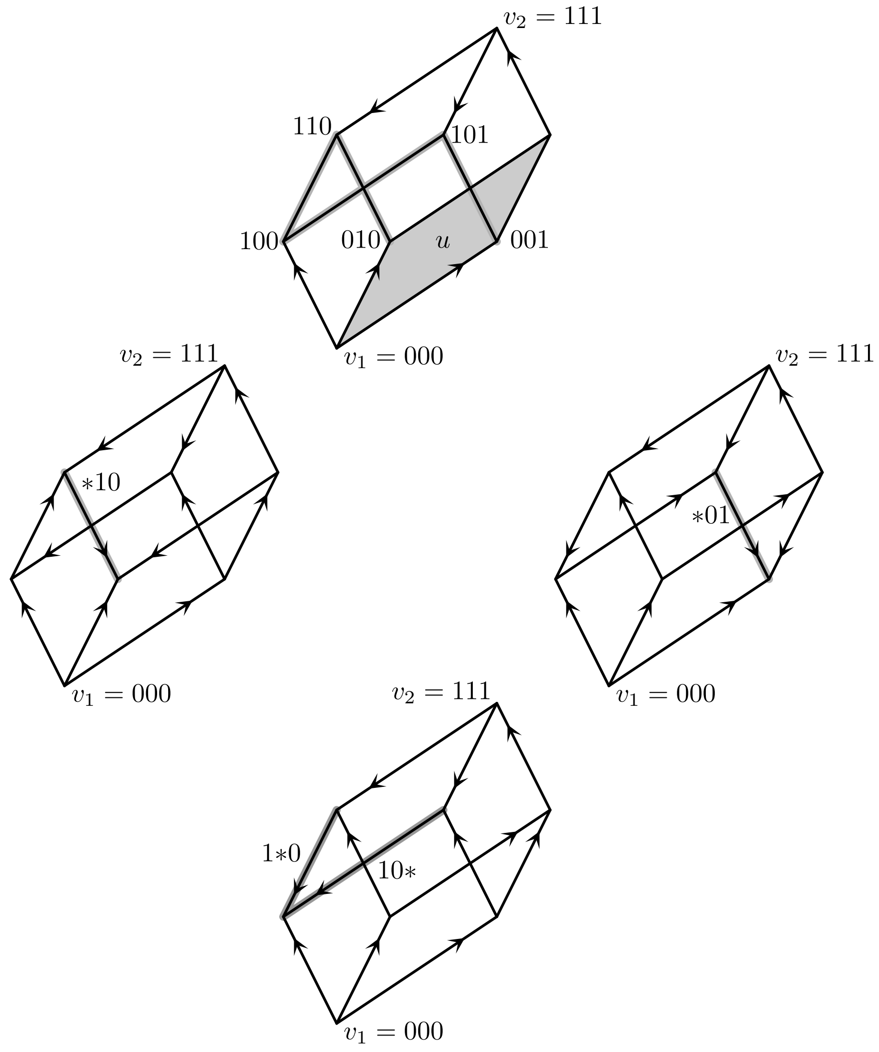


Figure 5.1: The orientations forcing 5 steps in dimension 3.

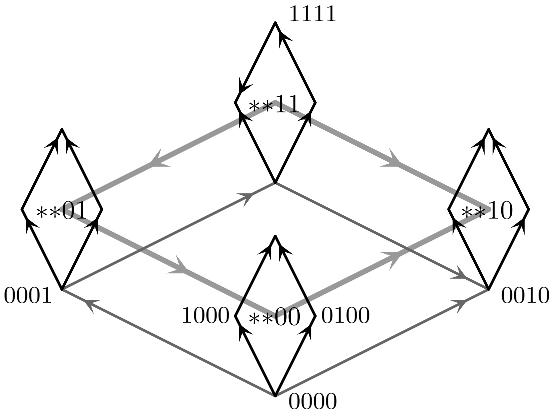


Figure 5.2: The construction of s_1.

In s_1 the subcube $00**$ is a hypersource. We can replace this hypersource with the bow having its sink in 0010 and source in 0000. The resulting USO s_1 is acyclic. See Figure 5.3 for a picture of s_1. For s_2 flip the two edges $01{*}0$ and $11{*}0$ in s_1. In fact, both edges are flippable. Hence, s_2 is a USO. See Figure 5.4 for a picture of s_2.

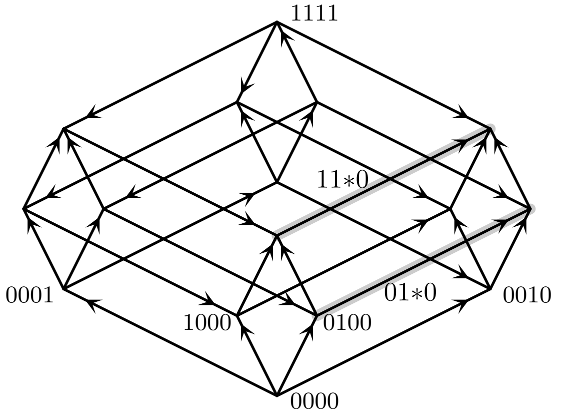
We only need some facts about s_1 and s_2. First of all, both orientations agree on all vertices outside the faces $**10$ and $*10*$. Furthermore, for v_1 and v_2 the outmap-values coincide with the answers we gave. And finally, for s_1 the face $**10$ is a hypersink, whereas for s_2 the face $*10*$ is a hypersink.

We are now ready to deal with the remaining queries of $\mathcal{A}$. For the third query v_3 there are six cases with respect to $v_1 \oplus v_3$. So far we only fixed label 1. All other labels can still be permuted without affecting $s(v_1)$ or $s(v_2)$. According to the six cases we now decide on the other three labels.

(1) $v_1 \oplus v_3 = \{1\}$. Then v_3 is the vetex 1000.

(2) $v_1 \oplus v_3 = \{\mu\}$, $\mu \neq 1$. Then set $\mu = 4$, i.e., $v_3 = 0001$.

(3) $v_1 \oplus v_3 = \{1, \mu\}$, $\mu \neq 1$. Then set $\mu = 4$, i.e., $v_3 = 1001$.



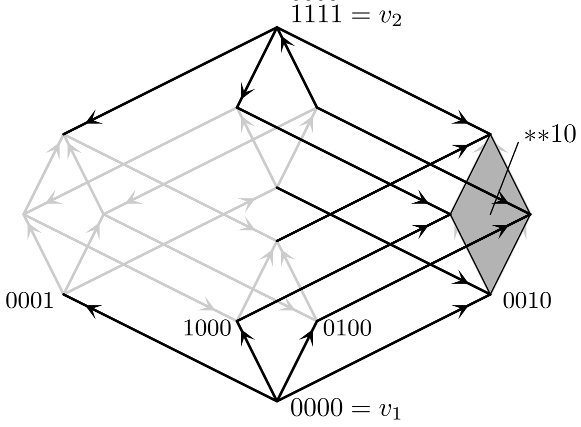
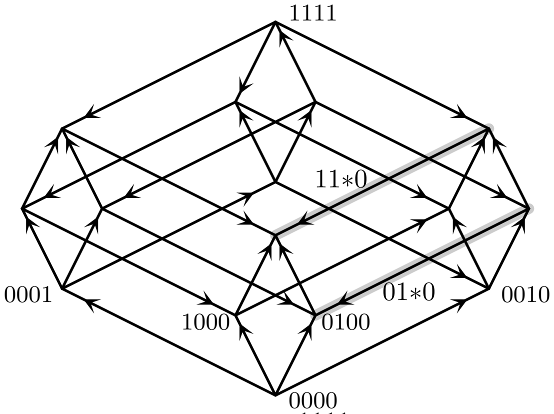


Figure 5.3: The USO s_1. The upper picture shows the whole USO, whereas the lower picture depicts only the relevant facts.



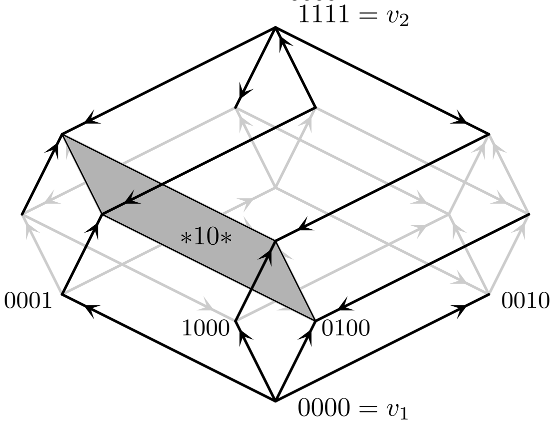


Figure 5.4: The USO s_2. The upper picture shows the whole USO, whereas the lower picture depicts only the relevant facts.

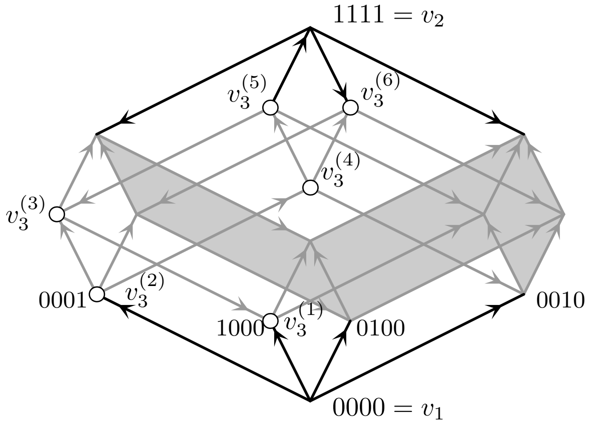


Figure 5.5: The possible positions of v_3. The gray shaded area covers the subcubes where we still want to change the orientation according to the remaining queries.

(4) $v_1 \oplus v_3 = \{\mu, \nu\}$, $\mu, \nu \neq 1$. Then set $\mu = 4$ and $\nu = 3$, i.e., $v_3 = 0011$.

(5) $v_1 \oplus v_3 = [4] \setminus \{\mu\}$, $\mu \neq 1$. Then set $\mu = 2$, i.e., $v_3 = 1011$.

(6) $v_1 \oplus v_3 = \{2, 3, 4\}$. Then v_3 is the vertex 0111.

See Figure 5.5 for the possible positions of v_3. In no case the two 2-faces $*10*$ and $**10$ contain a queried point. Answer to v_3 according to $s_1(v_3) = s_2(v_3)$.

If the fourth query v_4 is not in $**10$ we fix $s(v) = s_1(v)$ outside $**10$. Since $**10$ is hypersink in s_1, we can replace the orientation in this face arbitrarily and force the algorithm into 3 more queries to find the sink in $**10$. This makes 7 queries altogether.

If on the other hand v_4 is in $**10$, we fix $s(v) = s_2(v)$ outside $*10*$ and proceed as above. □

This proof provides us with a very small class of 4-dimensional USOs on which every deterministic algorithm needs 7 queries at least. Furthermore, the constructed USOs are rather simple. One would therefore

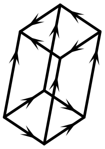


Figure 5.6: A possible choice for s_0 in the proof of Proposition 5.8.

suspect that 7 is not the best lower bound and more sophisticated examples would give better bounds. However, the SEVENSTEPSTOHEAVEN algorithm introduced by Szabó and Welzl [42] finds the sink of a 4-dimensional USO in 7 steps.

For dimension five the correct value is still not known. The best upper bound is 12 (see [4]). For the lower bound, there is a simple argument for 9 which will be given below. Again, the underlying idea is that after a small number of queries (in this case, four), there still remains a large untouched subcube.

Proposition 5.8
Every deterministic algorithm needs at least nine steps to find the sink of a five-dimensional acyclic unique sink orientation.

PROOF. As usual, we answer to the first query v_1 with the source, i.e. $s(v_1) = 11111$. For the second query v_2, we choose $\lambda \in v_1 \oplus v_2$ and set $s(v_2) = 11111 \oplus \{\lambda\}$. If v_1 and v_2 are not antipodal, the algorithm needs $2 + t(4) = 9$ queries. For the rest of the proof, assume $v_1 = 00000$, $v_2 = 11111$ and $\lambda = 3$.

Our goal is to answer to the first four queries in such a way, that after the fourth query we can fix a USO with a 3-dimensional hypersink, such that no vertex in the hypersink is queried. In fact, all four queries will be in two antipodal 3-dimensional cubes.

Fix an acyclic USO s_0 on $\mathfrak{C}^{***00}$ with outmap $s_0(00000) = 11100$ and $s_0(11100) = 11000$. That is, s_0 is compatible with the answers to v_1 and v_2 relative to $\mathfrak{C}^{***00}$. See, for instance, Figure 5.6

The third query v_3 has distance 2 to either v_1 or v_2, say v_i. If $3 \in v_i \oplus v_3$ we can relable the cube, such that v_3 is in the subcube spanned by v_i and the labels $2, 3$. Otherwise, we can assume that v_3 is in the subcube spanned by v_i and $1, 2$. See Figure 5.7.

We answer to v_3 either $s(v_3) = s_0(v_3 \cap \{1, 2, 3\}) \cup \{4, 5\}$ if $i = 1$ or $s(v_3) = s_0(v_3 \cap \{1, 2, 3\}) \cup \{5\}$ if $i = 2$.

If $v_i \oplus v_3 \subseteq \{2, 3\}$, we can always achieve that v_4 is in one of the subcubes $\mathfrak{C}^{***00}$ or $\mathfrak{C}^{***11}$. See Figure 5.8.

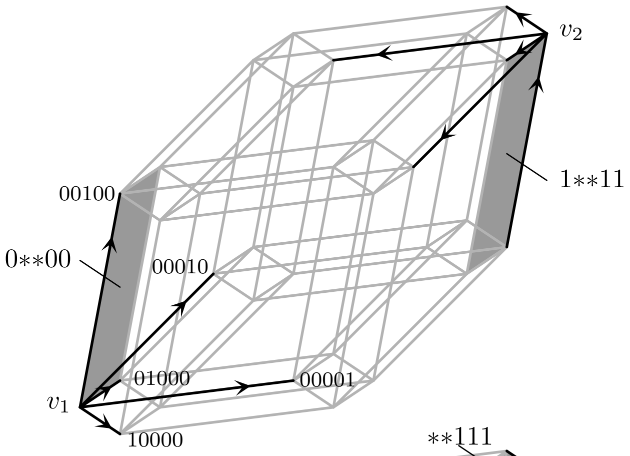
For $v_i \oplus v_3 \subseteq \{1, 2\}$ after relabeling the fourth query v_4 is either in one of the subcubes $\mathfrak{C}^{**00*}$ and $\mathfrak{C}^{**11*}$ or in one of the subcubes $\mathfrak{C}^{***00}$ or $\mathfrak{C}^{***11}$. See Figure 5.9.

If all four queries $v_1, \ldots, v_4$ are in $\mathfrak{C}^{**00*}$ and $\mathfrak{C}^{**11*}$, we construct the final answer s by using Lemma 4.20 with a 2-dimensional frame and 3-dimensional hypervertices. For the frame orientation $\bar{s}$ on $\mathfrak{C}^{00**0}$ choose the bow with source in 00000 and sink in 00100. For the hypervertices $\mathfrak{C}^{**00*}$ and $\mathfrak{C}^{**11*}$ choose orientations compatible with the answers to v_1, v_2, and v_3. (Such USOs exist by Lemma 4.21.) Answer to v_4 according to this construction. For the remaining hypervertices $\mathfrak{C}^{**01*}$ and $\mathfrak{C}^{**10*}$ we can choose arbitrary acyclic USOs. In particular, we can force 5 more queries in $\mathfrak{C}^{**01*}$. See Figure 5.10.

If on the other hand all four queries $v_1, \ldots, v_4$ are in $\mathfrak{C}^{***00}$ and $\mathfrak{C}^{***11}$, we use Lemma 4.20, now with a 3-dimensional frame and 2-dimensional hypervertices. The frame orientation is $\bar{s} = s_0$. For the hypervertices we always choose the bow with source in $***00$ and sink in $***01$, except for the hypervertex in 11100. There we choose the bow with source in $***11$ and sink in $***01$. See Figure 5.11. The resulting orientation is compatible with the answers so far. Answer to v_4 according to this construction. Furthermore, $\mathfrak{C}^{***01}$ is a hypersink. Thus, we can force five more queries.

$\square$

The idea of the four proofs above looks promising at first sight: After the first few queries, there are still untouched large subcubes. Unfortunately, in general, for fixed k the number of vertices one has to query to pierce every $(d-k)$-dimensional subcube is only logarithmic in d (see [2]).



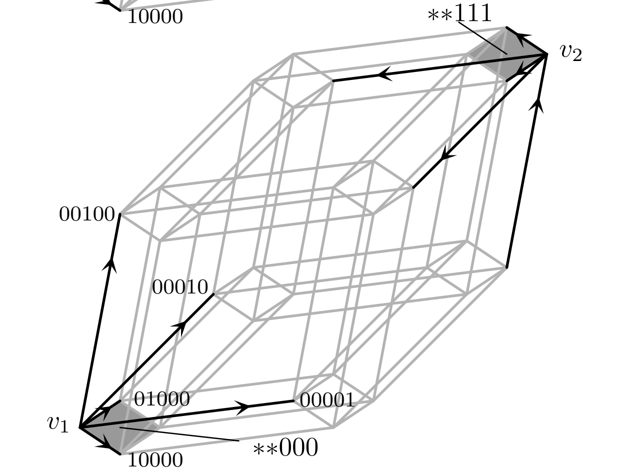


Figure 5.7: The possible positions of v_3. The gray shaded area covers the subcubes where v_3 can be found.

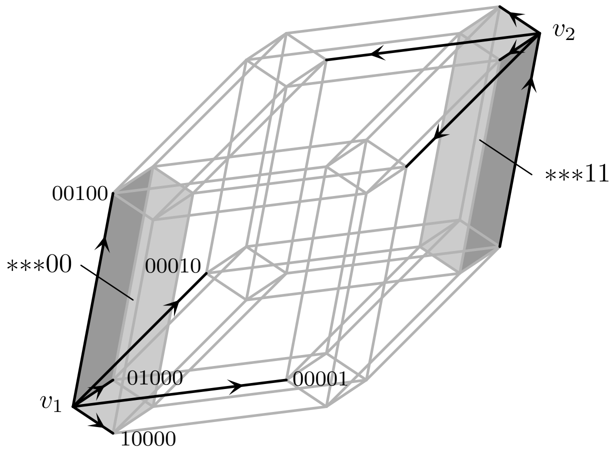


Figure 5.8: The possible positions of v_4 provided $v_i \oplus v_3 \subseteq 01100$. The dark gray shaded area covers the subcubes where v_3 can be found, whereas v_4 can be found in the light grey.

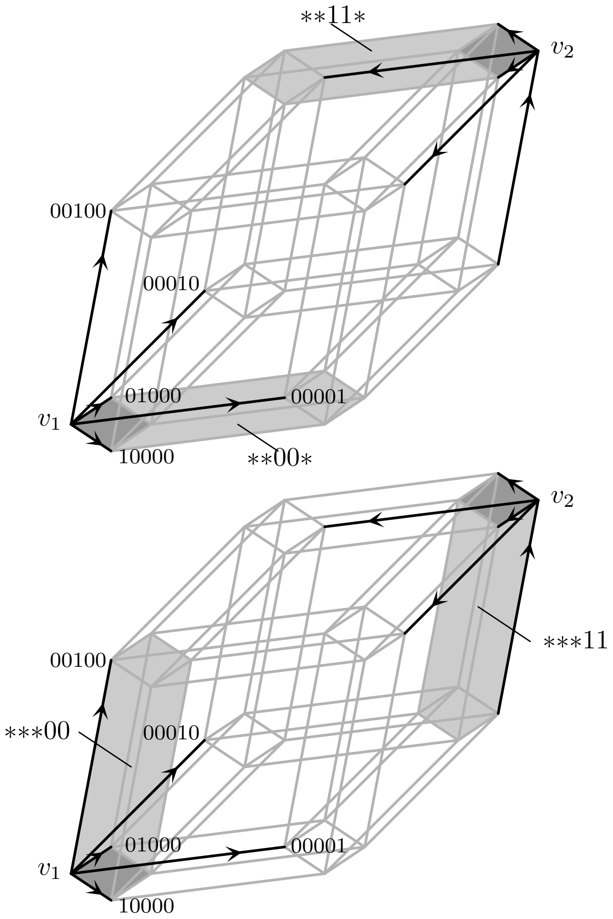


Figure 5.9: The possible positions of v_4 provided $v_i \oplus v_3 \subseteq 11000$. The dark gray shaded area covers the subcubes where v_3 can be found, whereas v_4 can be found in the light grey.

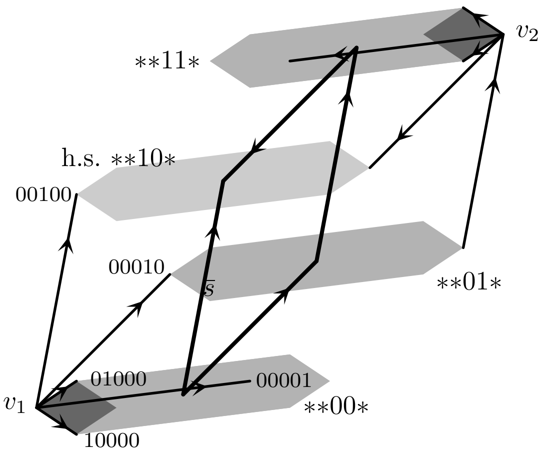


Figure 5.10: The construction for $v_1, \ldots, v_4 \in \mathfrak{C}^{**00*} \cup \mathfrak{C}^{**11*}$.

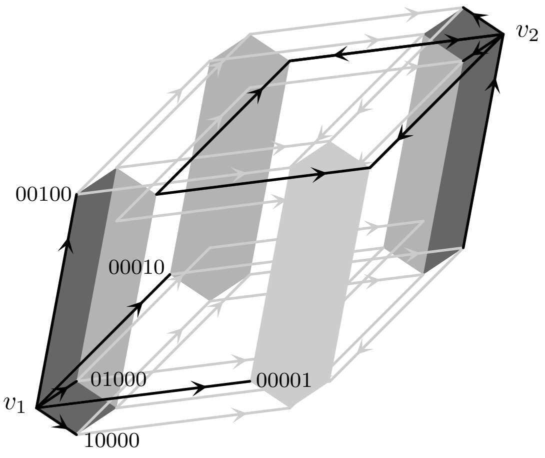


Figure 5.11: The construction for $v_1, \ldots, v_4 \in \mathfrak{C}^{***00} \cup \mathfrak{C}^{***11}$.

5.4 Fast Subclasses

A matching flip orientation s, as defined in Section 4.6.4, is given by a vertex $\hat{o}$ and a matching M: s is obtained by flipping the edges in M in the uniform orientation $\oplus_{\hat{o}}$. In $\oplus_{\hat{o}}$ every vertex v fulfills the equation $\oplus_{\hat{o}}(v) \oplus v = \hat{o}$. In other words, if one knows that a USO s is uniform then with two queries one finds the sink: First query an arbitrary vertex v_1 and then $v_2 = s(v_1) \oplus v_1$.

If s is a matching flip orientation given by $\hat{o}$ and M, then $s(v) \oplus v$ must no longer be $\hat{o}$. But M has at most one edge adjacent to v and therefore $s(v) \oplus v$ has at most distance one to $\hat{o}$. Furthermore, the sink o of s is either a neighbor of $\hat{o}$ or $\hat{o}$ itself: If $o \neq \hat{o}$, there has to be an edge adjacent to $\hat{o}$ in M. But then o is the other vertex incident with this edge.

This observation allows us to find the sink with 5 queries. After the first query we are already very close to the sink. Exploring the structure near $\hat{o}$ we need 4 more queries in the direct neighborhood of $\hat{o}$ *independent of the dimension.*

Proposition 5.9 ([39, Proposition 8])
In a matching flip orientation s, the sink can be found in at most 5 steps (independent of the dimension of s). The value 5 here is best possible, except in dimension ≤ 2.

Proof. Let s be obtained by flipping the edges of a matching M in $\oplus_{\hat{o}}$. After querying an arbitrary vertex v_1, the vertex $v_2 = s(v_1) \oplus v_1$ is a neighbor of $\hat{o}$. Thus, v_2 has at most two outgoing edges.

Obviously, v_2 is our next query. 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 $\hat{o}$, v_3 has to be the sink. Otherwise, $v_3 = \hat{o}$ 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 $\hat{o}$ 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 $\hat{o}$ and we need one more query to find the sink. This makes five queries altogether.

On the other hand, after answering the first query v_1 the source, its antipodal vertex $\bar{v}_1$, and all neighbors $v \in N(\bar{v}_1)$ of $\bar{v}_1$ are potentially the sink. Until one of these vertices is evaluated we answer to all queries according to $\oplus_{\bar{v}_1}$. (Since all edges of one label could be flipped, our answers do not reveal any information on $\bar{v}_1$ or $N(\bar{v}_1)$.) To find the sink among the vertices in $N(\bar{v}_1) \cup \{\bar{v}_1\}$, four queries are needed. This proves the optimality. □

In the above algorithm, the first step made the most improvement, as v_1 could have been anywhere and $v_2 = s(v_1) \oplus v_1$ is close to the sink. In general, $v \oplus s(v)$ will not improve with respect to the distance to the sink. (For instance, if source and sink are neighbors and v is the source.) But if s is combed then $v \oplus s(v)$ will be in the facet which is a hypersink according to the combed direction.

Proposition 5.10 ([39, Proposition 7])
For a decomposable unique sink orientation of dimension d, one needs at most $d+1$ vertex evaluations to evaluate the sink. Moreover, $d+1$ is best possible.

Proof. Let s be a decomposable orientation and o its sink.

The observation above suggests the following algorithm: Start by evaluating an arbitrary vertex v_1, 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, both the sink o of the orientation and v_i are in a $(d-i+1)$-dimensional hypersink. This implies that v_{n+1} is the sink.

The claim is true for $i = 1$ as the whole cube is a hypersink (no outgoing edges). Suppose now that it is true for i. Since the orientation is decomposable, the hypersink $\mathfrak{C}_i$ containing v_i and o is combed in some label λ. Say $\mathfrak{C}_{i+1}$ is the facet of $\mathfrak{C}_i$ with incoming λ-edges. As $\mathfrak{C}_i$ is a hypersink of dimension $(d-i+1)$ the facet $\mathfrak{C}_{i+1}$ is a hypersink of dimension $d-i$.

Furthermore, since $v_i \in \mathfrak{C}_i$, only edges in $\mathfrak{C}_i$ can leave v_i. If $v_i \notin \mathfrak{C}_{i+1}$, then $\lambda \in s(v_i)$, otherwise $\lambda \notin s(v_i)$. In particular, $v_{i+1} = v_i \oplus s(v_i) \in \mathfrak{C}_{i+1}$.

For optimality, suppose an algorithm first requests the vertex v_1. We (the oracle) return the source, $s(v_1) = [d]$. Let v_2 be the second request and let $\lambda \in v_1 \oplus v_2$. Thus, v_2 and v_1 are in different λ-facets, say $v_2 \in \mathfrak{C}_2$. We reveal to the algorithm that the first combed label is λ, thus the sink is in $\mathfrak{C}_2$. There we follow a strategy (which exists by induction) which forces the algorithm to do d evaluations in $\mathfrak{C}_2$. This adds up to $d+1$ evaluations all together. $\square$

5.5 Jump Antipodal

Let JumpAntipodal be the following algorithm: In a vertex v with queried outmap $s(v)$ we jump to the vertex $v' = v \oplus s(v)$ and proceed until we reach the sink. The name comes from the fact that v' is the vertex antipodal to v in the subcube spanned by v and all outgoing edges in v. This is exactly the algorithm we described in Proposition 5.10. At first sight JumpAntipodal looks promising, especially since it is fast on the Klee-Minty cube.

Recall Proposition 4.29, where we proved that the simplex algorithm Simplex needs exponential many queries to find the sink of the Klee-Minty cube. JumpAntipodal is only a slight modification of Simplex in the sense, that instead choosing one of the outgoing edges we now consider all outgoing edges at once.

For the Klee-Minty cube this little twist makes a big difference. By construction, the Klee-Minty cube is combed. Thus, JumpAntipodal needs only a linear number of queries to find the sink. This indicates, that JumpAntipodal might be a good algorithm. In the rest of this chapter we will destroy this hope.

The behavior of JumpAntipodal on a USO s can be described by the following directed graph $T_{\text{ap}}(s)$: The vertex set of $T_{\text{ap}}(s)$ is the vertex set of the underlying cube of s. Two vertices v, v' form an edge $v \to v'$ if $v \oplus v' = s(v)$. In particular, every vertex v has exactly one outgoing edge $v \to v \oplus s(v)$. We call v' the *successor* of v. The sink o of s is special since it is the only vertex in $T_{\text{ap}}(s)$ having a loop. For instance, see Figure 5.12 and Figure 5.13.

The unique path starting in a vertex v in $T_{\text{ap}}(s)$ will be called *trace* of v. Obviously, the trace of a vertex v is the sequence of queries JumpAntipodal produces starting in v.

Figure 5.13 shows that JumpAntipodal can cycle. We will now show

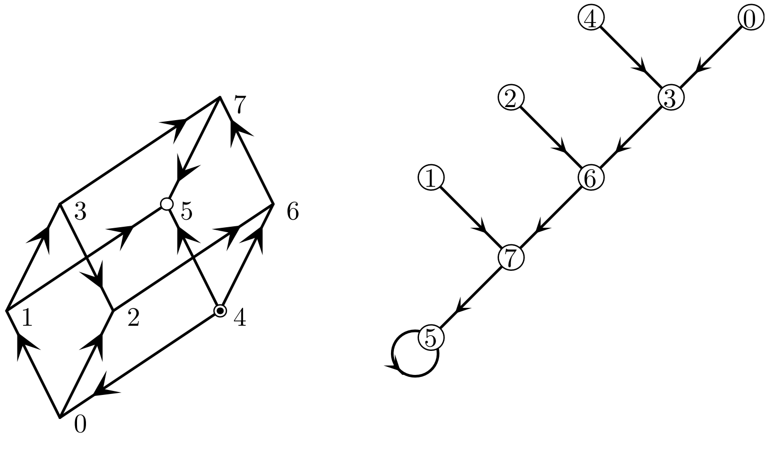


Figure 5.12: A USO s with a high graph $T_{\mathrm{ap}}(s)$.

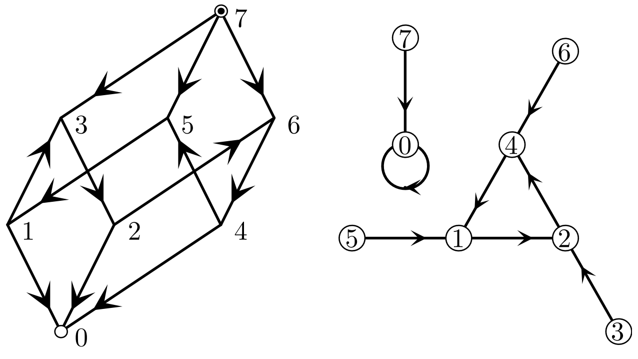


Figure 5.13: A USO s with a cyclic graph $T_{\mathrm{ap}}(s)$.

that this can only happen in cyclic USOs. The vertex v is a source in the cube spanned by v and $v \oplus s(v)$. The following lemma provides us with a path $v \rightarrow^* v \oplus s(v)$. Hence, the trace of JumpAntipodal induces a trail in s and if JumpAntipodal cycles, the induced trail contains a cycle of s.

Lemma 5.11
Let s be a unique sink orientation on $\mathfrak{C}$ and v a vertex in $\mathfrak{C}$. Then there is a path in the USO from the source of s to the sink of s via v.

Proof. We will prove the statement by induction on the dimension d of $\mathfrak{C}$. For $d = 1$ the statement is trivially true. Now assume, we known that in any $(d-1)$-dimensional USO there is a path from source to sink via a given vertex v.

For a d-dimensional USO s and a vertex v we will split the USO into two facets and apply induction to the facets. Let o be the sink of s and w its source. Choose $\lambda \in o \oplus w$. Then o and w are in different λ-facets $\mathfrak{C}_1$ and $\mathfrak{C}_2$, say $o \in \mathfrak{C}_1$ and $w \in \mathfrak{C}_2$. Let w_1 be the source of $\mathfrak{C}_1$ and o_2 the sink in $\mathfrak{C}_2$.

If $v \in \mathfrak{C}_1$, consider $w_1' = w_1 \oplus \{\lambda\}$. The vertex w_1' is in $\mathfrak{C}_2$ and by induction we find a path $w \rightarrow^* w_1'$. The λ-edge $\{w_1', w_1\}$ is directed towards w_1, since w_1 is not the global source. Thus, we find a path $w \rightarrow^* w_1$. But now by induction applied to $\mathfrak{C}_1$ we find a path $w_1 \rightarrow^* v \rightarrow^* o$. See the picture on the left in Figure 5.14.

If $v \in \mathfrak{C}_2$ we repeat the arguments from above with the interim vertex $o_2' = o_2 \oplus \{\lambda\}$, that is, we find a path

$$w \rightarrow^* v \rightarrow^* o_2 \rightarrow o_2' \rightarrow^* o.$$

See the picture on the right in Figure 5.14. □

In the following, we only consider USOs for which JumpAntipodal does not cycle. For such USO s the graph $T_{\mathrm{ap}}(s)$ is a tree, the so-called *bottom-antipodal tree.* This tree was first introduced by Kaibel [20] in connection to randomized simplex algorithms. In particular, Kaibel raised the question, how high such a tree can be. The height of a vertex v in $T_{\mathrm{ap}}(s)$ differs by one from the number of queries of JumpAntipodal starting in this vertex. Thus, if we can construct a family of examples

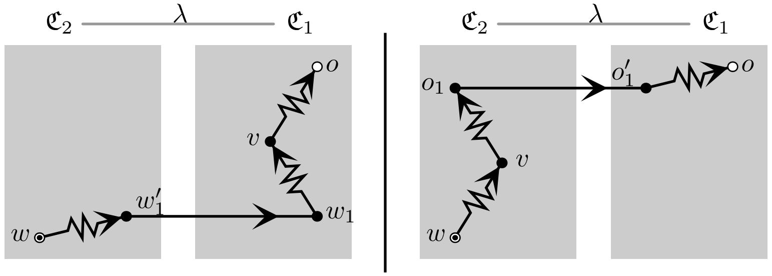


Figure 5.14: How to find a path trough v. The picture on the left side shows the situation when v and o are separated from w by a label λ. The picture on the right side shows the reverse situation, where v and w are separated from o.

for which the maximal height grows exponentially JumpAntipodal has complexity $\Omega(2^d)$. We will even show, that the average height (that is, the arithmetic mean of the heights of all vertices) grows exponentially.

Theorem 5.12
In every dimension d there is an acyclic USO, such that the height of the corresponding bottom-antipodal tree is at least $\sqrt{2}^d$ and the average height is $(\frac{\sqrt{5}}{2})^d$.

Proof. We will inductively construct acyclic USOs $s^{(d)}$ on $\mathfrak{C}^d$ with maximal height $h(d) = 2h(d-2)+2$ and average height $\tilde{h}(d) \geq \frac{5}{4}\tilde{h}(d-2)$.

Fix some orientations $s^{(2)}$ and $s^{(3)}$. For instance, we can choose the orientations of height $h(2)$ and $h(3)$, repectively. That is, for $s^{(2)}$ choose a bow and for $s^{(3)}$ choose the orientation in Figure 5.12.

Now assume we have already constructed $s^{(d-2)}$. Without restriction we assume that $s^{(d-2)}$ has its sink in $\emptyset$. Let $T = T_{\text{ap}}(s^{(d-2)})$ be its bottom-antipodal graph, o its sink and $v_{\max}$ a vertex of maximal height. Based on $s^{(d-2)}$ we use Lemma 4.20 and Lemma 4.18 to construct $s^{(d)}$.

As an intermediate step we construct a d-dimensional s' by applying Lemma 4.20 with frame $s^{(d-2)}$. The hypervertices are one of the two bows s_0 and s_1 with sink in $\emptyset$ and source in $\{d-1\}$ and $\{d\}$, respectively.

Vertices of $s^{(d-2)}$ with an even height in T are replaced by s_0 and vertices of odd height by s_1. Formally, for $v \in \mathfrak{C}^{d-2}$ with height congruent i modulo 2 set $s_v = s_i$ and let s' be the product of all s_v with frame $s^{(d-2)}$ according to Lemma 4.20.

Since s_0 and s_1 have the sink at the same vertex, by construction the subcube $* \cdots *00$ is a hypersink of s'. In this hypersink replace $s^{(d-2)}$ by $s^{(d-2)} \circ \oplus_{v_{\max}}$. That is, we change $s^{(d-2)}$ in $* \cdots *00$, such that the sinks of the other three copies in $* \cdots *01$, $* \cdots *10$, and $* \cdots *11$ are above the vertex $v_{\max}$ in $* \cdots *00$.

The resulting USO is $s = s^{(d)}$. As bows and $s^{(d-2)}$ are acyclic, so is s.

Color the vertices of $\mathfrak{C}^d$ according to their position in the $\{d-1, d\}$-hypervertices: a vertex is blue if it is in the sink of a hypervertex, it is green if in a hypervertex it is antipodal to the sink, and it is yellow if it is the source of a hypervertex. The remaining vertices are red. In other words, the vertices in $* \cdots *00$ are colored blue and the vertices in $* \cdots *11$ green. The remaining vertices are colored yellow if their $\{d-1, d\}$-edges are outgoing and red otherwise. We will refer to the set of blue vertices by B, the set of green vertices by G, the set of yellow vertices by Y, and the set of red vertices by R.

For a vertex v define $h_T(v)$ as the height of v in the corresponding copy of $s^{(d-2)}$. Formally, for a non-blue vertex v the height $h_T(v)$ is the height of $v \cap [d-2]$ in T. A blue v lives in a shifted copy of $s^{(d-2)}$, thus for such v the height $h_T(v)$ is the height of $v \cap [d-2] \oplus v_{\max}$ in T. In the following we want to express the height $h(v)$ in $T_{\mathrm{ap}}(s^{(d)})$ in terms of $h_T(v)$. The blue subcube is a hypersink. Thus, a blue vertex v has a blue successor. In particular, $h(v) = h_T(v)$.

Next we study the height of the vertices $0 \cdots 000$, $0 \cdots 010$, $0 \cdots 001$, and $0 \cdots 011$. By assumption, on $s^{(d-2)}$, the vertices $0 \cdots 010$, $0 \cdots 001$, and $0 \cdots 011$ are the sink in their corresponding copy of $s^{(d-2)}$. We exchanged the orientation in the subcube $* \cdots *00$, such that the sink and $v_{\max}$ change position. Hence, $0 \cdots 000$ has height

$$h(0 \cdots 000) = h_T(0 \cdots 000) = h(d-2).$$

By choice of s_0 the vertex $0 \cdots 010$ has only the $(d-1)$-edge outgoing.

Thus, its successor is $0\cdots 000$ and

$$h(0\cdots 010) = 1 + h(0\cdots 000) = 1 + h(d-2).$$

Similarly, the vertex $0\cdots 001$ has outgoing edges of label $d-1$ and d and its successor is $0\cdots 010$. Therefore

$$h(0\cdots 001) = 1 + h(0\cdots 010) = 2 + h(d-2).$$

Finally, the vertex $0\cdots 011$ has the d-edge outgoing and

$$h(0\cdots 011) = 1 + h(0\cdots 010) = 2 + h(d-2).$$

The remaining vertices are classified by their color. In general, red vertices have blue parents. Let v be a red vertex. It's outmap $s(v)$ consists of two components, $s(v) = s^{(d-2)}(v \cap [d-2]) \oplus s_i(v \cap \{d-1, d\})$. Since v is red, $s_i(v \cap \{d-1, d\})$ has one outgoing edge which points towards the sink of this hypervertex. Hence, $v \oplus s(v)$ relative to the corresponding hypervertex will be in the sink, i.e., blue.

A yellow vertex $v \neq 0\cdots 010$ has a yellow successor v' with height $h_T(v) = 1 + h_T(v')$. Again, we consider the two components $s(v) = s^{(d-2)}(v \cap [d-2]) \oplus s_i(v \cap \{d-1, d\})$. Relative to the frame $s^{(d-2)}$ the vertex $v \cap [d-2]$ has a successor v'_1. Relative to the hypervertex s_i the vertex $v \cap \{d-1, d\}$ has its antipodal vertex v'_2 as successor. Hence, the successor of v is $v' = v'_1 \oplus v'_2$. In particular, $h_T(v) = 1 + h_T(v')$. Since v'_1 has a different parity than $v \cap \{d-1, d\}$, we now are in a hypervertex of different type s_j. In s_j the source is antipodal to the source of s_i. Hence, $v' \oplus v''$ is yellow.

A green vertex $v \neq 0\cdots 011$ has a yellow successor. We can copy the argument from above. Again, $v'' = s_i(v)$ is antipodal to the source relative to s_i .

Now consider the green vertex $u = v_{\max} \cup \{d-1, d\}$, i.e., the green vertex with $h_T(u) = h(d-2)$. The trace of u will consist of yellow vertices, until we hit the yellow sink $0\cdots 010$. Hence, we get

$$\begin{aligned} h(d) \geq h(u) &= h_T(u) + h(0\cdots 010) \\ &= h(d-2) + 1 + h(d-2) = 2h(d-2) + 1. \end{aligned}$$

This proves $h(d) \geq \sqrt{2}^d$.

For the average height $\tilde{h}(d)$ we forget about the red vertices (as we don't know their height) and get

$$\begin{aligned}
\tilde{h}(d) &= \frac{1}{2^d}\sum_{v} h(v) \\
&\geq \frac{1}{2^d}\left(\sum_{v\in B} h(v) + \sum_{v\in Y} h(v) + \sum_{v\in G} h(v)\right) \\
&= \frac{1}{2^d}\left(\sum_{v\in B} h_T(v) + \sum_{v\in Y}(h_T(v) + 1 + h(d-2)) + \right. \\
&\qquad \left. + \sum_{v\in G}(h_T(v) + 2 + h(d-2))\right) \\
&= \frac{1}{4}\cdot\frac{1}{2^{d-2}}\sum_{v\in B} h_T(v) + \frac{1}{4}\cdot\frac{1}{2^{d-2}}\sum_{v\in Y} h_T(v) + \\
&\quad + \frac{1}{4}\cdot\frac{1}{2^{d-2}}\sum_{v\in G} h_T(v) + \frac{1}{2^d}(2^{d-2} + 2^{d-1} + 2^{d-1}h(d-2)) \\
&\geq \frac{3}{4}\tilde{h}(d-2) + \frac{1}{2}h(d-2) \\
&\geq \frac{5}{4}\tilde{h}(d-2).
\end{aligned}$$

This proves that $\tilde{h}(d) > (\frac{\sqrt{5}}{2})^d$. $\square$

5.6 Remarks

In terms of classical complexity theory, Theorem 5.3 relies on the fact that a polynomial unique sink oracle induces a *total* relation. The class of all total relations in FNP is called TFNP. It is known that a FNP-complete problem in TFNP would imply NP = coNP (see [31, Theorem 2.1]). In this sense, the main part of Theorem 5.3 is to show that a polynomial unique sink oracle defines a language in TFNP.

The best known algorithms for solving Sink can be found in [42]: The algorithm FibonacciSeesaw needs $O(1.61^d)$ queries to find the sink in a d-dimensional USO. The best randomized algorithm needs $O(1.44^d)$ expected queries. If we restrict our attention to acyclic orientations, the RandomFacet algorithm solves the problem in $O(e^{2\sqrt{d}})$ queries (see [12]).

So far, there is no indication whether there is a superpolynomial lower bound or a polynomial algorithm for Sink. There is still too little structural insight into USOs. The crucial point is the global dependency between vertices, i.e., the phases. On the one hand such global relations make it difficult to construct examples. On the other hand, we do not really know how to exploit these dependencies to speed up algorithms.

For instance, the SevenStepsToHeaven algorithm [42] heavily uses dependencies between vertices to rule out possible positions of the sink. Attempts have been made to repeat this in dimension five [4]. However, already there the relations get very involved. It is believed that the true value for $t(5)$ is 10 or 11.

The simplex algorithm originates from linear programming. In this setting there are many possible pivot rules even in the memoryless setting. Still, for most pivot rules, variants of the Klee-Minty cube show an exponential running time. From this point of view Proposition 4.29 is a reformulation of the classical result [24].

If we allow memory, there is one pivot rule which survived all attacks over the course of 24 years. The running time of Zadeh's rule [43] is still not known. In the language of USOs Zadeh's rule reads: Leave the facet you left the least number of times. Or from a different perspective: from the outgoing edges, choose the *label* you chose the least often before. Note, that on cyclic USOs Zadeh's rule can cycle.

The randomized variant of Simplex (called RandomEdge) which chooses among the outgoing edges one uniformly at random is known to perform bad on USOs. On the Klee-Minty cube for RandomEdge the expected number of queries is $O(d^2)$ (see [14]). For cyclic USOs, cases are known where the expected number of queries RandomEdge performs is much higher than the number of vertices (see [33]). Finally, a recent result by Matoušek and Szabó shows that on acyclic USOs RandomEdge needs an exponential number of queries [29].

6 Data

If you prick me, do I not... leak?

(Data)

The best known algorithms for finding the sink of a USOs, FibonacciSeesaw and the (randomized) product algorithm both benefit from knowledge about low-dimensional cases. The product algorithm uses an optimal randomized algorithm in dimension three, whereas FibonacciSeesaw benefits from an optimal deterministic algorithm in dimension four.

In the following we concentrate on questions arising while studying low dimensional USOs. In Section 6.1 we describe a format for storing USOs. In Section 6.2 we address the two algorithmic problems of isomorphism test and generating all USOs of a given dimension. In the remainder of this chapter we describe the data we have gained. A list of all 3- and 4-dimensional USOs can be found at [38].

6.1 Storing Orientations

In this section we will describe a data structure for storing and/or describing unique sink orientations. Our main concern is not a space-optimal encoding of USOs, but rather a human readable representation.

The data structure is given in the form of a primitive stack language. The language itself include only five types of tokens:

1. $\langle$`int`$\rangle$ $:= [0-9]^+$
2. $\langle$`label`$\rangle$ $:=$ '`.`'$[1-9][0-9]^*$
3. commands `product`, `flip`, `mirror`, `relabel`, `change`, and `reorient`
4. special USOs `uniform`, `sw`, `km`, `eye`, and `bow`
5. brackets '{}', '`()`', '`(())`', and '`[]`'

Tokens are separated by whitespaces. In addition, the stack can store the following data types: $\langle$`set`$\rangle$, $\langle$`cube`$\rangle$, $\langle$`perm`$\rangle$, and $\langle$`uso`$\rangle$. The syntax of the language will be highly context sensitive. Thus, we shall give the semantics of the reduction rules rather than completely define a formal grammar.

The primitive type $\langle\texttt{int}\rangle$ represents a positive integer. Nevertheless, depending on the context we sometimes interpret a positive integer as the characteristic vector of a set. That is, we have the reduction rule

$$\begin{array}{rcl} \langle\texttt{int}\rangle & \rightarrow & \langle\texttt{set}\rangle \\ n & \rightarrow & \left\{\lambda \in \mathbb{N}^{+} \;\middle|\; \lfloor n/2^{\lambda-1}\rfloor \equiv 0 \,(\mathrm{mod}\, 2)\right\}, \end{array}$$

which is triggered whenever a $\langle\texttt{set}\rangle$ is required and an $\langle\texttt{int}\rangle$ is given.

The primitive type $\langle\texttt{label}\rangle$ can be seen as syntactic sugar. It allows to distinguish between sets and their elements. This gives us a second possibility to write down a set of positive numbers:

$$\begin{array}{rcl} \text{'\{'}\ \langle\texttt{label}\rangle^{*}\ \text{'\}'} & \rightarrow & \langle\texttt{set}\rangle \\ \{\ \texttt{.}\lambda_1 \cdots\ \texttt{.}\lambda_k\ \} & \rightarrow & \{\lambda_1, \ldots, \lambda_k\} \end{array}$$

For instance, the set $\{1, 2, 5\}$ can be written as '`{ .1 .2 .5 }`' or as '`19`'.

The brackets '{}' have a second semantics. If the first token inside the brackets can be reduced to $\langle\texttt{set}\rangle$, the term is interpreted as a cube. There are two possible ways to determine a cube. The first reduction rule is

$$\begin{array}{rcl} \text{'\{'}\ \langle\texttt{set}\rangle\ \langle\texttt{label}\rangle^{*}\ \text{'\}'} & \rightarrow & \langle\texttt{cube}\rangle \\ \{\ v\ \ \texttt{.}\lambda_1 \cdots\ \texttt{.}\lambda_k\ \} & \rightarrow & \{u \mid \exists \Lambda \subseteq \{\lambda_1, \ldots, \lambda_k\} : u = v \oplus \Lambda\}. \end{array}$$

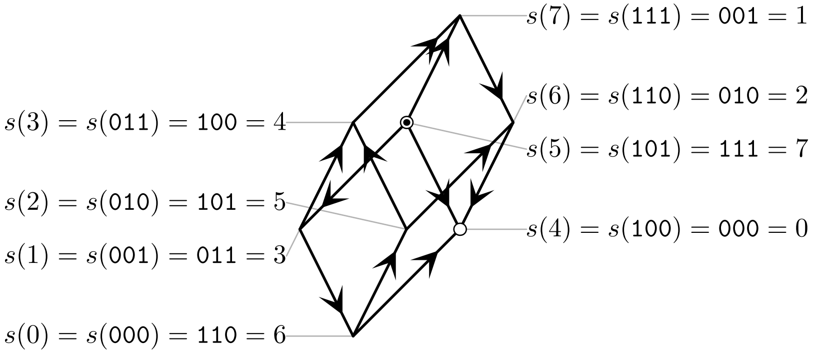
That is, we give a base vertex v and the carrier $\{\lambda_1, \ldots, \lambda_k\}$ of the cube. In particular, the λ-edge in a vertex v is denoted by "$\{\ v\ \ \texttt{.}\lambda\ \ \}$". Alternatively,

$$\begin{array}{rcl} \text{'\{'}\ \langle\texttt{set}\rangle\ \langle\texttt{set}\rangle\ \text{'\}'} & \rightarrow & \langle\texttt{cube}\rangle \\ \{\ u\ \ v\ \} & \rightarrow & \{w \mid u \cap v \subseteq w \subseteq u \cup v\} \end{array}$$

describes the minimal cube containing u and v. For instance, the terms '`{0 .1 .2 .3}`' and '`{1 6}`' both describe the standard 3-dimensional cube.

A USO can be defined in terms of its outmap. An outmap of a d-dimensional USO is given by

$$\begin{array}{rcl} \text{'['}\ \langle\texttt{set}\rangle_0 \cdots\ \langle\texttt{set}\rangle_{2^d-1}\ \text{']'} & \rightarrow & \langle\texttt{uso}\rangle \\ \texttt{[}\ t_0 \cdots\ t_{2^d-1}\ \texttt{]} & \rightarrow & s : 2^{[d]} \rightarrow 2^{[d]}, v \mapsto t_{\chi(v)}, \end{array}$$



$s(7) = s(\texttt{111}) = \texttt{001} = 1$
$s(6) = s(\texttt{110}) = \texttt{010} = 2$
$s(3) = s(\texttt{011}) = \texttt{100} = 4$
$s(5) = s(\texttt{101}) = \texttt{111} = 7$
$s(2) = s(\texttt{010}) = \texttt{101} = 5$
$s(4) = s(\texttt{100}) = \texttt{000} = 0$
$s(1) = s(\texttt{001}) = \texttt{011} = 3$
$s(0) = s(\texttt{000}) = \texttt{110} = 6$

Figure 6.1: The orientation $s =$ `[ 6 3 5 4 0 7 2 1 ]`. Each value is denoted as a 0/1-word as well as the integer interpretation of the word.

where $\chi(v) = \sum_{\lambda \in v} 2^{\lambda-1}$. See Figure 6.1 for an example. Syntactically we cannot distinguish outmaps of USOs from arbitrary maps. Whenever a term "`[` $v_1 \ \dots v_{2^d}$ `]`" does not reduce to a USO, the semantics will be undefined. All USOs will be on a standard cube $\mathfrak{C}^d$.

The remaining elements of the language aim to manipulate USOs already defined. To define a relabeling manipulator, we have to formulate permutations. Permutations can be given in two different ways. The following rule defines a permutation in S_d:

$$\begin{aligned} \text{'}\texttt{((}\text{'}\ \langle\texttt{int}\rangle \cdots \ \langle\texttt{int}\rangle_d\ \text{'}\texttt{))}\text{'} &\ \rightarrow\ \langle\texttt{perm}\rangle \\ \texttt{((}\ t_1 \cdots \ t_d\ \texttt{))} &\ \rightarrow\ \tau : [d] \rightarrow [d]\,, \lambda \mapsto t_\lambda, \end{aligned}$$

provided τ is a permutation, i.e., $\{t_1, \dots, t_d\} = [d]$. We use double brackets to distinguish such an exhaustive enlisting of τ from its cycle decomposition. A cycle in S_d is defined by

$$\begin{aligned} \text{'}\texttt{(}\text{'}\ \langle\texttt{int}\rangle \cdots \ \langle\texttt{int}\rangle_k\ \text{'}\texttt{)}\text{'} &\ \rightarrow\ \langle\texttt{perm}\rangle \\ \texttt{(}\ t_1 \cdots \ t_k\ \texttt{)} &\ \rightarrow\ \tau : [d] \rightarrow [d] \\ &\qquad \lambda \mapsto \begin{cases} \lambda, & \lambda \notin \{t_1, \dots, t_k\} \\ t_1, & \lambda = t_k \\ t_{i+1}, & \lambda = t_i,\ i < k \end{cases} \end{aligned}$$

provided that $k < d$ and $\{t_1, \ldots, t_k\} \subseteq \binom{[d]}{k}$. The value of d will always be clear from the context. Furthermore, two permutations following each other are composed:

$$\begin{array}{rcl} \langle\texttt{perm}\rangle\ \langle\texttt{perm}\rangle & \to & \langle\texttt{perm}\rangle \\ \tau_1\ \tau_2 & \to & \tau_1 \circ \tau_2. \end{array}$$

For instance, the terms "`((2 3 1 4))`", "`(1 2 3)`", and "`(1 3) (1 2)`" all define the same permutation in S_4.

We are now ready to list the manipulation operations. Given a permutation $\tau \in S_d$ and a d-dimensional USO s, the rule for the relabeled USO $s_\tau = \tau' \circ s \circ {\tau'}^{-1}$ described in Lemma 4.4) is as follows:

$$\begin{array}{rcl} \langle\texttt{uso}\rangle\ \langle\texttt{perm}\rangle\ \texttt{'relabel'} & \to & \langle\texttt{uso}\rangle \\ s\ \tau\ \texttt{relabel} & \to & s_\tau : v \mapsto \tau[s(\tau^{-1}[v])]. \end{array}$$

The operations $s \to s \circ \oplus_a$ and $s \to \oplus_a \circ s$ from Proposition 4.5 are defined similarly. Let s be a d-dimensional USO and $v \subseteq [d]$. Then

$$\begin{array}{rcl} \langle\texttt{uso}\rangle\ \langle\texttt{set}\rangle\ \texttt{'mirror'} & \to & \langle\texttt{uso}\rangle \\ s\ v\ \texttt{mirror} & \to & s \circ \oplus_v \end{array}$$

and

$$\begin{array}{rcl} \langle\texttt{uso}\rangle\ \langle\texttt{set}\rangle\ \texttt{'reorient'} & \to & \langle\texttt{uso}\rangle \\ s\ v\ \texttt{reorient} & \to & \oplus_v \circ s. \end{array}$$

Let s be a USO on $\mathfrak{C}^d$ and $\mathfrak{C}_0$ a d'-dimensional hypervertex of s. Futhermore let $\hat{s}_0$ be a USO on $\mathfrak{C}^{d'}$. In order to substitute the orientation on $\mathfrak{C}_0$ by $\hat{s}_0$, we must rename the labels in $\hat{s}_0$. Let $\operatorname{carr} \mathfrak{C}_0 = \{\lambda_1, \ldots, \lambda_{d'}\}$ with $\lambda_1 < \lambda_2 < \cdots < \lambda_{d'}$ and $\tau : \{\lambda_1, \ldots, \lambda_{d'}\} \to [d']$ the bijection defined by $\tau(\lambda_i) = i$. Set $s_0 = \tau' \circ \hat{s}_0 \circ {\tau'}^{-1}$. Then Corollary 4.19 is applicable to s and s_0. The resulting USO s' is denoted by

$$\begin{array}{rcl} \langle\texttt{uso}\rangle\ \langle\texttt{uso}\rangle\ \langle\texttt{cube}\rangle\ \texttt{'change'} & \to & \langle\texttt{uso}\rangle \\ s\ \hat{s}_0\ \mathfrak{C}_0\ \texttt{change} & \to & s'. \end{array}$$

Let $\bar{s}$ be a USO on $\mathfrak{C}^d$ and $\hat{s}_0, \ldots, \hat{s}_{2^d-1}$ USOs on $\mathfrak{C}^{d'}$. To form a product with frame orientation $\bar{s}$ and hypervertices s_v, the carrier of

$\bar{s}$ needs to be different from the carriers of the s_v's. Let $\tau : [d'] \to \{d+1, \dots, d+d'\}$ be the bijection $\tau(\lambda) = d + \lambda$. For $v \in 2^{[d]}$ and $i = \sum_{\lambda \in v} 2^{\lambda-1}$ define $s_v = \tau' \circ \hat{s}_i \circ {\tau'}^{-1}$. Then all s_v are USOs on $\mathfrak{C}^{\{d+1,\dots,d+d'\}}$ and we can apply Lemma 4.20 to $\bar{s}$ and s_v, $v \in 2^{[d]}$. The resulting USO s on $\mathfrak{C}^{d+d'}$ is denoted by

$$\begin{aligned} \langle\texttt{uso}\rangle_0 \cdots \ \langle\texttt{uso}\rangle_{2^d-1} \ \langle\texttt{uso}\rangle \ \texttt{'product'} \quad &\to \quad \langle\texttt{uso}\rangle \\ \hat{s}_0 \cdots \ \hat{s}_{2^d-1} \ \bar{s} \ \texttt{product} \quad &\to \quad s. \end{aligned}$$

Finally, let s be a USO on $\mathfrak{C}^d$ and e an edge in $\mathfrak{C}^d$. Let s' be the USO we obtain from s by flipping all edges in phase to e (cf. Proposition 4.9). The rule for s' is

$$\begin{aligned} \langle\texttt{uso}\rangle \ \langle\texttt{cube}\rangle \ \texttt{'flip'} \quad &\to \quad \langle\texttt{uso}\rangle \\ s \ e \ \texttt{flip} \quad &\to \quad s'. \end{aligned}$$

We are now ready to define some special USOs. An eye is completely determined by the position of its sink. The eye with sink in v is given by

$$v \ \texttt{eye} \to \texttt{[0 1 2 3]} \ v \ \texttt{mirror}.$$

On the other hand, a bow is given by the position of the sink and the label along which we find the source. The bow with sink in v and source in $v \oplus \{\lambda\}$ is:

$$v \ \texttt{.}\lambda \ \texttt{bow} \to v \ \texttt{eye} \ \{ \ (v \oplus \{1,2\}) \ \texttt{.}\bar{\lambda} \ \} \ \texttt{flip}.$$

The uniform orientation can be defined recursively:

$$\begin{aligned} (d+1) \ \texttt{uniform} \quad &\to \quad d \ \texttt{uniform} \ d \ \texttt{uniform} \ \texttt{[0 1]} \ \texttt{product} \\ 1 \ \texttt{uniform} \quad &\to \quad \texttt{[0 1]}. \end{aligned}$$

The same can be achieved for Klee-Minty cubes:

$$\begin{aligned} (d+1) \ \texttt{km} \quad &\to \quad d \ \texttt{km} \ d \ \texttt{km} \ (2^d - 1) \ \texttt{reorient} \ \texttt{[0 1]} \ \texttt{product} \\ 1 \ \texttt{km} \quad &\to \quad \texttt{[0 1]}. \end{aligned}$$

We close this section by defining terms for the 3-dimensional USOs described in [40]. Their relevance will become clear later. The corresponding reduction rules are defined in Table 6.1, a sketch of the orientations can be found in Figure 6.2.

1 `sw`	$\rightarrow$	`0 .2 bow 3 eye [ 0 1 ] product`
2 `sw`	$\rightarrow$	`0 eye 3 eye [ 0 1 ] product`
3 `sw`	$\rightarrow$	`0 .2 bow 1 .1 bow [ 0 1 ] product`
4 `sw`	$\rightarrow$	`3 sw` { `6 .1` } `flip`
5 `sw`	$\rightarrow$	`0 .2 bow 2 .2 bow [ 0 1 ] product`
6 `sw`	$\rightarrow$	`0 eye 1 .1 bow [ 0 1 ] product`
7 `sw`	$\rightarrow$	`0 .2 bow 1 eye [ 0 1 ] product`
8 `sw`	$\rightarrow$	`0 .2 bow 3 .1 bow [ 0 1 ] product`
9 `sw`	$\rightarrow$	`0 eye 1 eye [ 0 1 ] product`
10 `sw`	$\rightarrow$	`0 .1 bow 1 eye [ 0 1 ] product`
11 `sw`	$\rightarrow$	`0 .1 bow 3 .1 bow [ 0 1 ] product`
12 `sw`	$\rightarrow$	`0 eye 3 .1 bow [ 0 1 ] product`
13 `sw`	$\rightarrow$	`0 eye 0 .2 bow [ 0 1 ] product`
14 `sw`	$\rightarrow$	`0 .1 bow 0 .2 bow [ 0 1 ] product`
15 `sw`	$\rightarrow$	`3 uniform`
16 `sw`	$\rightarrow$	`0 eye 1 .2 bow [ 0 1 ] product`
17 `sw`	$\rightarrow$	`0 .2 bow 1 .2 bow [ 0 1 ] product`
18 `sw`	$\rightarrow$	`0 .2 bow 2 .1 bow [ 0 1 ] product`
19 `sw`	$\rightarrow$	`3 uniform` {`1 .2`} `flip` {`2 .3`} `flip` {`4 .1`} `flip`

Table 6.1: A definition for the 19 Stickney-Watson orientations.

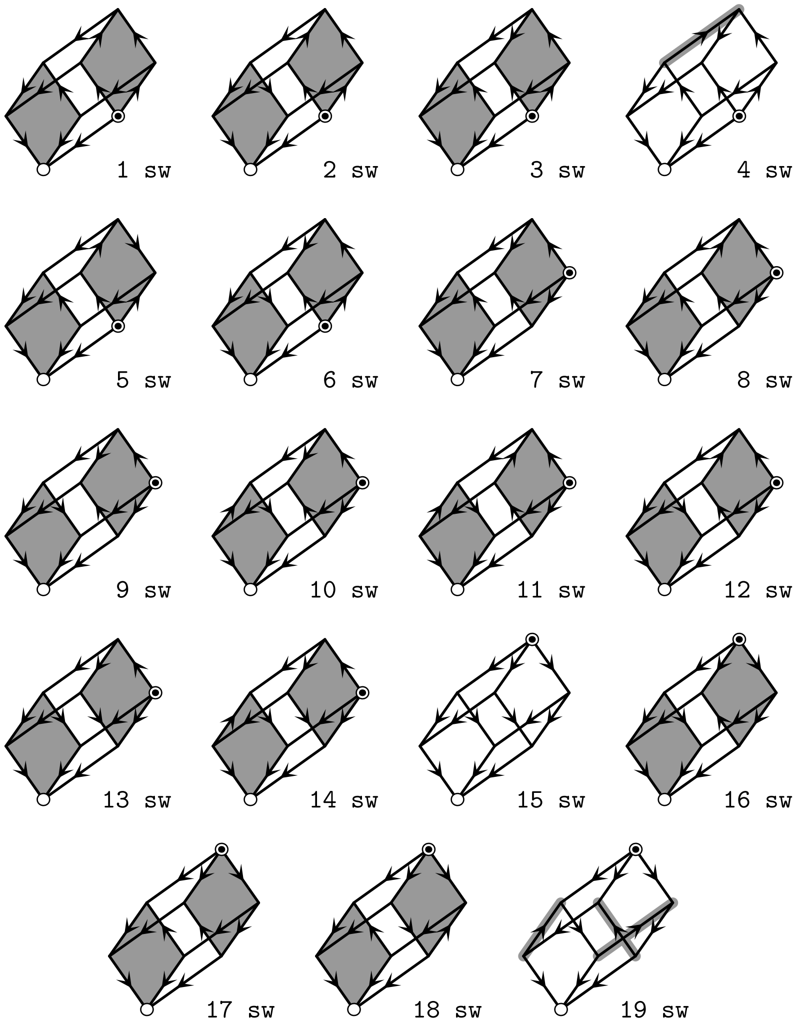


Figure 6.2: The 19 Stickney-Watson orientations. The gray area indicates how they are constructed, i.e., of which subcubes they are a product or which edges are flipped.

6.2 Implementational Aspects

An implementation of the operations in the last section is straightforward. The only operation which is a bit more sophisticated is the `flip`, since `flip` must know the phase of an edge. This problem is equivalent to finding the connected component in the following graph G. The vertices of G are the edges of the cube, where two edges of the cube are connected if they are in direct phase. In the following we will study two other natural problems: isomorphism test and enumeration.

6.2.1 Isomorphism Test

Let s be the uniform orientation on $\mathfrak{C}_d$ and $s_e = (s\ e\ \texttt{flip})$ for $e \in \mathrm{E}(\mathfrak{C}^d)$. Let $s_1, s_2 \in \{s\} \cup \left\{ s_e \;\middle|\; e \in \mathrm{E}(\mathfrak{C}^d) \right\}$. Any algorithm for testing *equality* of s_1 and s_2 can be forced to to query 2^{d-1} vertices. In fact, as long as the algorithm queried less than 2^{d-1} vertices there is one edge e not incident to a queried vertex. Thus, the algorithm cannot distinguish the case $s_1 = s$ and $s_2 = s_e$ from the case $s_1 = s = s_2$. In particular, we cannot expect to perform an isomorphism test in $o(2^d)$ oracle queries. We therefore assume in the following that we have full access to the outmap of an USO.

according to Lemma 4.4 two d-dimensional USOs s_1 and s_2 on $\mathfrak{C}_1$ and $\mathfrak{C}_2$ are isomorphic if there exist $a_1 \in \mathrm{V}(\mathfrak{C}_1)$, $a_2 \in \mathrm{V}(\mathfrak{C}_2)$ and a bijection $\tau : \operatorname{carr} \mathfrak{C}_1 \to \operatorname{carr} \mathfrak{C}_2$, such that

$$\tau' \circ s_1 \circ \oplus_{a_1} = s_2 \circ \oplus_{a_2} \circ \tau'.$$

A brute force approach would test all $2^d \cdot d!$ possible choices for a_2 and τ. Unfortunately, we are not able to improve much on this worst case behavior. Note, that the brute force method performs at least $d!$ test for *each* pair of USOs.

In the following we introduce a heuristic which performs much better in practice. We apply the usual trick and introduce a set of invariants.

Let s_1 and s_2 be two d-dimensional USOs. If they are isomorphic an isomorphism has to map sink to sink. Thus, as a first step we find the sinks o_1 of s_1 and o_2 of s_2. The orientations $\bar{s}_i = s_i \circ \oplus_{o_i}$ are orientations on $\mathfrak{C}^d$ with sink in $\emptyset$. It is rather straight-forward to prove the following.

Lemma 6.1
For $i = 1, 2$, let s_i be a d-dimensional USOs with sinks o_i and define $\bar{s}_i = s_i \circ \oplus_{o_i}$. Then s_1 and s_2 are isomorphic if and only if there is a permutation $\tau \in S_d$, such that $\tau' \circ \bar{s}_1 = \bar{s}_2 \circ \tau'$.

In the remainder of this section we will assume that all USOs are on the standard cube and have their sink in $\emptyset$. Notice that by Lemma 6.1 for each USO s on $\mathfrak{C}^d$ the set

$$\left\{ \tau' \circ s \circ \tau'^{-1} \;\middle|\; \tau \in S_d \right\}$$

contains all and only the USOs which are isomorphic to s.

Definition 6.2
Let s be a USO on $\mathfrak{C}^d$ with sink in $\emptyset$. For $i, j \in [d]$ and $\lambda \in [d]$ let

$$a_{i,j}^{(\lambda)}(s) := \left| \left\{ v \in \binom{[d] \setminus \{\lambda\}}{i-1} \;\middle|\; |s(v) \oplus s(v \oplus \{\lambda\})| = j \right\} \right|$$

Furthermore, set $A_i^{(\lambda)}(s) := (a_{i,1}^{(\lambda)}(s), \ldots, a_{i,d}^{(\lambda)}(s))$.

For a USO s on $\mathfrak{C}^d$ with sink in $\emptyset$, the number $a_{i,1}^{(\lambda)}(s)$ counts all flippable λ-edges between sets of cardinality $i - 1$ and sets of cardinality i. Any isomorphism test has to consider these flippable edges. In particular, flippable edges have to map to flippable edges.

Lemma 6.3
Let s be a USO on $\mathfrak{C}^d$ with sink in $\emptyset$, $\tau \in S_d$, and $i \in [d]$. Then for $s' = \tau' \circ s \circ \tau'^{-1}$ we have

$$A_i^{(\tau(\lambda))}(s') = A_i^{(\lambda)}(s).$$

The proof, although technically a bit involved, is straight-forward and will be omitted.

By Lemma 6.3, $A_i^{(\lambda)}(s)$ itself is not invariant under isomorphisms, but the multi-set of all $A_i^{(\lambda)}(s)$, $\lambda \in [d]$ is. Furthermore, an isomorphism has to preserve the value of $A_i^{(\lambda)}(s)$.

Definition 6.4
Let s be a USO on $\mathfrak{C}^d$ with sink in $\emptyset$ and $i \in [d]$. Let

$$\mathcal{L}_i(s) = \{\Lambda_1^{(i,s)}, \dots, \Lambda_{k_{i,s}}^{(i,s)}\}$$

be the partition of $[d]$ defined by

1. *$\lambda, \lambda' \in \Lambda_j^{(i,s)} \Rightarrow A_i^{(\lambda)}(s) = A_i^{(\lambda')}(s)$ and*
2. *$\lambda_1 \in \Lambda_{j_1}^{(i,s)}, \lambda_2 \in \Lambda_{j_2}^{(i,s)}, j_1 < j_2 \Rightarrow A_i^{(\lambda_1)}(s) <_{\text{lex}} A_i^{(\lambda_2)}(s)$.*

Furthermore, for $\lambda \in [d]$, the multi-set $\mathcal{I}_i(s)$ of all $A_i^{(\lambda)}(s)$, $\lambda \in [d]$ is called the i-th iso-set *of s.*

As an easy consequence of Lemma 6.3 we get the following:

Corollary 6.5
Let s_1 and s_2 be two isomorphic USOs on $\mathfrak{C}^d$ with sink in $\emptyset$, and let $\tau \in S_d$ be a permutation with $\tau' \circ s_1 = s_2 \circ \tau'$. Then, for all $i \in [d]$ the two multi-sets $\mathcal{I}_i(s_1)$ and $\mathcal{I}_i(s_2)$ agree. Furthermore, for $1 \leq j \leq k_{i,s_1}$, we have $\tau[\Lambda_j^{(i,s_1)}] = \Lambda_j^{(i,s_2)}$.

Let $\mathcal{M}$ and $\mathcal{N}$ be two partitions of $[d]$. The set

$$\mathcal{M} \cap \mathcal{N} := \{M \cap N \mid M \in \mathcal{M}, N \in \mathcal{N}\} \setminus \{\emptyset\}$$

is a partition of $[d]$. It is the largest common refinement of $\mathcal{M}$ and $\mathcal{N}$. Furthermore, for a permutation $\tau \in S_d$ let

$$\tau[\mathcal{M}] := \{\tau[M] \mid M \in \mathcal{M}\},$$

which is a partition again.

Based on Corollary 6.5 and the operations described above, Table 6.2 describes an isomorphism test, which seems to perform well in practice. The use of $\mathcal{I}_i(s)$ often allows to decide that two USOs are not isomorphic. But even if two USOs have the same invariants, the algorithm is able to cut down the number of possible relabelings by $\mathcal{L}_i(s)$. For example, in dimension four of the 14614 isomorphy classes 9930 classes are uniquely determined by their iso-sets. Furthermore, for 10641 of the

```
isotest($s_1, s_2$)
Preconditions: $s_1$ and $s_2$ are on $\mathfrak{C}^d$ with sink in $\emptyset$.
Postcondition: returns true if $s_1$ and $s_2$ are isomorphic,
               false otherwise.
  $\mathcal{M}_1 \leftarrow \{[d]\}$
  $\mathcal{M}_2 \leftarrow \{[d]\}$
  for each $i \in [d]$ do
    compute iso-sets $\mathcal{I}_i(s_1)$ and $\mathcal{I}_i(s_2)$
    if $\mathcal{I}_i(s_1) \neq \mathcal{I}_i(s_2)$ then
      return false
    compute $\mathcal{L}_i(s_1)$ and $\mathcal{L}_i(s_2)$
    $\mathcal{M}_1 \leftarrow \mathcal{M}_1 \cap \mathcal{L}_i(s_1)$
    $\mathcal{M}_2 \leftarrow \mathcal{M}_2 \cap \mathcal{L}_i(s_2)$
  for each $\tau \in S_d$ with $\tau[\mathcal{M}_2] = \mathcal{M}_1$
    if $\tau' \circ s_1 = s_2 \circ \tau'$
      return true
  return false
```

Table 6.2: A procedure to test if two USOs are isomorphic. The USOs s_1 and s_2 are assumed to be on $\mathfrak{C}^d$ with sink in $\emptyset$.

14614 isomorphy classes the lexicographic order of the iso-sets allows only one relabeling.

Let s_1 and s_2 be two USOs on $\mathfrak{C}^d$, both with their sink in $\emptyset$. A permutation $\tau \in S_d$ has to fulfill $\tau[\Lambda_j^{(i,s_1)}] = \Lambda_j^{(i,s_2)}$ for all $1 \leq j \leq k_{i,s_1}$ and all i. In many cases this narrows the choices of τ down to one permutation.

The procedure in Table 6.2 needs only $O(d^2)$ storage in addition to the outmaps of s_1 and s_2. In particular, the algorithm can perform an isomorphism test even in higher dimensions (say dimensions large than 64[1]); however we need to be willing to ask the oracle for a vertex

[1] In order to store the whole outmap of a d-dimensional USO we have to address 2^d objects. This limits the dimension to the bus-size of a computer. Although there are ways around this limitation, the solutions are highly impractical.

repeatedly, rather than storing the outmaps.

6.2.2 Enumeration

Given a d-dimensional USO s, the induced orientations s_1 and s_2 on the lower and upper d-facet are $(d-1)$-dimensional USOs. The edges inbetween partition into phases. Since edges in phase have to be oriented towards the same facet, s is uniquely described by s_1, s_2, and the orientation of one edge per phase.

This suggests the following procedure to generate all d-dimensional USOs provided we know all $(d-1)$-dimensional USOs. For each pair of $(d-1)$-dimensional USOs s_1 and s_2 apply:

1. Set $s = (s_1\ s_2\ [01]\ \texttt{product})$, i.e., s is the orientation which has s_1 in the lower d-facet, s_2 in the upper d-facet, and all d-edges directed towards the lower d-facet.
2. Calculate the d-phases of s.
3. For each set of d-phases, flip the set and output the resulting orientation.

This will enumerate all d-dimensional USOs.

The expensive operation in the above procedure is to compute all d-phases. In the following we want to minimize the number of phase calculations. We can do so by grouping USOs with essentially the same set of phases. For instance, isomorphic USOs have isomorphic phases. (So far, we have not proven this).

Definition 6.6
Two d-dimensional USOs s_1 and s_2 on $\mathfrak{C}^d$ are of the same o-type *if there exist $a, \Lambda \subseteq [d]$ and $\tau \in S_d$, such that*

$$\tau' \circ s_1 \circ \oplus_a = \oplus_\Lambda \circ s_2 \circ \tau'. \tag{6.1}$$

As it will turn out later, USOs of the same o-type will produce essentially the same orientation in the above procedure. But let us first show that the o-types partition USO(d).

Lemma 6.7
The relation "being of the same o-type" is an equivalence relation on USO(d)*.*

Proof. Obviously, a d-dimensional USO s is of the same o-type as s. This implies that the relation is reflexive.

Now let s_1 and s_2 be of the same o-type, say $\tau' \circ s_1 \circ \oplus_a = \oplus_\Lambda \circ s_2 \circ \tau'$. Consider $\Lambda' = \tau^{-1}[\Lambda]$ and $a' = \tau[a]$. For $v \in 2^{[d]}$ we get

$$\begin{aligned}
{\tau'}^{-1} \circ s_2 \circ \oplus_{a'}(v) &= \tau^{-1}[s_2(v \oplus \tau[a])] \\
&= \tau^{-1}[\Lambda \oplus \Lambda \oplus s_2(\tau[\tau^{-1}[v] \oplus a])] \\
&= \tau^{-1}[\Lambda \oplus \tau[s_1(\tau^{-1}[v] \oplus a \oplus a)]] \\
&= \tau^{-1}[\Lambda] \oplus s_1(\tau^{-1}[v]) \\
&= \oplus_{\Lambda'} \circ s_1 \circ {\tau'}^{-1}(v).
\end{aligned}$$

Hence, ${\tau'}^{-1} \circ s_2 \circ \oplus_{a'} = \oplus_{\Lambda'} \circ s_1 \circ {\tau'}^{-1}$ and the relation is symmetric.

For transitivity let s_1, s_2, and s_3 be USOs on $\mathfrak{C}^d$, such that s_1 and s_2 are of the same o-type and so are s_2 and s_3. Choose $a_1, \Lambda_1 \subseteq [d]$ and $\tau_1 \in S_d$ with $\tau_1' \circ s_1 \circ \oplus_{a_1} = \oplus_{\Lambda_1} \circ s_2 \circ \tau_1'$. Furthermore, choose $a_2, \Lambda_2 \subseteq [d]$ and $\tau_2 \in S_d$ with $\tau_2' \circ s_2 \circ \oplus_{a_2} = \oplus_{\Lambda_2} \circ s_3 \circ \tau_2'$. Let $\Lambda_3 = \tau_2[\Lambda_1] \oplus \Lambda_2$, $a_3 = a_1 \oplus {\tau_1}^{-1}[a_2]$, and $\tau_3 = \tau_2 \circ \tau_1$. Then for $v \in 2^{[d]}$ we get

$$\begin{aligned}
\tau_3' \circ s_1 \circ \oplus_{a_3}(v) &= \tau_2\Big[\tau_1\big[s_1(v \oplus a_1 \oplus \tau_2^{-1}[a_2])\big]\Big] \\
&= \tau_2\Big[\Lambda_1 \oplus s_2\big(\tau_1[v \oplus \tau_2^{-1}[a_2]]\big)\Big] \\
&= \tau_2[\Lambda_1] \oplus \tau_2\big[s_2(\tau_1[v] \oplus a_2)\big] \\
&= \tau_2[\Lambda_1] \oplus \Lambda_2 \oplus s_3\big(\tau_2\big[\tau_1[v]\big]\big) \\
&= \oplus_{\Lambda_3} \circ s_3 \circ \tau_3'(v).
\end{aligned}$$

Thus, are s_1 and s_3 of the same o-type. $\square$

If two USOs are isomorphic, say $\tau' \circ s_1 \circ \oplus_{a_1} = s_2 \circ \oplus_{a_2} \circ \tau'$, then they are of the same o-type, namely $\tau' \circ s_1 \circ \oplus_{a_1 \oplus \tau^{-1}[a_2]} = \oplus_\emptyset \circ s_2 \circ \tau'$.

We now define a revised version of the above procedure. Given a set $U \subseteq$ USO$(d-1)$ of USOs, let `Generate(`U`)` be the process in Table 6.3. For $U =$ USO$(d-1)$, this is essentially the procedure from the beginning

```
  Generate($U$)
1   for all $s_1 \in U$ do
2     for all $s_2 \in \mathrm{USO}(d-1)$ do
3       $s :=$ ($s_1$ $s_2$ [01] product)
4       compute the set $\mathcal{L}$ of $d$-phases of $s$
5       for all $L \subseteq \mathcal{L}$ do
6         $s' :=$ orientation obtained from $s$ by flipping all edges in $L$.
7         for all $a \subseteq [d-1]$, $\Lambda \subseteq [d-1]$ and $\tau \in S_d$ with $\tau(d) = d$ do
8           output $\oplus_\Lambda \circ \tau' \circ s' \circ {\tau'}^{-1} \circ \oplus_a$.
```

Table 6.3: A procedure to generate all USOs which have o-types given by $U \subseteq \mathrm{USO}(d-1)$ in the lower facet.

of this section. For $U = \{s\}$, `Generate(`U`)` will find all USOs which have the same o-type as s on the lower facet.

Lemma 6.8
Let $U \subseteq \mathrm{USO}(d-1)$, such that for each $s \in \mathrm{USO}(d-1)$ there is a $s' \in U$ with the same o-type. Then `Generate(`U`)` *generates all d-dimensional USOs.*

Proof. Let s be a d-dimensional USO, s_1 the induced orientation on the lower d-facet and s_2 the induced orientation on the upper facet. By assumption, there exists $s_1' \in U$, such that s_1 and s_1' are of the same o-type. In particular, there exists $\Lambda \subseteq [d]$, $a \subseteq [d]$ and $\tau_0 \in S_{d-1}$ such that

$$s_1 = \oplus_\Lambda \circ \tau_0' \circ s_1' \circ {\tau_0'}^{-1} \circ \oplus_a.$$

Extend τ_o to $\tau \in S_d$ by setting $\tau(d) = d$ and $\tau(\lambda) = \tau_0(\lambda)$ for $\lambda < d$. Furthermore, set

$$s_2' = {\tau_0'}^{-1} \circ \oplus_\Lambda \circ s_2 \circ \oplus_a \circ \tau_0'.$$

Now consider the set of edges in s pointing towards the upper d-facet,

$$L = \{\{v, v \cup \{d\}\} \mid v \subseteq [d-1], d \in s(v)\}.$$

Since s is a USO, L is closed under phases.

Let $L' = \{\{w, w \cup \{d\}\} \mid \{\tau[w] \oplus a, \tau[w] \oplus a \cup \{d\}\} \in L\}$. We will show that L' is closed under phases with respect to the orientation

$$s'' = (s_1\ s_2\ \texttt{[0 1] product}).$$

In particular, let s' be the orientation obtained by flipping in s'' all edges in L. For $v \in 2^{d-1}$ with $\{v, v \cup \{d\}\} \in L$ the edge $\{\tau^{-1}[v \oplus a], \tau^{-1}[v \oplus a] \cup \{d\}\}$ is in L' and $s'(\tau^{-1}[v \oplus a])$ equals to $s_1'(\tau^{-1}[v \oplus a]) \cup \{d\}$. Hence

$$\begin{aligned} \oplus_\Lambda \circ \tau' \circ s' \circ {\tau'}^{-1} \circ \oplus_a(v) &= \Lambda \oplus \tau[s'(\tau^{-1}[v \oplus a])] \\ &= \Lambda \oplus \tau_0[s_1(\tau_0^{-1}[v \oplus a])] \cup \{d\} \\ &= s_1(v) \cup \{d\} = s(v). \end{aligned}$$

Virtually the same argument works for $v \cup \{d\}$, as well as for the other cases $v \in 2^{d-1}$ with $\{v, v \cup \{d\}\} \notin L$.

Therefore, $s = \oplus_\Lambda \circ \tau' \circ s' \circ {\tau'}^{-1} \circ \oplus_a$, which is produced by the call `Generate(`U`)`.

It remains to show that L' is closed under phases. Let $\{w, w \cup \{d\}\} \in L'$ and $\{w', w' \cup \{d\}\}$ be in phase with $\{w, w \cup \{d\}\}$ with respect to s'. We have to show that $\{w', w' \cup \{d\}\} \in L'$. By definition of s', the two edges are in phase if and only if either $s_1'(w) = s_2'(w')$ or $s_2'(w) = s_1'(w')$ holds. For $s_1'(w) = s_2'(w')$, we get

$$\tau^{-1}[\Lambda \oplus s_1(\tau[w] \oplus a)] = \tau^{-1}[\Lambda \oplus s_2(\tau[w'] \oplus a)],$$

which can only be the case whence $s_1(\tau[w] \oplus a) = s_2(\tau[w'] \oplus a)$. Thus, the d-edges in $\tau[w] \oplus a$ and $\tau[w'] \oplus a$ are in phase and have to be oriented towards the same facet. Since $\{\tau[w] \oplus a, \tau[w] \oplus a \cup \{d\} \in L$ the d-edge in $\tau[w'] \oplus a$ is in L. Hence, $\{w', w' \cup \{d\}\} \in L'$.

The remaining case $s_2'(w) = s_1'(w')$ is handled analogously. □

It will be convenient to fix some notation for the following discussion. For two d-dimensional USOs s and s' denote by $\operatorname{ext}(s, s')$ the number of d-phases of the orientation $\bar{s} = (s\ s'\ [01]\ \texttt{product})$, i.e., $\operatorname{ext}(s, s') = \operatorname{ph}_d(\bar{s})$. Let

$$\operatorname{ext}(s) = \sum_{s' \in \mathrm{USO}(d)} 2^{\operatorname{ext}(s,s')}.$$

Furthermore, denote by $\operatorname{iso}(s)$ the number of USOs isomorphic to s and with $\operatorname{ocl}(s)$ the number of USOs of the same o-type as s.

Lemma 6.9
Let U be a set of representatives for the o-types in $\mathrm{USO}(d-1)$*. Then the number of d-dimensional USOs is*

$$|\,\mathrm{USO}(d)| = \sum_{s\in U} \mathrm{ocl}(s)\cdot \mathrm{ext}(s).$$

Proof. For $s, s' \in \mathrm{USO}(d-1)$ denote by $O(s,s')$ the set of all d-dimensional USOs inducing s on the lower d-facet and s' on the upper d-facet. Furthermore, set

$$\begin{aligned} O(s) &= \bigcup \{O(s,s') \mid s' \in \mathrm{USO}(d-1)\} \\ \bar{O}(s) &= \bigcup \{O(s') \mid s' \text{ same o-class as } s\}. \end{aligned}$$

Observe, that $\{O(s,s') \mid s,s' \in \mathrm{USO}(d-1)\}$ forms a partition of the set $\mathrm{USO}(d)$. For the cardinalities of the above sets we calculate

$$\begin{aligned} |O(s,s')| &= 2^{\mathrm{ext}(s,s')} \\ |O(s)| &= \mathrm{ext}(s) \\ |\bar{O}(s)| &= \mathrm{ocl}(s)\cdot \mathrm{ext}(s). \end{aligned}$$

Finally, $\mathrm{USO}(d) = \bigcup_{s\in U} \bar{O}(s)$ and hence

$$|\,\mathrm{USO}(d)| = \sum_{s\in U} \mathrm{ocl}(s)\cdot \mathrm{ext}(s).$$

□

6.3 Dimension 3

Remember that there are two isomorphy-types of 2-dimensional USOs: The eye $s_e = (0$ `eye`$)$ and the bow $s_b = (0\ .1$ `bow`$)$. (See Figure 2.5.) Since a reorientation of an eye is an eye and the reorientation of a bow is a bow, the isomorpy-classes and the o-types coincide. There are eight bows and four eyes, i.e., $\mathrm{ocl}(s_b) = 8$ and $\mathrm{ocl}(s_e) = 4$.

We apply `Generate(`$\{s_e, s_b\}$`)` to generate all 3-dimensional USOs. Figure 6.3 depicts all 3-phases for the case that s_e is in the lower 3-facet. There is one orientation with four phases, four orientations have

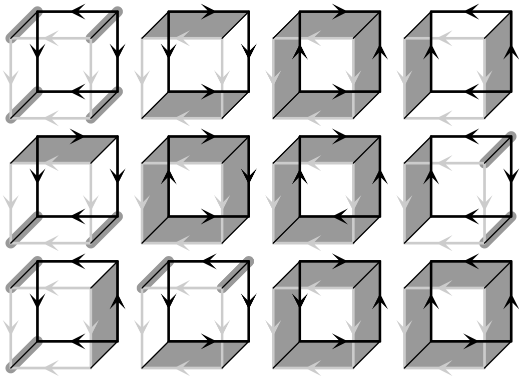


Figure 6.3: `Generate( 0 eye )`: All 3-dimensional USOs with an eye in the lower facet (schematic). For each choice for the upper facet, the resulting phases are drawn. Edges connected by a gray area are in 3-phase.

three phases, two have two phases, and five orientations allow only one phase. Thus

$$\mathrm{ext}(s_e) = 2^4 + 4 \cdot 2^3 + 2 \cdot 2^2 + 5 \cdot 2^1 = 66.$$

Figure 6.4 is the corresponding picture for s_b. Counting in this case yields

$$\mathrm{ext}(s_b) = 2^4 + 2 \cdot 2^3 + 5 \cdot 2^2 + 4 \cdot 2^1 = 60.$$

Altogether we get $4 \cdot 66 + 8 \cdot 60 = 744$ USOs of dimension 3.

Among the 744 USOs on $\mathfrak{C}^3$ $744/8 = 93$ have their sink in $\emptyset$. By Lemma 6.1, we find all possible isomorphy types among these 93 orientations. As a first step we group the orientations according to their iso-sets. There are 17 different iso-sets. Only in two cases do the iso-sets not restrict the number of possible isomorphisms. Five of the iso-sets allow only two permutations and the remaining ten iso-sets already fix the permutation. That is, for these ten iso-sets the corresponding clusters of USOs with the same iso-set are already isomorphism classes. Of the remaining 7 clusters only two are not isomorphism classes. The two clusters with non-restricting iso-sets *are* isomorphism classes.

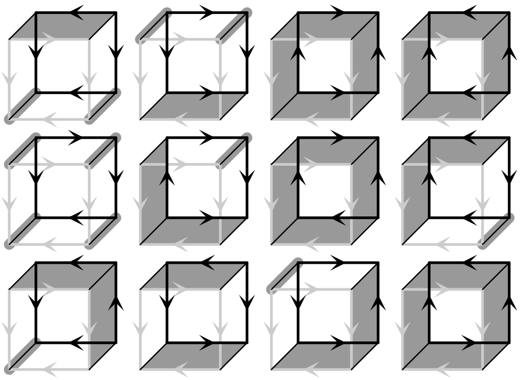


Figure 6.4: `Generate( 0 .1 bow )`: All 3-dimensional USOs with a bow in the lower facet (schematic). For each choice for the upper facet, the resulting phases are drawn. Edges connected by a gray area are in 3-phase.

The result are 19 isomorphy classes. As it turns out, the orientations in Table 6.1 form a system of representatives of these 19 isomorpism classes. If we further restrict to o-types, only ten o-types remain. See Table 6.4. For the distribution of all USOs over the isomorphism classes and the o-types consult Table 6.5.

6.4 Dimension 4

Based on the data in the last section, we can use `Generate` to generate all 4-dimensional USOs. Since we have to check each of the ten o-types in Table 6.4 against *all* 744 USOs in USO(3), an enumeration as in Figures 6.3 and 6.4 seems to be inadequate. In the following we just list the resulting numbers. A list of all 4-dimensional USOs can be found at [38].

See Table 6.6 for the number of 4-dimensional USOs. To count the number of non-isomorphic 4-dimensional USOs we first partition USO(4) according to the iso-sets. There are 11525 different iso-sets. For 9337 iso-sets the corresponding classes are already isomorphism classes.

o-type	orientations
1	`1 sw`, `6 sw`, `10 sw`, `12 sw`
2	`2 sw`
3	`3 sw`, `8 sw`, `14 sw`, `18 sw`
4	`4 sw`, `19 sw`
5	`5 sw`
6	`7 sw`, `13 sw`, `16 sw`
7	`9 sw`
8	`11 sw`
9	`15 sw`
10	`17 sw`

Table 6.4: The o-types in USO(3).

`sw`	1	2	3	4	5	6	7	8	9	10
iso. USOs	48	24	48	48	48	48	24	48	48	48
same o-type	196	24	196	64	48		96		48	

`sw`	11	12	13	14	15	16	17	18	19	
iso. USOs	48	48	24	48	8	48	24	48	16	
same o-type	48				8		24			

Table 6.5: The distribution of USO(3) over the isomorphism classes and the o-types.

o-type	s	ocl(s)	ext(s)	|`Generate`(s)|
1	`1 sw`	192	7296	1400832
2	`2 sw`	24	7572	181728
3	`3 sw`	192	7322	1405824
4	`4 sw`	64	5918	378752
5	`5 sw`	48	6592	316416
6	`7 sw`	96	8868	851328
7	`9 sw`	48	8716	418368
8	`11 sw`	48	6592	316416
9	`15 sw`	8	10586	84688
10	`17 sw`	24	7808	187392
Total:				5541744

Table 6.6: The number of 4-dimensional USOs.

The remaining 2188 iso-set classes split into 3089 isomorphism classes. Overall, there are 14614 isomorphism classes in dimension 4.

6.5 Dimension 5 and higher

From dimension 5 on, the number of USOs is so large that an explicit list of all USOs is not desirable. But even an implicit list as provided by `Generate` is too large to be practical. We therefore only give estimates for the number of USOs in dimension 5 and higher.

Lemma 6.10

$$|\,\mathrm{USO}(d)| \geq 4 \cdot |\,\mathrm{USO}(d-1)|^2.$$

PROOF. For each pair s_1 and s_2 of $(d-1)$-dimensional USOs we get two combed d-dimensional USOs

```
(s1 s2 [ 0 1 ] product)
(s1 s2 [ 1 0 ] product)
```

This construction provides us with exactly $2 \cdot |\operatorname{USO}(d-1)|^2$ combed d-dimensional USOs. In fact, we obtain all combed USOs in this way.

Now choose $s_1 \in \operatorname{USO}(d-1)$ and a vertex $v_0 \in \operatorname{V}(\mathfrak{C}^{d-1})$. Consider the sets

$$X_t := \{s \in \operatorname{USO}(d-1) \mid s(v_0) = t\}$$

for $t \in 2^{d-1}$. The map $s \mapsto \oplus_t \circ s$ establishes a bijection between X_t and $X_\emptyset$. In particular, each set X_t has cardinality $|\operatorname{USO}(d-1)|/2^{d-1}$. For each $s_2 \in X_{s_1(v_0)}$ the constructions

$$s_1\ s_2\ \texttt{[0 1] product}\ \{\ v_0\ .d\ \}\ \texttt{flip}$$

and

$$s_1\ s_2\ \texttt{[1 2] product}\ \{\ v_0\ .d\ \}\ \texttt{flip}$$

yield non-combed USOs. (The edge $\{v_0, v_0 \oplus \{\lambda\}\}$ is flippable.)

Repeating of all possible s_1, v_0, and s_2 we have

$$2 \cdot |\operatorname{USO}(d-1)| \cdot 2^{d-1} \cdot \frac{|\operatorname{USO}(d-1)|}{2^{d-1}} = 2|\operatorname{USO}(d-1)|^2$$

different non-combed USOs. Together with the $2|\operatorname{USO}(d-1)|^2$ combed USOs we obtain the promised $4 \cdot |\operatorname{USO}(d-1)|^2$ orientations. $\square$

A similar approach can be used for an upper bound. Let s be a d-dimensional USO. The orientations in the lower and the upper d-facet are $(d-1)$-dimensional USOs. Obviously there are at most $2^{2^{d-1}}$ choices for the orientations of the d-edges. This gives an upper bound of $|\operatorname{USO}(d)| \leq 2^{2^{d-1}} |\operatorname{USO}(d-1)|^2$. Since there are orientations for which each d-edge is flippable, this bound cannot be improved. But if we subdivide into $(d-2)$-dimensional subcubes, we obtain a better bound.

Lemma 6.11
Let U be a system of representatives for the o-types of $\operatorname{USO}(d-2)$. *Then*

$$|\operatorname{USO}(d)| \leq 3 \cdot 2^{d-2} \cdot |\operatorname{USO}(d-2)|^2 \cdot \sum_{s \in U} \operatorname{ocl}(s) \cdot \operatorname{ext}(s)^2.$$

Proof. Let s be a d-dimensional USO. Consider the following orientations on subcubes of $\mathfrak{C}^d$:

- s_0 is the orientation on $\mathfrak{C}^{d-2}$.
- s_1 is the orientation on the lower $(d-1)$-facet.
- s_2 is the orientation on the lower d-facet.
- s_3 is the orientation on the 3-dimensional cube antipodal to $\mathfrak{C}^{d-2}$.
- For $v \in \mathfrak{C}^{d-2}$ let s_v be the orientation on $\mathfrak{C}^{[v, v\oplus\{d-1,d\}]}$.

There are at most $|\operatorname{USO}(d-2)|$ choices for s_0 and s_3. Depending on the choice of s_0 we have $\operatorname{ext}(s_0)$ possible choices for s_1 and s_3. After the choice of s_1 and s_3 we know two edges in each s_v, thus we have at most 3 possible choices for each s_v. Altogether we get

$$\begin{aligned} |\operatorname{USO}(d)| &\leq |\operatorname{USO}(d-2)|^2 \cdot \left(\sum_{s_0 \in \operatorname{USO}(d-2)} \operatorname{ext}(s_0)^2 \right) \cdot 3 \cdot 2^{d-2} \\ &= 3 \cdot 2^{d-2} \cdot |\operatorname{USO}(d-2)|^2 \cdot \sum_{s \in U} \operatorname{ocl}(s) \cdot \operatorname{ext}(s)^2. \end{aligned}$$

□

Corollary 6.12
The number of 5*-dimensional unique sink orientations is between* $1.23 \cdot 10^{14}$ *and* $7.47 \cdot 10^{14}$.

Proof. The number of 4-dimensional USOs is 5541744, thus

$$|\operatorname{USO}(5)| \geq 4 \cdot 5541744^2 = 122843706246144 = 1.23 \cdot 10^{14}$$

by Lemma 6.10. For the lower bound consult Table 6.5.

$$\begin{aligned} |\operatorname{USO}(5)| &\leq 3 \cdot 8 \cdot 744^2 \cdot \sum_{s \in U} \operatorname{ocl}(s) \cdot \operatorname{ext}(s)^2 \\ &= 17856 \cdot (192 \cdot 7296^2 + 24 \cdot 7572^2 + 192 \cdot 7322^2 + \\ &\quad + 64 \cdot 5918^2 + 48 \cdot 6592^2 + 96 \cdot 8868^2 + \end{aligned}$$

$$\begin{aligned} & + 48 \cdot 8716^2 + 48 \cdot 6592^2 + 8 \cdot 10586^2 + \\ & + 24 \cdot 7808^2) = 17856 \cdot 41858776992 \\ = \quad & 747430321969152 = 7.4743 \cdot 10^{14}. \end{aligned}$$

□

Two different experiments indicate that the true value is closer to the upper bound than to the lower bound. The first experiment is based on an implementation of the Markov-chain introduced in Section 4.3. Let p be the probability that a 5-dimensional USO is combed in direction 1. As already observed in the proof of Lemma 6.10, the number of combed orientations in USO(5) is

$$n_c = 2 \cdot |\,\mathrm{USO}(4)|^2 = 61421853123072.$$

Let $n = |\,\mathrm{USO}(5)|$. Then $n_c = n \cdot p$. If we run the Markov-chain defined by (4.6) $\tilde{n}$ times and count the number $\tilde{n}_c$ of orientations combed in direction 1, then we get

$$\frac{\tilde{n}_c}{\tilde{n}} \to p$$

since the Markov-chain converges to the uniform distribution. Thus,

$$n_c \cdot \frac{\tilde{n}}{\tilde{n}_c} \to n.$$

The Markov-chain was implemented in `C++`. Starting with the uniform 5-dimensional orientation we receptively choose a label uniformly at random from $\{1, \ldots, 5\}$, calculate the phases along this label and flip each phase with probability 1/2. An implementation can be found at [38]. In an experiment with 10^9 iterations of the Markov-chain, where we counted every 1000-st orientation, we observed $\tilde{n}_c = 9554$ orientations combed in direction 1. This gives an estimate of

$$n_c \cdot \frac{\tilde{n}}{\tilde{n}_c} = 661421853123072 \cdot \frac{100000}{9554} = 6.4289 \cdot 10^{14}. \qquad (6.2)$$

Although the numbers seem to converge, and appear reasonable as compared to our bounds, we want to point out that we have, in actual fact, no guarantee that the Markov-chain is already converging.

The second experiment is using Lemma 6.9. Applied to the case of dimension five, we have 1291 o-classes in dimension four. For each of them we have to calculate the number of phases combined with all 5541744 USOs of dimension four. Even though this is not a trivial task, it is something a computer program can do. An implementation of this idea gave (after thee weeks calculation time) the following number: 638560878292512. Although we do not have an actual proof that this is the right result (the implementation could be wrong), it is quite likely that this is the actual number of five-dimensional USOs.

For higher dimensions, the upper bound from Lemma 6.11 fails, since we do not have access to $\mathrm{ext}(s)$. The lower bound from Lemma 6.10 yields:

$$\begin{aligned} |\,\mathrm{USO}(6)| &\geq 4 \cdot |\,\mathrm{USO}(5)|^2 \geq 64 \cdot |\,\mathrm{USO}(4)|^4 = 6.04 \cdot 10^{28} \\ |\,\mathrm{USO}(7)| &\geq 4 \cdot |\,\mathrm{USO}(6)|^2 \geq 2^{14} \cdot |\,\mathrm{USO}(4)|^8 = 1.46 \cdot 10^{58} \\ |\,\mathrm{USO}(8)| &\geq 4 \cdot |\,\mathrm{USO}(7)|^2 \geq 2^{30} \cdot |\,\mathrm{USO}(4)|^{16} = 8.50 \cdot 10^{116} \end{aligned}$$

Since the number of atoms in the universe is estimated to be between 10^{78} and 10^{79}. This means that from dimension 8 on, any enumeration of $\mathrm{USO}(d)$ will necessarily fail.

Bibliography

[1] L. Adler and R. Saigal. Long monotone paths in abstract polytopes. *Math. Oper. Res.*, 1:89–95, 1976.

[2] N. Alon, J. Bruck, J. Naor, M. Naor, and R. Roth. Construction of asymptotically good, low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38:509–516, 1992.

[3] N. Amenta and G. M. Ziegler. Deformed products and maximal shadows of polytopes. *Contemporary Mathematics*, 223:57–90, 1999.

[4] M. Balmer and T. Szabó. Deterministic algorithms on unique sink orientations of small dimension. Manuscript in preparation, 2003.

[5] E. Behrends. *Introduction to Markov Chains.* Vieweg, Braunschweig, 2002.

[6] R. W. Cottle, J.-S. Pang, and R. E. Stone. *The Linear Complementary Problem.* Academic Press, 1992.

[7] G. Dantzig. Linear programming: The story about how it began. *Operations Research*, 50(1):42–47, 2002.

[8] M. Develin. LP-orientations of cubes and crosspolytopes. *Advances in Geometry*, 2002. To appear.

[9] B. Gärtner. A subexponential algorithm for abstract optimization problems. *SIAM Journal on Computing*, 24:1018–1035, 1995.

[10] B. Gärtner. Combinatorial linear programming: Geometry can help. In *Proc. 2nd Int. Workshop on Randomization and Approximation Techniques in Computer Science*, volume 1518, pages 82–96, 1998. Lecture Notes in Computer Science.

[11] B. Gärtner. Combinatorial structure in convex programs. Manuscript, 2001.

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

[13] B. Gärtner. Strong LP-type problems. Manuscript, 2003.

[14] B. Gärtner, M. Henk, and G. M. Ziegler. Randomized simplex algorithms on Klee-Minty cubes. *Combinatorica*, 18(3):349–372, 1998.

[15] B. Gärtner and E. Welzl. Linear programming – randomization and abstract frameworks. In *Proc. 13th Annual Symp. on Theoretical Aspects of Computer Science (STACS)*, pages 669–687. Lecture Notes in Computer Science 1064, Springer, 1996.

[16] F. Granot and P. L. Hammer. On the use of boolean functions in 0-1 programming. *Methods of Operations Research*, 34:154–184, 1971.

[17] P. L. Hammer, B. Simeone, T. Liebling, and D. de Werra. From linear separability to unimodality: a hierarchy of pseudo–boolean functions. *SIAM J. Discrete Math.*, 1:174–184, 1988.

[18] K. W. Hoke. Completely unimodular numberings of a simple polytope. *Discrete Applied Mathematics*, 20:69–81, 1988.

[19] F. Holt and V. Klee. A proof of the strict monotone 4-step conecture. In J. Chazelle, J. B. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, Contemporary Mathematics, pages 201–216. American Mathematical Society, 1998.

[20] V. Kaibel. Bottom-top graphs. Manuscript, http://www.ti.inf.ethz.ch/ew/workshops/01-lc/kaibel.html, 2001.

[21] G. Kalai. A subexponential randomized simplex algorithm. In *Proc. 24th ACM Symposium on Theory of Computing*, pages 475–482, 1992.

[22] O. H. Keller. Über die lückenlose Erfüllung des Raumes mit Würfeln. *J. Reine Angew. Math.*, 163:231–248, 1930.

[23] L. G. Khachiyan. Polynomial algorithms in linear programming. *U.S.S.R Comput. Math. and Math. Phys.*, 20:53–72, 1980.

[24] V. Klee and G. J. Minty. How good is the simplex algorithm. In O. Shisha, editor, *Inequalities - III.* Academic Press Inc., New York and London, 1972.

[25] J. C. Lagarias and P. W. Shor. Keller's cube-tiling conjecture is false in high dimensions. *Bulletin of the American Mathematical Society*, 27(2):279–283, 1992.

[26] J. Mackey. A cube tiling of dimension eight with no facetsharing. *Discrete & Computational Geometry*, 28:275–279, 2002.

[27] J. Matoušek. The number of unique-sink orientations of the hypercube. *Combinatorica*, 2002. To appear.

[28] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498–516, 1996.

[29] J. Matoušek and T. Szabó. Random edge can be exponential on abstract cubes. To appear in FOCS, 2004.

[30] J. Matoušek and P. Valtr. personal communication, 2001.

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

[32] W. D. Morris. Distinguishing cube orientations arising from linear programs. Submitted, 2002.

[33] W. D. Morris. Randomized principal pivot algorithms for P-matrix linear complementarity problems. *Mathematical Programming*, Ser. A(92):285–296, 2002.

[34] C. H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[35] A. L. Peressini, F. E. Sullivan, and J. J. Uhl. *The Mathematics of Nonlinear Programming.* Undergraduate Texts in Mathematics. Springer, 1988.

[36] O. Perron. Über die lückenlose Ausfüllung des n-dimensionalen Raumes durch kongruente Würfel I & II. *Math. Z.*, 47:1–26, 161–180, 1940.

[37] H. Samelson, R. M. Thrall, and O. Wesler. A partition theorem for Euclidean n-space. *Proceedings of the American Mathematical Society*, 9:805–807.

[38] I. Schurr. The unique sink orientations in dimension 3 and 4. http://www.inf.ethz.ch/personal/schurr/uso.

[39] I. Schurr and T. Szabó. Finding the sink takes some time. *Discrete Comput Geom*, 31:627–642, 2004.

[40] A. Stickney and L. Watson. Digraph models of bard-type algorithms for the linear complementary problem. *Mathematics of Operations Research*, 3:322–333, 1978.

[41] S. Szabó. Cube tilings as contributions of algebra to geometry. *Beiträge zur Algebra und Geometrie*, 34(1):63–75, 1993.

[42] T. Szabó and E. Welzl. Unique sink orientations of cubes. In *Proc. 42^{nd} IEEE Symp. on Foundations of Comput. Sci.*, pages 547–555, 2000.

[43] N. Zadeh. What is the worst case behavior of the simplex algorithm? Technical Report 27, Dept. Operation Research, Stanford, 1980.

[44] G. M. Ziegler. *Lectures on Polytopes.* Number 152 in Graduate texts in mathematics. Springer, 1995.

Curriculum Vitae

Ingo Andreas Schurr

born on March 12, 1972 in Baden–Baden, Germany.

1985–1992	**high school in Baden–Baden, Germany** Degree: Abitur
1993–2000	**studies at Freie Universität Berlin, Germany** Major: Mathematics Minor: Computer Science Degree: Diplom
2000–2004	**Ph.D. student at ETH Zürich, Switzerland**