Diplomarbeit

Randomized Pivot Algorithms in Linear Programming

Lukas Finschi

Eidgenössische Technische Hochschule Zürich

Institut für Operations Research
Clausiusstrasse 45
CH-8092 Zürich

February 28, 1997

Meinen Eltern

in Dankbarkeit gewidmet

To my Parents

in Grateful Dedication

Contents

Abstract

The study of algorithms in linear programming has produced in the last years remarkable developments; especially randomization has led to new bounds for the (expected) computation time. An algorithm of Matoušek, Sharir, and Welzl (as well as similar algorithms, e.g. of Kalai) established a subexponential upper bound for the expected run time of a class of abstract optimization problems (we will call them *optimization problems of LP-type*): These problems are specified by a pair (H, v), where H is a set (e.g. a set of linear constraints in $\mathbb{R}^d$) and $v : \mathcal{P}(H) \to \mathcal{L}$ is an abstract value function mapping the subsets of H to a linearly ordered set $(\mathcal{L}, \leq)$, such that the following two axioms hold:

- *Monotonicity:* $F \subseteq G \subseteq H \Rightarrow v(F) \geq v(G)$
- *Locality:* $F \subseteq G \subseteq H$ and $v(F) = v(G) > v(G \cup \{h\}) \Rightarrow v(F) > v(F \cup \{h\})$

The first chapter of this diploma thesis contains a strict discussion of this axiomatic framework and the analysis of the mentioned algorithm of Matoušek, Sharir, Welzl; in addition we discuss in the same way a sampling technique of Clarkson which improves the subexponential bound further.

It is well known that in general a linear programming problem does not hold the axioms of the LP-type framework: The locality condition will usually fail when the solution of the linear program is not unique. When the feasible region of the LP is bounded, then this problem can be solved by maximizing $(c^T x, x_1, \ldots, x_d)$ instead of $c^T x$; in the unbounded case this does not help. The common way out is the use of a numerical bounding box; in this thesis we develop a technique with a combinatorial bounding box (see chapter 2) where the size of the box is represented by a parameter which is considered to be large (but we never set this parameter to a concrete value): We will not have no compute a concrete numerical bound such that the box is large enough for a given LP, also there are no huge numbers which come into computation; our technique is purely combinatorial and enables to design combinatorial algorithms.

The studies in chapter 1 and 2 are necessary for the design and analysis of the pivot algorithms in the third chapter; especially we discuss again the algorithm of Matoušek, Sharir, Welzl. An implementation of this algorithm in the C programming language demonstrates the concrete application of our techniques in practice.

We conclude this thesis by reporting some experiences from the experiments with our implementation and try to give some suggestions for the further research.

Chapter 1

Axiomatic Framework

1.1 Notations for Linear Programming

In this section we introduce several notations and settings for linear programming problems.

With the use of matrix notation we can write any linear programming problem in the form

$$\max c^T x, \text{ such that } Ax \leq b \tag{1.1}$$

where $c \in \mathbb{R}^d, b \in \mathbb{R}^n, A \in \mathbb{R}^{n \times d}$ are given and $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ is the vector of the unknowns. The inequality $Ax \leq b$ is satisfied when it holds for every coordinate, i.e. when $A_i x \leq b_i$ for all $i \in \{1, \ldots, n\}$, where A_i is the ith row of A; the superscript ^{T} denotes the transposed of a matrix or vector.

In order to have simple notations for the sampling algorithms which consider in general relaxed linear programming problems (like (1.3)) instead of the original problems, we define the set

$$H \equiv \{h_1, \ldots, h_n\} := \{A_1 x \leq b_1, \ldots, A_n x \leq b_n\}$$

and write (1.1) in the form

$$\max c^T x, \text{ such that for all } h \in H : x \text{ does not violate } h \tag{1.2}$$

where x violates $h_i \in H$ if and only if $A_i x > b_i$. For a given problem (LP) of the form (1.2) we denote for any $G \subseteq H$ the relaxed problem

$$\max c^T x, \text{ such that for all } h \in G : x \text{ does not violate } h \tag{1.3}$$

by $(LP)\mid_G$.

In a third setting (see section 1.2) we consider in an abstract framework socalled *optimization problems of LP-type*; problems of the form (1.3) are then written as follows:

$$\text{find a minimal subset } B \subseteq G, \text{ such that } v(B) = v(G);$$

v is an (abstract) value function which has to satisfy the axioms of *monotonicity* and *locality*. In this third setting we will discuss the randomized algorithm of Matoušek, Sharir, and Welzl which leads to a subexponential bound for linear programming (which can be improved by a sampling technique of Clarkson; see section 1.4); for a comparison of several randomized algorithms for linear programming see e.g. [5]. In chapter 2 we will discuss how to transform linear programming problems to optimization problems of LP-type (and, by the way, we present the algorithm of Seidel).

Chapter 3 will bring another, fourth setting of linear programs, namely *dictionaries* for pivot algorithms, and discusses the pivot forms of the algorithms of Seidel and of Matoušek, Sharir, Welzl, where the analysis of the abstract framework will be used.

1.2 Optimization Problems of LP-Type

The discussion of sampling algorithms in this chapter follows an article of Emo Welzl [10] and the chapter about linear programming in [8]. The abstract framework goes back to [6].

We start by defining the abstract framework:

Definition 1.1 (Optimization Problem of LP-Type) *Assume there are given a finite set $H \neq \emptyset$ and a function $v : \mathcal{P}(H) \to \mathcal{L}$, where $\mathcal{P}(H)$ denotes the set of all subsets of H and $(\mathcal{L}, \leq)$ is a linearly ordered set.*
The elements of H are called constraints, *and for any $G \subseteq H$ we call $v(G)$* the value of G.
A subset of constraints $B \subseteq H$ is called a basis *if for all $\tilde{B} \subsetneq B : v(\tilde{B}) > v(B)$.*
For any $G \subseteq H$: A set of constraints $B \subseteq H$ is called an optimal basis of G *if B is a basis with $B \subseteq G$ and $v(B) = v(G)$.*
Consider the following two properties (which will be used as axioms):

- **Monotonicity:** *For all F, G with $F \subseteq G \subseteq H : v(F) \geq v(G)$*
- **Locality:** *For all F, G with $F \subseteq G \subseteq H$ and $v(F) = v(G)$ and for any $h \in H$:*
  $v(G) > v(G \cup \{h\}) \Rightarrow v(F) > v(F \cup \{h\})$
  (Remark that monotonicity implies the converse:
  $v(F) > v(F \cup \{h\}) \Rightarrow v(G) = v(F) > v(F \cup \{h\}) \geq v(G \cup \{h\})$.)

Then: If for the pair (H, v) the axioms of monotonicity and locality hold, we say that (H, v) specifies an optimization problem of LP-type, *namely "find an optimal basis $B \subseteq H$ of H".*

Remark 1.2 *If (H, v) specifies an optimization problem of LP-type, then also (G, v) for any $G \subseteq H$ (where the value function $v : \mathcal{P}(G) \to \mathcal{L}$ is defined by restriction on $\mathcal{P}(G)$).*

For completeness we give the following theorem which ensures that an optimization problem of LP-type has always a solution (which has not to be unique):

Theorem 1.3 (Existence of an Optimal Basis of $G \subseteq H$) *Given an optimization problem of LP-type specified by (H, v) and any $G \subseteq H$: Then there exists an optimal basis B of G.*

Proof: Set $B_1 := G$, then $v(B_1) = v(G)$.
If B_1 is not a basis, then there exists $B_2 \subsetneq B_1$ with $v(B_2) = v(B_1) = v(G)$ (in the first equality we use the definition of a basis and monotonicity).
If B_2 is not a basis, then there exists $B_3 \subsetneq B_2$ with $v(B_3) = v(B_2) = v(G)$.
With induction we have for a general $k \in \{1, 2, \dots\}$: If B_k is not a basis, then there exists $B_{k+1} \subsetneq B_k$ with $v(B_{k+1}) = v(B_k) = v(G)$.
Since $0 \leq |B_{k+1}| < |B_k|$, this process stops for a $\tilde{k} \leq |G|$ with $B_{\tilde{k}} \subseteq G$ a basis with $v(B_{\tilde{k}}) = v(G)$, i.e. $B_{\tilde{k}}$ is an optimal basis of G. ∎

The following definition introduces (next to definition 1.10) the key notions for the discussion of the algorithms within the framework of LP-type problems:

Definition 1.4 (Violated and Extreme Constraint)
Given an optimization problem of LP-type specified by (H, v) and any $G \subseteq H$ and any $h \in H$.
h is called violated by G *if $v(G) > v(G \cup \{h\})$ (then $h \notin G$).*
h is called extreme in G *if $v(G \setminus \{h\}) > v(G)$ (then $h \in G$).*

Remark 1.5 *h extreme in $G \Leftrightarrow$ ($h \in G$ and h violated by $G \setminus \{h\}$).*

Remark 1.6 *We can formulate the locality condition of an LP-type problem in the following way, using contraposition: For all F, G with $F \subseteq G \subseteq H$ and $v(F) = v(G)$ and for any $h \in H$: h not violated by $F \Rightarrow h$ not violated by G.*

Notation: For any set M we denote the cardinality of M with $|M|$.

Definition 1.7 (Combinatorial Dimension $\dim(H, v)$ of (H, v))
Given an optimization problem of LP-type specified by (H, v). The maximum cardinality of any basis $B \subseteq H$ is called the combinatorial dimension of (H, v) and is denoted by $\dim(H, v)$ (remark that we do not only consider every optimal basis of H but any basis):

$$\dim(H, v) := \max_{\substack{B \subseteq H \\ B \text{ a basis}}} |B|.$$

The combinatorial dimension will take the place of the usual dimension d of linear programs when we study the complexity of the algorithms within this framework. The fundamental lemma for this is the following (especially point (ii)):

Lemma 1.8 *Given an optimization problem of LP-type specified by (H, v) and any $G \subseteq H$. Let be B any optimal basis of G (B exists, see theorem 1.3). Then:*

(i) For all $h \in H$: h extreme in $G \Rightarrow h \in B$ (the converse is not true in general).

(ii) G has at most $\dim(H, v)$ extreme constraints.

(iii) For all $h \in H$: h violated by $G \Leftrightarrow h$ violated by B.

(iv) $v(G) = v(G \cup \{h \in H | h \text{ not violated by } G\})$.

Proof:

(i) Given $h \in H$ extreme in G, i.e. $v(G \setminus \{h\}) > v(G)$. Assume $h \notin B$. Then $B \setminus \{h\} = B$, so $v(G) = v(B) = v(B \setminus \{h\}) \geq v(G \setminus \{h\})$, a contradiction.

(ii) B is a basis, so $|B| \leq \dim(H, v)$. Together with (i) follows (ii).

(iii) *($\Rightarrow$)* Given $h \in H$ with $v(G) > v(G \cup \{h\})$. Since $B \subseteq G \subseteq H$ and $v(B) = v(G)$, the locality implies $v(B) > v(B \cup \{h\})$.
($\Leftarrow$) Given $h \in H$ with $v(B) > v(B \cup \{h\})$. Since $B \cup \{h\} \subseteq G \cup \{h\}$, monotonicity implies $v(G) = v(B) > v(B \cup \{h\}) \geq v(G \cup \{h\})$.

(iv) Since H is finite, $\{h \in H | h \text{ not violated by } G\} = \{h_1, \ldots, h_k\}$ (if not $k \geq 1$, then (iv) is trivial). For $\ell = 1, \ldots, k-1$ we have (see remark 1.6: here $G \subseteq G \cup \{h_1, \ldots, h_\ell\} \subseteq H$ instead of $F \subseteq G \subseteq H$)

$$\begin{gathered} v(G) = v(G \cup \{h_1, \ldots, h_\ell\}) \text{ and } v(G) = v(G \cup \{h_{\ell+1}\}) \\ \Rightarrow \\ v(G) = v(G \cup \{h_1, \ldots, h_\ell\}) = v(G \cup \{h_1, \ldots, h_{\ell+1}\}), \end{gathered}$$

and by induction follows (iv).

■

Under the assumption of monotonicity, we can prove the equivalence between *locality* and *lemma 1.8 (iii)* "$\Rightarrow$" and *lemma 1.8 (iv)*:

Remark 1.9 *Given (H, v) as in definition 1.1 but without the assumption of locality (i.e. with monotonicity only). Furthermore there are given F, G with $F \subseteq G \subseteq H$ and $v(F) = v(G)$ and a restriction $h \in H$ with $v(G) > v(G \cup \{h\})$.*

(i) If for every optimal basis B of G the constraint h is violated by B, then $v(F) > v(F \cup \{h\})$.

(ii) If $v(F) = v(F \cup \{\tilde{h} \in H | \tilde{h}$ not violated by $F\})$, then $v(F) > v(F \cup \{h\})$.

Proof:

(i) Monotonicity implies (as in theorem 1.3) the existence of an optimal basis $B \subseteq F$ of F. Because of $v(F) = v(G)$ and $F \subseteq G$, B is an optimal basis of G. Since we then assume that h is violated by B, we conclude with monotonicity for $B \cup \{h\} \subseteq F \cup \{h\}$

$$v(F) = v(B) > v(B \cup \{h\}) \geq v(F \cup \{h\}).$$

(ii) Assume $v(F) \leq v(F \cup \{h\})$, i.e. (because of monotonicity) $v(F) = v(F \cup \{h\})$. For all $g \in G$ we know with monotonicity $v(F) \geq v(F \cup \{g\}) \geq v(G) = v(F)$, so g is not violated by F; with the assumption for h follows $G \cup \{h\} \subseteq \{\tilde{h} \in H | \tilde{h}$ not violated by $F\}$. Monotonicity and $v(F) = v(F \cup \{\tilde{h} \in H | \tilde{h}$ not violated by $F\})$ imply

$$v(F) = v(F \cup \{\tilde{h} \in H | \tilde{h} \text{ not violated by } F\}) \leq v(G \cup \{h\}) \leq v(G) = v(F),$$

so $v(G) = v(G \cup \{h\})$, a contradiction.

■

1.3 The Algorithm of Matoušek, Sharir, Welzl

This section presents the algorithm of Matoušek, Sharir, and Welzl, which will be used for the subexponential bound of section 1.4 and which will be discussed again in chapter 3 (then in pivot form). We call this algorithm **BasisLP** since the key idea is to keep the information of a basis which will be improved until it is optimal.

We give a short description of the algorithm: The input of **BasisLP**(G, F) is a set $G \subseteq H$ and a basis $F \subseteq G$; we want to find and output an optimal basis B of G.

- If $G = F$, then the solution is trivial (namely $B = F$).
- If $G \neq F$, then we solve the problem by recursion: For an arbitrary $h \in G \setminus F$ (we choose h at random, where every entry of $G \setminus F$ is chosen with the same probability) we compute (by a call of **BasisLP**$(G \setminus \{h\}, F)$) an optimal basis $\tilde{F}$ of $G \setminus \{h\}$; a *violation test* determines whether $\tilde{F}$ is also an optimal basis of G or not: In the first case we can return $B = \tilde{F}$, in the second case we have another recursive call with arguments G and x, where x is an optimal basis of $\tilde{F} \cup \{h\}$ (computed by a given algorithm **Basis**). The miracle will be that x is significantly better than the previous basis F; we will be sure that the recursion has to stop at a certain level, and we expect that this will happen after a certain (low) run time.

Algorithm BasisLP:
Input: (H, v) specifying an optimization problem of LP-type, a set of constraints $G \subseteq H$, a basis $F \subseteq G$; furthermore there is given an algorithm **Basis**(S: a subset of H) which computes an optimal basis of S; finally there is given as a computational primitive a violation test of the form "given any basis $x \subseteq H$ and any constraint $h \in H$, is h violated by x?".
Output: An optimal basis B of G.

begin BasisLP(G, F);
  if $G = F$ **then return** F
  else
    choose $h \in G \setminus F$ at random;
    $\tilde{F} :=$ **BasisLP**$(G \setminus \{h\}, F)$;
    if h not violated by $\tilde{F}$ **then return** $\tilde{F}$
    else return BasisLP(G, **Basis**$(\tilde{F} \cup \{h\})$)
    endif
  endif
end BasisLP.

We remark at this point that the computation of **Basis**$(\tilde{F} \cup \{h\})$ has to be easy (e.g. with costs polynomial in $\dim(H, v)$); if this computation would be difficult, the algorithm **BasisLP** would not be of practical interest. The same is true for the violation test: It has to be easy.

We give the last definition in the framework of LP-type problems:

Definition 1.10 (Enforcing Constraint) *Given an optimization problem of LP-type specified by (H, v) and F, G with $F \subseteq G \subseteq H$ and any $h \in G$.*
h is called enforcing in (G, F), *if $v(G \setminus \{h\}) > v(F)$.*
$\mathcal{E}(G, F) := \{h \in H | h$ *enforcing in* $(G, F)\}$.

The following lemma gives elementary properties of enforcing constraints; remark that point (iii) gives a relation between enforcing constraints and combinatorial dimension.

Lemma 1.11 *Given an optimization problem of LP-type specified by (H, v) and subsets F, G with $F \subseteq G \subseteq H$ and any $h \in H$, such that h is enforcing in (G, F). Then:*

(i) $h \in F$, *i.e.* $\mathcal{E}(G, F) \subseteq F$.

(ii) For all S with $F \subseteq S \subseteq G$: h is extreme in S.

(iii) $|\mathcal{E}(G, F)| \leq \dim(H, v)$.

Proof:

(i) Assume $h \notin F$. Then $F \subseteq G \setminus \{h\}$ implies (because of monotonicity and because h is enforcing in (G, F)) $v(F) \geq v(G \setminus \{h\}) > v(F)$, a contradiction.

(ii) Given S with $F \subseteq S \subseteq G$, then (because of monotonicity and because h is enforcing in (G, F)) $v(S \setminus \{h\}) \geq v(G \setminus \{h\}) > v(F) \geq v(S)$, i.e. h is extreme in S.

(iii) follows by (ii) and Lemma 1.8 (ii).

■

For the next lemma consider the algorithm **BasisLP**, especially the arguments of the second recursive call; the points (iii) and (iv) of the lemma contain the properties which are fundamental for the analysis of **BasisLP**.

Lemma 1.12 *Given an optimization problem of LP-type specified by (H, v), subsets F, S, and G with $F \subseteq S \subseteq G \subseteq H$, $h \in S \setminus F$ extreme in S, $\tilde{F}$ an optimal basis of $S \setminus \{h\}$, and x an optimal basis of $\tilde{F} \cup \{h\}$. Then:*

(i) *If $g \in H$ is enforcing in (G, F), then g is enforcing in (S, x).*

(ii) *h is enforcing in (S, x).*

(iii) *$\mathcal{E}(G, F) \subsetneqq \mathcal{E}(S, x)$.*

(iv) *If $g \in S$ satisfies $v(S \setminus \{g\}) \geq v(S \setminus \{h\})$, then g is enforcing in (S, x).*

Proof:

(i) Given $g \in H$ enforcing in (G, F), i.e. $v(G \setminus \{g\}) > v(F)$. Then:

$$v(S \setminus \{g\}) \geq v(G \setminus \{g\}) > v(F) \geq v(S \setminus \{h\}) = v(\tilde{F}) \geq v(\tilde{F} \cup \{h\}) = v(x),$$

where we use several times monotonicity and the definition of an optimal basis, and in the second inequality that g is enforcing in (G, F).

(ii) h extreme in $S \Rightarrow h$ violated by $S \setminus \{h\}$, so (by Lemma 1.8 (iii)) h violated by $\tilde{F}$ and then $v(S \setminus \{h\}) = v(\tilde{F}) > v(\tilde{F} \cup \{h\}) = v(x)$.

(iii) From (i) we know $\mathcal{E}(G, F) \subseteq \mathcal{E}(S, x)$, and from (ii) $h \in \mathcal{E}(S, x)$; by Lemma 1.11 (i) $h \in \mathcal{E}(G, F)$ is not possible, since $h \notin F$.

(iv) $v(S \setminus \{g\}) \geq v(S \setminus \{h\}) > v(x)$ (the second inequality follows from (ii)).

■

The next two theorems give the analysis of the algorithm **BasisLP**.

Theorem 1.13 (Correctness and Finiteness of BasisLP)

(i) *For any F, G with $F \subseteq G \subseteq H$, F a basis:*
When **BasisLP** *(G, F) terminates, then the output is an optimal basis of G.*

(ii) *For any F, G with $F \subsetneqq G \subseteq H$, F a basis:*
A call of **BasisLP** *(G, F) causes (on the top level of the recursion) one or two calls of* **BasisLP***: One call of* **BasisLP** *(G_1, F) with*

$$G_1 \subsetneqq G,\ |G_1| = |G| - 1 \text{ and } \mathcal{E}(G, F) \subseteq \mathcal{E}(G_1, F),$$

and at most one further call of **BasisLP** *(G, x) with*

$$\mathcal{E}(G, F) \subsetneqq \mathcal{E}(G, x).$$

(iii) *For any F, G with $F \subseteq G \subseteq H$, F a basis:*
BasisLP *(G, F) terminates after at most $2^{|G|-|\mathcal{E}(G,F)|+1} - 2$ further calls of* **BasisLP**.

Proof:

(i) We consider the tree of the recursive calls of **BasisLP**. If for given $F \subseteq G$ **BasisLP**(G, F) terminates, then the recursion tree is finite and we can define its height $n_{G,F}$ in the common sense, i.e. here

$$n_{G,F} := \begin{cases} 1 & \text{if } G = F \\ \max\{n_{G_1,F_1}, n_{G_2,F_2}\} + 1 & \text{if } G \neq F \end{cases},$$

where G_1, F_1 and G_2, F_2 are the arguments of the recursive calls of **BasisLP** at the same recursion level (if there is no second call, we set $n_{G_2,F_2} := 0$). We consider the recursion tree according to the call of **BasisLP**(G, F) and all subtrees (each according to the call of **BasisLP**$(\bar{G}, \bar{F})$ for some $\bar{F}, \bar{G}$). With induction in the height of the recursion (sub-)trees we prove that the output of **BasisLP**$(\bar{G}, \bar{F})$ is an optimal basis of G:
$n_{\bar{G},\bar{F}} = 1$: We have $\bar{G} = \bar{F}$ (otherwise $n_{\bar{G},\bar{F}} > 1$), so the return value is $\bar{F}$, a basis and obviously optimal for $\bar{G}$.
$n_{\bar{G},\bar{F}} > 1$: We assume by induction that all calls of **BasisLP** which cause a tree of smaller height than $n_{\bar{G},\bar{F}}$ have as return value an optimal basis of the first argument. We consider the algorithm after the call of **BasisLP**$(\bar{G}, \bar{F})$: Since $\bar{G} \neq \bar{F}$ (otherwise $n_{\bar{G},\bar{F}} = 1$), we choose $h \in \bar{G} \setminus \bar{F}$, compute recursively an optimal basis $\tilde{F}$ of $\bar{G} \setminus \{h\}$ (since $n_{\bar{G}\setminus\{h\},\bar{F}} = n_{G_1,F_1} < n_{\bar{G},\bar{F}}$, we can use the assumption from the induction). If h is not violated by $\tilde{F}$, then the return value is $\tilde{F}$, a basis with $v(\tilde{F}) = v(\bar{G} \setminus \{h\}) = v(\bar{G})$ (by lemma 1.8 (iii) and (iv)) and $\tilde{F} \subseteq \bar{G} \setminus \{h\} \subseteq \bar{G}$, i.e. $\tilde{F}$ is an optimal basis of $\bar{G}$. If h is violated by $\tilde{F}$, then the return value is **BasisLP**$(\bar{G}, x)$, where $x \subseteq \tilde{F} \cup \{h\} \subseteq \bar{G}$ is a basis, so (since $n_{\bar{G},x} = n_{G_2,F_2} < n_{\bar{G},\bar{F}}$) the return value is an optimal basis of $\bar{G}$.

(ii) The call of **BasisLP**(G_1, F) with $G_1 = G \setminus \{h\}$ for an $h \in G \setminus F$ is obvious; if this terminates, we have $\tilde{F} :=$ **BasisLP**$(G\setminus\{h\}, F)$, i.e. $\tilde{F}$ is an optimal basis of $G\setminus\{h\}$ (by (i)). Furthermore we have $\mathcal{E}(G, F) \subseteq \mathcal{E}(G \setminus \{h\}, F)$: Given $g \in G$ such that $v(G \setminus \{g\}) > v(F)$, then $v((G\setminus\{h\})\setminus\{g\}) \geq v(G\setminus\{g\}) > v(F)$. If h is violated by $\tilde{F}$, then we have one further call of **BasisLP**(G, x) with x an optimal basis of $\tilde{F} \cup \{h\}$. Since $h \in G \setminus F$ is extreme in G ($v(G \setminus \{h\}) = v(\tilde{F}) > v(\tilde{F} \cup \{h\}) \geq v(G)$), the conditions of lemma 1.12 hold (with $S = G$); lemma 1.12 (iii) then gives $\mathcal{E}(G, F) \subsetneqq \mathcal{E}(G, x)$.

(iii) For given $\bar{F}, \bar{G}$, where $\bar{F} \subseteq \bar{G} \subseteq H$ and $\bar{F}$ is a basis, define $m_{\bar{G},\bar{F}} := |\bar{G}| - |\mathcal{E}(\bar{G}, \bar{F})|$. By $\bar{F} \subseteq \bar{G}$ and lemma 1.11 (i): $m_{\bar{G},\bar{F}} \geq 0$. By (ii) we know that a call of **BasisLP**$(\bar{G}, \bar{F})$ with $\bar{F} \neq \bar{G}$ causes (on top level) one or two calls: For the first call of **BasisLP**$(G_1, \bar{F})$ we have

$$m_{G_1,\bar{F}} = |\bar{G}| - 1 - |\mathcal{E}(G_1, \bar{F})| \leq |\bar{G}| - |\mathcal{E}(\bar{G}, \bar{F})| - 1 = m_{\bar{G},\bar{F}} - 1. \tag{1.4}$$

For the second call of **BasisLP**$(\bar{G}, x)$ we have

$$m_{\bar{G},x} = |\bar{G}| - |\mathcal{E}(\bar{G}, x)| \leq |\bar{G}| - |\mathcal{E}(\bar{G}, \bar{F})| - 1 = m_{\bar{G},\bar{F}} - 1. \tag{1.5}$$

We complete the proof by induction over $m_{\bar{G},\bar{F}}$:
If $m_{\bar{G},\bar{F}} = 0$, then $\bar{F} = \bar{G}$ (since by lemma 1.11 (i) $\mathcal{E}(\bar{G}, \bar{F}) \subseteq \bar{F} \subseteq \bar{G}$): There is no further call of **BasisLP** (and in fact $2^1 - 2 = 0$).
If $m_{\bar{G},\bar{F}} > 0$: The case $\bar{F} = \bar{G}$ is trivial, so we assume $\bar{F} \neq \bar{G}$. By induction and the inequalities (1.4) and (1.5) the (at most) two calls of **BasisLP** cause each at most $2^{m_{\bar{G},\bar{F}}} - 2$ further calls, so the call of **BasisLP**$(\bar{G}, \bar{F})$ causes at most $2 + 2 \cdot (2^{m_{\bar{G},\bar{F}}} - 2) = 2^{m_{\bar{G},\bar{F}}+1} - 2$ further calls of **BasisLP**.

■

Definition 1.14 ($T(m,k)$) *For $m,k \in \mathbb{Z}$, $0 \le m \le |H|$, $0 \le k \le \min\{m, \dim(H,v)\}$:*
$T(m,k) :=$ maximal expected number of calls of **Basis** *after a call of* **BasisLP***(G,F), where the maximum is taken over all F,G with $F \subseteq G \subseteq H$, F a basis, $m = |G|$, $k \le |\mathcal{E}(G,F)|$ (if there are no such F,G for given m,k, then we set $T(m,k) := 0$).*

Theorem 1.15 (Expected Run Time of BasisLP) *With $d := \dim(H,v)$:*

(i) For all m with $0 \le m \le d$: $T(m,m) = 0$.

(ii) For all m and k with $0 \le k < m \le |H|, k \le d$:

$$T(m,k) \le T(m-1,k) + \frac{1}{\max\{m-d,1\}} \sum_{i=1}^{\min\{d-k, m-\tilde{d}\}} (1 + T(m,k+i)),$$

where $\tilde{d} := \min\left\{|\tilde{F}| \mid \tilde{F} \subseteq \tilde{G}, |\tilde{G}| = m, \tilde{F} \text{ a basis}, k \le |\mathcal{E}(\tilde{G},\tilde{F})|\right\}$ (so we have for $\tilde{d}$ the inequalities $d \ge \tilde{d} \ge k$ and $m \ge \tilde{d}$).

(iii) For all m with $d < m \le |H|$: $T(m,d) = 0$.

(iv) If for every call of **BasisLP***$(\bar{G},\bar{F})$ with $\bar{F} \subseteq \bar{G}$, $\bar{F}$ a basis, holds $|\bar{F}| = \min\{|\bar{G}|, d\}$, then the function $\beta(\ell,r) := T(r+d, d-\ell) + 1$ holds*

$$\text{for } \ell, r \text{ with } 0 \le \ell \le d, 0 \le r \le |H| - d: \quad \beta(\ell,0) = \beta(0,r) = 1, \tag{1.6}$$

and

$$\text{for } \ell, r \text{ with } 1 \le \ell \le d, 1 \le r \le |H| - d: \quad \beta(\ell,r) \le \beta(\ell,r-1) + \frac{1}{r} \sum_{i=1}^{\min\{\ell,r\}} \beta(\ell - i, r). \tag{1.7}$$

Then follows from (1.6) and (1.7):

$$\text{For } \ell, r \text{ with } 0 \le \ell \le d, 0 \le r \le |H| - d: \quad \beta(\ell,r) \le e^{4\sqrt{\ell \ln(r+1)}}. \tag{1.8}$$

(v) **BasisLP***(G,F), $F \subseteq G \subseteq H$, F a basis, computes an optimal basis B of G in an expected number of at most $T(m,k)$ calls of* **Basis** *(with argument $\tilde{F} \cup \{h\}$, where $\tilde{F}$ is a basis and $h \in H \setminus \tilde{F}$), in an expected number of at most $m \cdot (T(m,k)+1)$ violation tests of the form "given any basis $x \subseteq H$ and any constraint $h \in H$, is h violated by x?", and in an expected number of at most $O(m \cdot (T(m,k)+1))$ elementary arithmetic operations which can be executed in constant time.*

Proof:

(i) follows by theorem 1.13 (iii).

(ii) Without loss of generality we assume $F \neq G$. After a call of **BasisLP**(G,F) we have (by theorem 1.13 (ii)) one call which causes an expected number of at most $T(m-1,\tilde{k})$ calls of **Basis** for a $\tilde{k} \ge k$, so at most $T(m-1,k)$ (by definition we have $T(m-1,\tilde{k}) \le T(m-1,k)$). The second call occurs in the case "h violated by $\tilde{F}$": For every $h \in G \setminus F$ this case occurs (by lemma 1.8 (iii)) if and only if $v(G \setminus \{h\}) > v(G)$, i.e. exactly when h is extreme in G. We denote with $h_1, \ldots, h_s$ the extreme constraints in G which are not in F, ordered in a way such that $v(G \setminus \{h_1\}) \ge v(G \setminus \{h_2\}) \ge \cdots \ge v(G \setminus \{h_s\})$; by lemma 1.11 (i) and (ii)

there are at least $|\mathcal{E}(G,F)| \geq k$ constraints in F which are extreme in G, so with lemma 1.8 (ii)

$$s \leq \min\{d-k, m-|F|\}.$$

Every constraint in $G \setminus F$ is chosen as h with probability $\frac{1}{|G\setminus F|} = \frac{1}{m-|F|}$. If h is not one of $h_1, \ldots, h_s$, then there is no further call of **BasisLP** or **Basis**; otherwise $h = h_i$ for some $i \in \{1, \ldots, s\}$, and with lemma 1.12 (iii) and (iv) we conclude $|\mathcal{E}(G,F)| + i \leq |\mathcal{E}(G,x)|$, where G, x are the arguments of the second call of **BasisLP**. This means that for each $i \in \{1, \ldots, s\}$ we have with probability $\frac{1}{m-|F|}$ an expected number of at most $1+T(m,k+i)$ further calls of **Basis**, so

$$\begin{aligned} T(m,k) &\leq T(m-1,k) + \frac{1}{m-|F|}\sum_{i=1}^{s}(1+T(m,k+i)) \\ &\leq T(m-1,k) + \frac{1}{\max\{m-d,1\}}\sum_{i=1}^{s}(1+T(m,k+i)) \end{aligned}$$

(remark that $m-d \leq m-|F|$ since F is a basis).

(iii) Consider (ii) with $k=d$: $d-k=0$, so $T(m,d) \leq T(m-1,d) \leq \cdots \leq T(d,d) = 0$ (by (i)).

(iv) For $\ell \geq 0$: $\beta(\ell,0) = T(d,d-\ell)+1 = 1$ (if $\ell = 0$: apply (i); otherwise apply (ii) (remark $\tilde{d} = m$): $T(d,d-\ell) \leq T(d-1,d-\ell) \leq \cdots \leq T(d-\ell,d-\ell) = 0$, where the last step follows by (i)).
For $r \geq 0$: $\beta(0,r) = T(r+d,d)+1 = 1$ (by (i), if $r=0$, or by (iii), if $r>0$).
For given $1 \leq \ell \leq d, 1 \leq r \leq |H|-d$ we set $m := r+d$ and $k := d-\ell$ and apply (ii) (remark $\tilde{d} = d$):

$$\begin{aligned} \beta(\ell,r) &= T(m,k)+1 \leq T(m-1,k)+1+\frac{1}{m-d}\sum_{i=1}^{\min\{d-k,m-d\}}(1+T(m,k+i)) \\ &= \beta(\ell,r-1)+\frac{1}{r}\sum_{i=1}^{\min\{d-k,m-d\}}\beta(d-k-i,m-d) \\ &= \beta(\ell,r-1)+\frac{1}{r}\sum_{i=1}^{\min\{\ell,r\}}\beta(\ell-i,r). \end{aligned}$$

The proof of (1.8) can be found in [6] and follows from (1.6) and (1.7).

(v) Consider a run of the algorithm **BasisLP** and its recursion tree: Each call of **Basis** is followed by a call of **BasisLP** which can obviously cause at most $|G| = m$ violation tests *before the next node in the recursion tree with a call of* **Basis** *is reached;* together with the initial call of **BasisLP** we have an upper bound of at most $m \cdot$ (number of calls of **Basis** $+1$) violation tests. The costs of the elementary arithmetic operations are $O(1)$ for every run through the **else** part of the algorithm **BasisLP**.

■

The additional assumptions in theorem 1.15 (iv) will be proved to be satisfied when we apply the abstract framework in the context of linear programming problems.

There exist examples of LP-type problems for which the algorithm **BasisLP** has an expected run time close to the upper bound (see [7]); anyway, there are no known linear programming problems with such a long run time.

1.4 A Subexponential Bound for Linear Programming

The algorithm of Matoušek, Sharir, Welzl does already lead to a subexponential bound for the expected run time of $O(dn \cdot e^{4\cdot\sqrt{d\ln(n-d+1)}})$. A sampling technique of Clarkson which uses two algorithms (presented in the next two subsections) improves this bound to $O(\ln n \cdot e^{O(\sqrt{d\ln d})} + d^2 n)$ (see subsection 1.4.3).

1.4.1 Clarkson's First Algorithm

Algorithm ClarksonLP:
Input: (H, v) specifying an optimization problem of LP-type with $d = \dim(H, v)$; furthermore there is given an algorithm **ClarksonSubLP**(S: a subset of H) which computes an optimal basis of S for $|S| \leq 3d\sqrt{n}$, $n := |H|$; finally there is given as a computational primitive a violation test of the form "given any basis $x \subseteq H$ and any constraint $h \in H$, is h violated by x?".
Output: An optimal basis B of H.

begin ClarksonLP(H);
  $n := |H|$;
  if $n \leq 9d^2$ **then return ClarksonSubLP**(H)
  else
    $G := \emptyset$; $V := H$; $r := \lfloor d\sqrt{n} \rfloor$;
    while $V \neq \emptyset$ **do**
      choose $R \subseteq H \setminus G$ at random with $|R| = \min\{r, |H \setminus G|\}$;
      $x :=$ **ClarksonSubLP**$(G \cup R)$;
      $V := \{h \in H | h$ violated by $x\}$;
      if $|V| \leq 2\sqrt{n}$ **then** $G := G \cup V$ **endif**
    endwhile;
    return x
  endif
end ClarksonLP.

Lemma 1.16 *In the* **while***-loop of algorithm* **ClarksonLP** *the following is true immediately before the execution of the* **if***-statement:*

$$V \neq \emptyset \Rightarrow \text{ for any optimal basis } B \text{ of } H : B \cap V \neq \emptyset.$$

Proof: We will show the contraposition: Given any optimal basis B of H with $B \cap V = \emptyset$, we have to show $V = \emptyset$.
$B \cap V = \emptyset$ implies $B \subseteq \{h \in H | h$ not violated by $x\}$, so with lemma 1.8 (iv) and monotonicity

$$v(B) = v(H) \leq v(x) = v(x \cup \{h \in H | h \text{ not violated by } x\}) \leq v(x \cup B) \leq v(B),$$

i.e. $v(x) = v(H)$, and since x is a basis it follows that x is an optimal basis of H, and then lemma 1.8 (iii) implies $V = \emptyset$. ∎

Lemma 1.17 *After at most* $\dim(H, v)$ *iterations of the* **while***-loop in algorithm* **ClarksonLP** *with* $|V| \leq 2\sqrt{n}$ *in the* **if***-statement the case* $V = \emptyset$ *occurs; then* x *is an optimal basis of* H.

Proof: Consider any optimal basis B of H (existence see theorem 1.3). As we have seen in lemma 1.16, each time when we have $V \neq \emptyset$ and $|V| \leq 2\sqrt{n}$ in the **if**-statement, at least one new constraint h of B is in V (and, according to lemma 1.8 (iii), h violates $G \cup R$, so $h \notin G$), which is added to G. This can happen at most $|B|$ times, so at most $\dim(H, v)$ times. When we have $V = \emptyset$, lemma 1.8 (iv) implies $v(x) = v(x \cup H) = v(H)$, i.e. x is an optimal basis of H. ■

Lemma 1.18 *Given an optimization problem of LP-type specified by (H, v) with $d = \dim(H, v)$, $|H| = n$ and $G \subseteq H$. For $r \in \{0, 1, \ldots, n - |G|\}$ and a random subset $R \subseteq H \setminus G$ with $|R| = r$ the expected size $E(|V_R|)$ of $V_R := \{h \in H | h$ violated by $x\}$, x any optimal basis of $G \cup R$ (according to lemma 1.8 (iii), the choice of x does not affect the definition of V_R), is bounded by*

$$E(|V_R|) \leq d \cdot \frac{n - |G| - r}{r + 1}.$$

Proof: For $S \subseteq H \setminus G$ (S of any size) and $h \in H$ we define $\chi_G(S, h)$ by setting its value to 1 if $h \in V_S$, to 0 otherwise, where V_S is defined in the same way as V_R. Then

$$\begin{aligned}
\binom{n - |G|}{r} E(|V_R|) &= \sum_{\substack{R \subseteq H \setminus G \\ |R| = r}} |V_R| = \sum_{\substack{R \subseteq H \setminus G \\ |R| = r}} \sum_{h \in H \setminus (G \cup R)} \chi_G(R, h) \\
&= \sum_{\substack{Q \subseteq H \setminus G \\ |Q| = r+1}} \sum_{h \in Q} \chi_G(Q \setminus \{h\}, h) \leq \sum_{\substack{Q \subseteq H \setminus G \\ |Q| = r+1}} d = d \binom{n - |G|}{r + 1},
\end{aligned}$$

where the first and second equality is a write-out of definitions and the third equality a rewriting of sums; the inequality follows from the fact, that for $h \in Q$ for fixed $Q \subseteq H \setminus G$

$$\begin{aligned}
\chi_G(Q \setminus \{h\}, h) = 1 &\Leftrightarrow v(x) > v(x \cup \{h\}), x \text{ an optimal basis of } G \cup (Q \setminus \{h\}) = (G \cup Q) \setminus \{h\} \\
&\Leftrightarrow v((G \cup Q) \setminus \{h\}) > v(G \cup Q) \text{ (by locality)} \\
&\Leftrightarrow h \text{ is extreme in } G \cup Q,
\end{aligned}$$

and (with lemma 1.8 (ii)) there are at most d extreme constraints in $G \cup Q$. So

$$E(|V_R|) \leq d \cdot \frac{(n - |G|)! r! (n - |G| - r)!}{(r + 1)! (n - |G| - r - 1)! (n - |G|)!} = d \cdot \frac{n - |G| - r}{r + 1}.$$

■

Theorem 1.19 (Markov Inequality) *Given $X \geq 0$ a random variable with discrete random distribution and $E(X) > 0$, $\alpha > 0$, then $\Pr(X \geq \alpha \cdot E(X)) \leq \frac{1}{\alpha}$.*

Proof:

$$\begin{aligned}
E(X) &= \sum_{\substack{0 \leq y \\ \Pr(X = y) > 0}} y \cdot \Pr(X = y) = \sum_{\substack{0 \leq y < \alpha \cdot E(X) \\ \Pr(X = y) > 0}} y \cdot \Pr(X = y) + \sum_{\substack{\alpha \cdot E(X) \leq y \\ \Pr(X = y) > 0}} y \cdot \Pr(X = y) \\
&\geq 0 + \alpha \cdot E(X) \cdot \sum_{\substack{\alpha \cdot E(X) \leq y \\ \Pr(X = y) > 0}} \Pr(X = y) = \alpha \cdot E(X) \cdot \Pr(X \geq \alpha \cdot E(X)) \\
&\Rightarrow \Pr(X \geq \alpha \cdot E(X)) \leq \frac{1}{\alpha}.
\end{aligned}$$

■

Corollary 1.20 *In the* **while***-loop of algorithm* **ClarksonLP** *we have*

$$\Pr(|V| \leq 2\sqrt{n}) \geq \frac{1}{2}.$$

Proof: With $r := \lfloor d\sqrt{n} \rfloor$ and lemma 1.18 we have

$$E(|V|) \leq d \cdot \frac{n - |G| - r}{r+1} \leq d \cdot \frac{n}{r+1} = \frac{dn}{\lfloor d\sqrt{n} \rfloor + 1} \leq \frac{dn}{d\sqrt{n}} = \sqrt{n},$$

so theorem 1.19 with $X = |V|$ and $\alpha = 2$ implies $\Pr(|V| \geq 2\sqrt{n}) \leq \frac{1}{2}$, so

$$\Pr(|V| \leq 2\sqrt{n}) \geq \Pr(|V| < 2\sqrt{n}) = 1 - \Pr(|V| \geq 2\sqrt{n}) \geq \frac{1}{2}.$$

■

Corollary 1.21 *The expected number of iterations of the* **while***-loop in algorithm* **ClarksonLP** *is bounded by*

$$2 \cdot \dim(H, v) + 1.$$

Proof: Follows from lemma 1.17 and corollary 1.20. ■

Theorem 1.22 (Analysis of ClarksonLP) *With* $d := \dim(H, v)$

(i) Whenever **ClarksonLP** *calls* **ClarksonSubLP** *with argument* $S \subseteq H$*, then* $|S| \leq 3d\sqrt{n}$*.*

(ii) **ClarksonLP** *computes an optimal basis* B *of* H *in an expected number of at most* $2d+1$ *calls of* **ClarksonSubLP** *(with at most* $3d\sqrt{n}$ *constraints), in an expected number of at most* $(2d+1) \cdot n$ *violation tests of the form "given any basis* $x \subseteq H$ *and any constraint* $h \in H$*, is* h *violated by* x*?", and in an expected number of at most* $O(dn)$ *elementary arithmetic operations which can be executed in constant time.*

Proof:

(i) If $n \leq 9d^2$, then we have one call of **ClarksonSubLP** with $n = \sqrt{n}\sqrt{n} \leq 3d\sqrt{n}$ constraints. If $n > 9d^2$, then we call **ClarksonSubLP** with $G \cup R$, where $|G| \leq d \cdot 2\sqrt{n}$ (see lemma 1.17) and $|R| \leq r \leq d\sqrt{n}$.

(ii) If $n \leq 9d^2$, then we once call **ClarksonSubLP**.
If $n > 9d^2$, then with corollary 1.21 and $|H| = n$ we have found the result of (ii) for the calls of **ClarksonSubLP** and for the violation tests. The arithmetic operations are caused by arithmetic with real numbers and set operations; when we assume that the computation of $\lfloor d\sqrt{n} \rfloor$ is $O(dn)$ and all elementary arithmetic on real numbers (as $+, *$) is $O(1)$ and operations like $\cup, \setminus$ on subsets of H are $O(n)$, then (by corollary 1.21) the total expected costs for all arithmetic operations are at most $O(dn)$.

■

1.4.2 Clarkson's Second Algorithm

Algorithm ClarksonSubLP:
Input: (H, v) specifying an optimization problem of LP-type with $d = \dim(H, v)$; furthermore there is given an algorithm **SubLP**(S: a subset of H) which computes an optimal basis of S for $|S| \leq 6d^2$; finally there is given as a computational primitive a violation test of the form "given any basis $x \subseteq H$ and any constraint $h \in H$, is h violated by x?".
Output: An optimal basis B of H.
Remark: In this algorithm **ClarksonSubLP** we denote with H_w the set (or, in fact, the multiset) where each constraint $h \in H$ occurs with multiplicity w_h, so $|H_w| = \sum_{h \in H} w_h$. When we write $S \cap H$ for a multiset $S \subseteq H_w$, this means the set where every constraint $h \in S$ occurs only once and not with multiplicity w_h as in S.

begin ClarksonSubLP(H);
  $n := |H|$;
  if $n \leq 6d^2$ **then return SubLP**(H)
  else
    for all $h \in H$ **do** $w_h := 1$ **endfor**;
    $V := H$; $r := 6d^2$;
    while $V \neq \emptyset$ **do**
      choose $R \subseteq H_w$ (as a multiset) at random with $|R| = r$;
      $x :=$ **SubLP**$(R \cap H)$;
      $V := \{h \in H_w | h$ violated by $x\}$ (as a multiset);
      if $|V| \leq \frac{1}{3d}|H_w|$ **then for** all $h \in V \cap H$ **do** $w_h := 2w_h$ **endfor endif**
    endwhile;
    return x
  endif
end ClarksonSubLP.

Lemma 1.23 *$d := \dim(H, v)$ and $k \in \{1, 2, \dots\} \Rightarrow$ after kd iterations of the* **while***-loop in algorithm* **ClarksonSubLP** *with $0 < |V| \leq \frac{1}{3d}|H_w|$ we have for every optimal basis B of H*

$$|B| \cdot 2^k \leq \sum_{h \in B} w_h < n \cdot e^{\frac{k}{3}}.$$

Proof: Assume there have been (exactly) kd iterations of the **while**-loop in algorithm **ClarksonSubLP** with $0 < |V| \leq \frac{1}{3d}|H_w|$, and consider any optimal basis B of H. As in algorithm **ClarksonLP** we have here $V \neq \emptyset \Rightarrow B \cap V \neq \emptyset$ (see lemma 1.16), i.e. in each iteration with $0 < |V| \leq \frac{1}{3d}|H_w|$ there was at least one constraint $h \in B$ for which w_h was doubled, so with $n_h :=$ the number of times w_h was doubled:

$$\sum_{h \in B} w_h = \sum_{h \in B} 2^{n_h} = |B| \cdot \sum_{h \in B} \frac{1}{|B|} 2^{n_h} \geq |B| \cdot \prod_{h \in B} 2^{n_h \cdot \frac{1}{|B|}} = |B| \cdot 2^{\frac{1}{|B|} \cdot \sum_{h \in B} n_h} \geq |B| \cdot 2^{\frac{1}{|B|} \cdot kd} \geq |B| \cdot 2^k,$$

where the first inequality follows from the inequality between arithmetic and geometric mean, and the last inequality follows from $|B| \leq d$. On the other hand we augment in every iteration with $0 < |V| \leq \frac{1}{3d}|H_w|$ the actual value $|H_w|$ by $|V| \leq \frac{1}{3d}|H_w|$, so after kd such steps

$$\sum_{h \in B} w_h \leq \sum_{h \in H} w_h \leq n \cdot \left(1 + \frac{1}{3d}\right)^{kd} < n \cdot e^{\frac{k}{3}}$$

(for the last inequality, we consider e.g. for $a > 0$ and $b > 0$:
$\ln(1 + a) < a \Rightarrow b \ln(1 + a) < ba \Rightarrow (1 + a)^b < e^{ba}$). ∎

Theorem 1.24 (Analysis of ClarksonSubLP) *With* $d := \dim(H, v)$

(i) Whenever **ClarksonSubLP** *calls* **SubLP** *with argument* $R \subseteq H$*, then* $|R| \leq 6d^2$.

(ii) **ClarksonSubLP** *computes an optimal basis* B *of* H *in an expected number of at most* $2d\lceil 3 \ln n \rceil$ *calls of* **SubLP** *(with at most* $6d^2$ *constraints), in an expected number of at most* $2d\lceil 3 \ln n \rceil \cdot n$ *violation tests of the form "given any basis* $x \subseteq H$ *and any constraint* $h \in H$*, is* h *violated by* x*?", and in an expected number of at most* $O(dn \ln n)$ *elementary arithmetic operations which can be executed in constant time.*

Proof:

(i) is obvious.

(ii) We have to show that for $n > 6d^2$ the expected number of iterations of the **while**-loop until $V = \emptyset$ is at most $2d\lceil 3 \ln n \rceil$. We will apply lemma 1.18 to **ClarksonSubLP**, but first we have to show that the multiset problem is an optimization problem of LP-type. For a fixed iteration of the **while**-loop we have at the beginning of its execution a multiset H_w and define for every $S \subseteq H_w$ (S as a multiset) the value of S by $\tilde{v}(S) := v(S \cap H)$, where $S \cap H$ is the set according to the multiset S as mentioned in the algorithm **ClarksonSubLP**, and v is the value function of the given optimization problem (H, v). Then $(H_w, \tilde{v})$ is an optimization problem of LP-type:

- **Monotonicity:** Given multisets F, G with $F \subseteq G \subseteq H_w$, then $F \cap H \subseteq G \cap H$, so $\tilde{v}(F) = v(F \cap H) \geq v(G \cap H) = \tilde{v}(G)$
- **Locality:** Given two multisets F, G with $F \subseteq G \subseteq H_w$ and $\tilde{v}(F) = \tilde{v}(G)$ and any constraint $h \in H_w$, such that $\tilde{v}(G) > \tilde{v}(G \cup \{h\})$.
  Then $F \cap H \subseteq G \cap H$, $v(F \cap H) = \tilde{v}(F) = \tilde{v}(G) = v(G \cap H)$, and
  $v(G \cap H) = \tilde{v}(G) > \tilde{v}(G \cup \{h\}) = v((G \cup \{h\}) \cap H) = v(((G \cap H) \cup \{h\}) \cap H)$,
  so with the locality for (H, v) follows
  $\tilde{v}(F) = v(F \cap H) > v(((F \cap H) \cup \{h\}) \cap H) = v((F \cup \{h\}) \cap H) = \tilde{v}(F \cup \{h\})$.

Furthermore we have to show that for any $R \subseteq H_w$ an optimal basis x of $R \cap H$ with respect to the problem (H, v) is an optimal basis of R with respect to the problem $(H_w, \tilde{v})$. Since x is a set (and not a true multiset), every multiset $\tilde{x} \subsetneq x$ is also a set, i.e. $\tilde{x} \cap H = \tilde{x}$; then we conclude $\tilde{v}(\tilde{x}) = v(\tilde{x}) > v(x) = \tilde{v}(x)$, i.e. x is a basis with respect to $(H_w, \tilde{v})$, and with $\tilde{v}(x) = v(x) = v(R \cap H) = \tilde{v}(R)$ together with $x \subseteq R \cap H \subseteq R$ follows that x is an optimal basis of R.
Now we are able to apply lemma 1.18 to $(H_w, \tilde{v})$ (remark that indeed we solve in the algorithm **ClarksonSubLP** exactly the problem $(H_w, \tilde{v})$) and conclude

$$E(|V|) \leq d \cdot \frac{|H_w| - r}{r + 1} \leq d \cdot \frac{|H_w|}{r} = \frac{d}{6d^2}|H_w| = \frac{1}{6d}|H_w|,$$

and with theorem 1.19 follows

$$\begin{aligned} \Pr(|V| \leq \frac{1}{3d}|H_w|) \;&\geq\; \Pr(|V| < \frac{1}{3d}|H_w|) = 1 - \Pr(|V| \geq \frac{1}{3d}|H_w|) \geq 1 - \Pr(|V| \geq 2E(|V|)) \\ &\geq\; 1 - \frac{1}{2} = \frac{1}{2}. \end{aligned}$$

By this result it remains to show that there are at most $d\lceil 3 \ln n \rceil - 1$ iterations of the **while**-loop with $0 < |V| \leq \frac{1}{3d}|H_w|$. If $B = \emptyset$ is an optimal basis of H, then the result is trivial; otherwise: For $\ell = 3 \ln n$ we have

$$2^\ell = e^{3 \ln n \cdot \ln 2} = n^{3 \ln 2} > n^2 = n \cdot e^{\frac{\ell}{3}},$$

and with $2 > e^{\frac{1}{3}}$ follows for $k = \lceil 3 \ln n \rceil$ the inequality $2^k > n \cdot e^{\frac{k}{3}}$, so lemma 1.23 implies that there are less than kd iterations of the **while**-loop with $0 < |V| \leq \frac{1}{3d}|H_w|$.

■

1.4.3 Combination of the Algorithms

In this subsection we put together the results of the analyses of the algorithms **ClarksonLP**, **ClarksonSubLP**, and **BasisLP**.

We sum up the results so far, where $T_{\mathbf{ClarksonSubLP}}, T_{\text{Violation Test}}, \ldots$ denote in an obvious way the expected run time costs of one call of the indicated subprogram; as usually n is the number of constraints and d the dimension:

- Expected run time of **ClarksonLP** (theorem 1.22):

$$\begin{aligned} T_{\mathbf{ClarksonLP}}(n,d) \quad \leq \quad & (2d+1) \cdot (T_{\mathbf{ClarksonSubLP}}(\leq 3d\sqrt{n}, d) \\ & + n \cdot T_{\text{Violation Test}}(\text{basis } x, h \in H)) \\ & + O(dn) \end{aligned}$$

- Expected run time of **ClarksonSubLP** (theorem 1.24):

$$\begin{aligned} T_{\mathbf{ClarksonSubLP}}(n_1,d) \quad \leq \quad & 2d\lceil 3 \ln n_1 \rceil \cdot (T_{\mathbf{SubLP}}(\leq 6d^2, d) \\ & + n_1 \cdot T_{\text{Violation Test}}(\text{basis } x, h \in H)) \\ & + O(dn_1 \ln n_1) \end{aligned}$$

- Expected run time of **BasisLP** (theorem 1.15 (v)):

$$\begin{aligned} T_{\mathbf{BasisLP}}(n_2,d) \quad \leq \quad & T(n_2,0) \cdot T_{\mathbf{Basis}}(\tilde{F} \cup \{h\}, \tilde{F} \text{ a basis}, h \in H \setminus \tilde{F}) \\ & + n_2 \cdot (T(n_2,0)+1) \cdot T_{\text{Violation Test}}(\text{basis } x, h \in H) \\ & + O(n_2 \cdot (T(n_2,0)+1)) \end{aligned}$$

- Expected number of calls of **Basis** from **BasisLP** (theorem 1.15 (iv)): If for every call of **BasisLP**$(\bar{G}, \bar{F})$ with $\bar{F} \subseteq \bar{G}$, $\bar{F}$ a basis, holds $|\bar{F}| = \min\{|\bar{G}|, d\}$, then for $d \leq m \leq |H|$ and $0 \leq k \leq d$:

$$T(m,k) = \beta(d-k, m-d) - 1 \leq e^{4 \cdot \sqrt{(d-k)\ln(m-d+1)}} - 1. \tag{1.9}$$

Until now we have avoided to make use of the special structure of linear programming in the study of LP-type problems. Now we focus on linear programming as a special case.

First we assume that in the two algorithms **ClarksonLP** and **ClarksonSubLP** the costs of a violation test $T_{\text{Violation Test}}(\text{basis } x, h \in H)$ are in $O(d)$; this will be obviously true for our techniques of representing linear programs as LP-type problems. This leads to

$$T_{\mathbf{ClarksonLP}}(n,d) \leq O(d) \cdot T_{\mathbf{ClarksonSubLP}}(\leq 3d\sqrt{n}, d) + O(d^2 n),$$

$$T_{\mathbf{ClarksonSubLP}}(n_1,d) \leq O(d \ln n_1) \cdot T_{\mathbf{SubLP}}(\leq 6d^2, d) + O(d^2 n_1 \ln n_1),$$

or, when we combine both algorithms (and remark $d \leq n$),

$$T_{\mathbf{ClarksonLP}}(n,d) \leq O(d^2 \ln n) \cdot T_{\mathbf{SubLP}}(\leq 6d^2, d) + O(d^4 \sqrt{n} \ln n) + O(d^2 n).$$

The next step is the completion of the run time analysis of **BasisLP**. Since we easily can transform any LP-type optimization problem (H, v) into a LP-type optimization problem $(H, \tilde{v})$ where (with $d = \dim(H, v)$)

$$\text{for any optimal basis } F \text{ of } G \text{ holds } |F| = \min\{|G|, d\} \tag{1.10}$$

(just define $\tilde{v}(G) := (v(G), \min\{|G|, d\})$ with the lexicographical order on $\mathcal{L} \times \{0, 1, \ldots, d\}$), we assume to have this property (1.10); then, once started with a call $\mathbf{BasisLP}(G, F)$ such that $|F| = \min\{|G|, d\}$ (if $|G| \leq d$, set $F := G$; otherwise take any $F \subsetneqq G$ with $|F| = d$: F will be a basis), all further calls of $\mathbf{BasisLP}(\bar{G}, \bar{F})$ have the property $|\bar{F}| = \min\{|\bar{G}|, d\}$ (when the initial G has cardinality $|G| \leq d$, then $\mathbf{BasisLP}(G, F) = \mathrm{G}$; otherwise all further calls have $|\bar{G}| \geq d$ and $|\bar{F}| = d$), i.e. we can apply the subexponential bound for the transformed problem $(H, \tilde{v})$ "find a set $B \subseteq H$ with $|B| = d$ and $v(B) = v(H)$": For this we have the bound

$$\begin{aligned} T_{\mathbf{BasisLP}}(n_2, d) \quad \leq \quad & e^{4 \cdot \sqrt{d \ln(n_2 - d + 1)}} \cdot (T_{\mathbf{Basis}}(\tilde{F} \cup \{h\}, \tilde{F} \text{ a basis}, h \in H \setminus \tilde{F}) \\ & + n_2 \cdot T_{\text{Violation Test}}(\text{basis } x, h \in H) \\ & + O(n_2)). \end{aligned}$$

For the violation test we assume as before that it is $O(d)$; for the basis computation we assume an order of $O(d^2)$ (we do not discuss this here, see e.g. [6]; the important observation is that the order is polynomial in d and n since the argument $\tilde{F} \cup \{h\}$ has this special form for $\tilde{F}$ a basis, $h \in H \setminus \tilde{F}$; a higher exponent than 2 or a polynomial dependence on n does not change the final results), so

$$\begin{aligned} T_{\mathbf{BasisLP}}(n_2, d) \quad \leq \quad & e^{4 \cdot \sqrt{d \ln(n_2 - d + 1)}} \cdot (O(d^2) + O(dn_2) + O(n_2)) \\ = \quad & O(dn_2 \cdot e^{4 \cdot \sqrt{d \ln(n_2 - d + 1)}}). \end{aligned}$$

Remark: When we discuss our setting of **BasisLP** in the pivot form, we will prove the same order of $O(dn_2 \cdot e^{4 \cdot \sqrt{d \ln(n_2 + 1)}})$; the difference "$n_2 + 1$" instead of "$n_2 - d + 1$" stems from a basis conversion which reduces the number of constraints by d; the new (reduced) number of constraints is denoted by n_2 again, i.e. the two terms reflect exactly the same cardinality of the (original) constraint set.

Now we combine the algorithms and use **BasisLP** instead of **SubLP** in **ClarksonSubLP**; then:

$$\begin{aligned} T_{\mathbf{ClarksonLP}}(n, d) \quad \leq \quad & O(d^2 \ln n) \cdot O(d^3 e^{4 \cdot \sqrt{2d \ln d}}) + O(d^4 \sqrt{n} \ln n) + O(d^2 n) \\ = \quad & O(\ln n \cdot e^{6 \cdot \sqrt{d \ln d} + 5 \ln d} + d^4 \sqrt{n} \ln n + d^2 n) \\ = \quad & O(\ln n \cdot e^{O(\sqrt{d \ln d})} + d^4 \sqrt{n} \ln n + d^2 n) \\ = \quad & O(\ln n \cdot e^{O(\sqrt{d \ln d})} + d^2 n); \end{aligned}$$

the last equality is true since the middle term $d^4 \sqrt{n} \ln n$ is dominated asymptotically (i.e. for large n) by one of the other terms (if $d^2 \leq \frac{\sqrt{n}}{\ln n}$, then $d^4 \sqrt{n} \ln n$ is dominated by $d^2 n$; otherwise we have (for large n) $\left(\sqrt{d \ln d}\right)^4 \geq d^2 \geq \frac{\sqrt{n}}{\ln n} \geq (\ln n)^4$, i.e. asymptotically $d^4 \sqrt{n} \ln n = e^{O(\ln n)}$ is dominated by $\ln n \cdot e^{O(\sqrt{d \ln d})}$).

Chapter 2

Transformation from LP to LP-Type

In this chapter we will discuss techniques which convert linear programming problems into optimization problems of LP-type (see chapter 1). After some notations concerning lexicographical order and some general remarks in the first section, we present several transformation techniques; some of them (section 2.2) are not very satisfying, while the transformations using combinatorial bounding boxes are considered to be a good answer for our transformation problem. The starting point is a technique of Seidel (and by the way, Seidel's incremental algorithm solving linear programming problems is presented; see subsection 2.3.1), which is then refined in subsection 2.3.2.

2.1 Notations and General Remarks

For notations concerning linear programming problems see section 1.1.

2.1.1 Lexicographical Order

We enlarge the real numbers by the infinity elements: $\bar{\mathbb{R}} := \mathbb{R} \cup \{-\infty, +\infty\}$; $(\bar{\mathbb{R}}, \leq)$ is a linearly ordered set with $-\infty < x < +\infty$ for $x \in \mathbb{R}$ as usual.

In the following we use the set of vectors of finite length

$$\mathcal{R} := \{(a_1, \ldots, a_k) | a_i \in \bar{\mathbb{R}}, k \geq 0\}$$

with the lexicographical order

$$(a_1, \ldots, a_{k_a}) \leq (b_1, \ldots, b_{k_b}) :\Leftrightarrow \begin{cases} \text{Either } k_a \leq k_b \text{ and } \forall i \in \{1, \ldots, k_a\} : a_i = b_i, \\ \text{or } \exists j \in \{1, \ldots, \min\{k_a, k_b\}\} : \forall i < j : a_i = b_i, \text{ and } a_j < b_j. \end{cases} \quad (2.1)$$

Remark: The empty vector $() \in \mathcal{R}$ is allowed and $\forall a \in \mathcal{R} : () \leq a$.

We denote with $\circ$ the concatenation of vectors in $\mathcal{R}$:

$$(a_1, \ldots, a_{k_a}) \circ (b_1, \ldots, b_{k_b}) := (a_1, \ldots, a_{k_a}, b_1, \ldots, b_{k_b}).$$

2.1.2 General Idea of the Transformation

Given a linear programming problem with constraint set H (cf. section 1.1), we have to define an optimization problem of LP-type (H, v), i.e. for every $G \subseteq H$ we have to define $v(G)$ such that monotonicity and locality hold. Naturally $v(G)$ is regarded as the optimal value of $(LP)\,|_G$:

- When $(LP)\,|_G$ is infeasible, we will mark this specially by setting $v(G)$ to a *minimal value* (e.g. $v(G) = -\infty$). We will see that this is appropriate with our aim.
- When $(LP)\,|_G$ has a finite optimal solution $x^* \in \mathbb{R}^d$, we might try to set $v(G) = c^T x^*$. This would not suffice for the locality: Figure 2.1 shows an example (the feasible side of each constraint is downwards). The key observation is that we have to force the solution x^* to be unique, and this uniqueness has to be taken into the definition of $v(G)$ (e.g. by defining $v(G) = (c^T x^*) \circ x^*$).
- When $(LP)\,|_G$ has unbounded solutions, then the definition of an optimal value $v(G)$ has to be done in a way which is reflecting some uniqueness as in the case of a finite optimal solution (otherwise, e.g. when we try $v(G) = +\infty$, the locality will fail). The definition of $v(G)$ for this unbounded case will be the major problem in the following discussion.

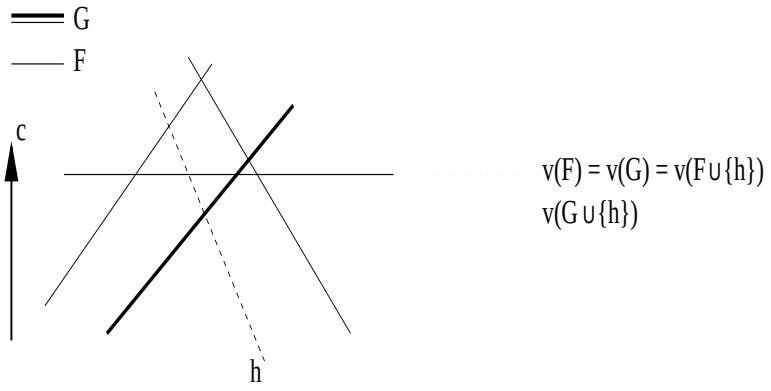

Figure 2.1: Linear Program without unique solution

2.2 Transformations with Drawbacks

2.2.1 Numerical Bounding Box

When for a given linear problem of the form (1.1) all entries in A and b are rational numbers (and in practice this is usually the case), then it is possible to compute a bound $N > 0$ such that all $x \in \mathbb{R}^d$ which are intersection of d linearly independent constraints lie in the box $-N < x_i < N$ for all $i \in \{1, \ldots, d\}$. We denote this box constraints by X. Instead of solving for $G \subseteq H$ the original problem, we solve the problem with constraints $G \cup X$, which then will have a finite optimal solution x^G:

Definition 2.1 (Optimal Solution x^G of LP with Numerical Bounding Box)
For a given linear programming problem (LP) and a bounding box X we write for $G \subseteq H$ short $(LP)\,|_G^{\square}$ for the linear program with constraints $G \cup X$ and same objective function c as (LP). We call $x^G \in \mathbb{R}^d$ the optimal solution of $(LP)\,|_G^{\square}$ *when it maximizes the vector $(c^T x, x_1, \ldots, x_d)$*

with respect to the lexicographical order for all $x \in \mathbb{R}^d$ which are feasible for the constraints in $G \cup X$.

We define now $v(G)$ and show then that we have an optimization problem of LP-type.

Definition 2.2 ((H, v) for LP with Numerical Bounding Box)
Given (LP) and a bounding box X, then we define for $G \subseteq H$ the value of G by

$$v(G) := \begin{cases} (-\infty) & \text{if } (LP)\,|_G^{\square} \text{ is infeasible} \\ (c^T x^G) \circ x^G & \text{if } (LP)\,|_G^{\square} \text{ has optimal solution } x^G \end{cases}$$

.

Theorem 2.3 (LP with Numerical Bounding Box is of LP-Type)
For any given (LP) specifies (H, v) as defined in definition 2.2 an optimization problem of LP-type. The combinatorial dimension of (H, v) is bounded by $\dim(H, v) \leq d + 1$ (in the case of feasibility even $\dim(H, v) \leq d$).

Proof: For the proof we refer to the similar proof of theorem 2.7: In each part (monotonicity, locality, dimension) only the first and second case has to be considered (infeasible linear program and finite solution); replace in the proof u^G etc. by the optimal solution x^G. ■

When we have solved the LP-type problem specified by (H, v), we can easily see whether the original problem (LP) was

- infeasible: if and only if $v(H) = (-\infty)$,
- feasible and bounded (in the sense of maximizing lexicographically $(c^T x, x_1, \ldots, x_d)$): if and only if $v(H) = (c^T x^H) \circ x^H$ and for all $i \in \{1, \ldots, d\}$: $-N < x_i^H < N$,
- feasible and unbounded (in the sense of maximizing lexicographically $(c^T x, x_1, \ldots, x_d)$): if and only if $v(H) = (c^T x^H) \circ x^H$ and there exists $i \in \{1, \ldots, d\}$: $x_i^H = -N$ or $x_i^H = +N$.

Drawbacks:

- In practice the bound $N > 0$ will be very large, so we would have to deal with very large numbers.
- If A and b are not rational, the computation of $N > 0$ is difficult (or even impossible).
- We can not design a purely combinatorial algorithm with a numerical bounding box as additional constraint set.

2.2.2 Two Combinatorial Methods with Large $\dim(H, v)$

We present two attempts of defining for given (LP) an optimization problem of LP-type (H, v) in a combinatorial way. For both we will see that the combinatorial dimension might be $O(|H|)$; since the algorithms (especially **ClarksonLP**, **ClarksonSubLP**) are efficient (i.e. useful) only for small $\dim(H, v)$, we can not work with these two definitions.

First Definition:

Similar to definition 2.1 (but without box constraints X) we say for $G \subseteq H$ that $x^G \in \mathbb{R}^d$ is the optimal solution of $(LP)\mid_G$ when it maximizes $(c^T x, x_1, \ldots, x_d)$ for all feasible x (i.e. then $(LP)\mid_G$ is *bounded* in this sense of optimization; otherwise $(LP)\mid_G$ is either *infeasible* or *unbounded*). Now we define for $G \subseteq H$

$$v(G) := \begin{cases} (-\infty) & \text{if } (LP)\mid_G \text{ is infeasible,} \\ (c^T x^G) \circ x^G & \text{if } (LP)\mid_G \text{ is bounded with optimal solution } x^G, \\ (+\infty, -|G|) & \text{if } (LP)\mid_G \text{ is unbounded.} \end{cases}$$

We omit the proof that with this definition we specify an optimization problem of LP-type; if there exists $G \subseteq H$ with $|G| > d$ and unbounded $(LP)\mid_G$, then

$$\dim(H, v) = \max\{|G| \mid (LP)\mid_G \text{ is unbounded}\},$$

i.e. we easily may have $\dim(H, v)$ in $O(|H|)$.

Second Definition:

Since in the last definition the part for unbounded $(LP)\mid_G$ (i.e. $v(G) := (+\infty, -|G|)$) was unlucky , we will replace this by a more sophisticated definition. The idea is to determine a unique *direction* $r^G \in \mathbb{R}^d$ *of* $(LP)\mid_G$ (i.e. a vector such that $x + \mu r^G$ is feasible for all $\mu \geq 0$ for some $x \in \mathbb{R}^d$) and to define in the unbounded case

$$v(G) := (+\infty) \circ f(r^G)$$

for a certain function f. In particular we define $f(r^G)$ to be a vector of numbers of constraints: First we enumerate $H = \{h_1, \ldots, h_n\}$ in an arbitrary way, then we define $f(r^G) := (i_1, \ldots, i_s)$ as the lexicographically smallest vector with $i_1 < \cdots < i_s$, where the intersection of the constraints $h_{i_1}, \ldots, h_{i_s}$ contains a direction of $(LP)\mid_G$ (let us say r^G), and for which s is maximal (i.e. there does not exist $i_{s+1} > i_s$ such that the intersection of $h_{i_1}, \ldots, h_{i_{s+1}}$ contains a direction of $(LP)\mid_G$). We can show monotonicity and locality (proof omitted), but again the combinatorial dimension might be in $O(n)$, as the example of figures 2.2 and 2.3 shows. The feasible region in the example is towards the inner of the cone. The upper circle in figure 2.3 marks the direction r^H as the intersection of h_{k+1} and h_{k+2}; when we consider $G = H \setminus \{h\}$ for any $h \in \{h_{k+1}, \ldots, h_{2k}\}$, then $f(r^G) \neq f(r^H)$ (r^G is marked by the other circle), i.e. the optimal basis of H is $\{h_{k+1}, \ldots, h_{2k}\}$: Here we have $\dim(H, v) \geq k = O(|H|)$.

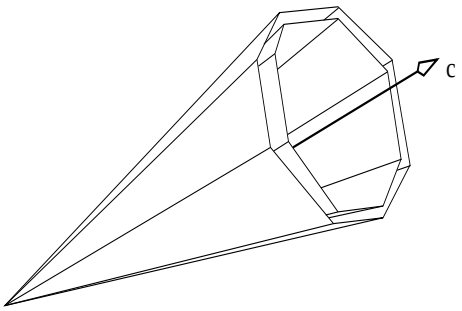

Figure 2.2: Cone in 3D

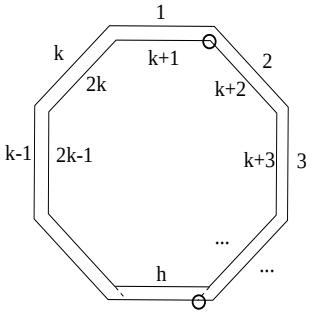

Figure 2.3: 2D-Cut of Cone

Drawback (both definitions): The combinatorial dimension can be $O(|H|)$; for such large $\dim(H, v)$ are the axiomatic framework and its algorithms useless.

2.3 Transformations Using a Combinatorial Bounding Box

In this section we use the concept of a bounding box in a combinatorial manner (in fact we use the idea of the second definition in subsection 2.2.2, but with a better function f). The first variant (subsection 2.3.1) follows an article of Seidel [9]: The box is of the form $-\lambda \leq x_i \leq \lambda$, $i \in \{1, \ldots, d\}$ where the parameter λ is considered for large values. We will implicitly add the constraints of such a bounding box to every set of constraints $G \subseteq H$, such that (in the case of feasibility) every optimal solution $x_\lambda(G)$ is finite (where $x_\lambda(G)$ is defined similarly as x^G in definition 2.1); we will see that for large values of λ the solutions are of the form $x_\lambda(G) = u + \lambda w$ for fixed $u, w \in \mathbb{R}^d$. We are able to compute u and w and use them for the definition of an optimization problem (H, v).

In subsection 2.3.2 we replace the λ-box by another combinatorial box, but the main ideas remain the same as described for the λ-box.

2.3.1 The λ-Box and Seidel's Algorithm

The origin of this subsection is the appendix of an article of Seidel [9].

We start with a linear programming problem of the form (1.1). Instead of adding a fix numerical bounding box as in subsection 2.2.1, we add a bounding box where the size is controlled by a parameter $\lambda \geq 0$:

$$\max c^T x, \text{ such that } Ax \leq b, \text{ for all } i \in \{1, \ldots, d\} : -\lambda \leq x_i \leq \lambda. \tag{2.2}$$

Since we like to have nonnegativity constraints for x, we introduce shifted coordinates $\tilde{x}_i := \frac{x_i + \lambda}{2}$, so we can write (2.2) in the form

$$\max c^T \tilde{x}, \text{ such that } \tilde{A}\tilde{x} \leq b + \lambda \tilde{b}, \text{ for all } i \in \{1, \ldots, d\} : 0 \leq \tilde{x}_i \leq \lambda,$$

for $\tilde{A}_{ji} = 2A_{ji}$, $\tilde{b}_j = \sum_{i=1}^d A_{ji}$; for the following we omit again the tilde (except for $\tilde{b}$):

$$\max c^T x, \text{ such that } Ax \leq b + \lambda \tilde{b}, \text{ for all } i \in \{1, \ldots, d\} : 0 \leq x_i \leq \lambda. \tag{2.3}$$

Definition 2.4 (Optimal Solution $x(\lambda)$ of LP with λ-box)
For a given (generalized) linear programming problem (LP) of the form (2.3) we write $(LP)\,|^\lambda$ for the same problem with fixed $\lambda \geq 0$: Then a feasible point $x(\lambda) \in \mathbb{R}^d$ is called the optimal solution of $(LP)\,|^\lambda$ *when it maximizes the vector $(c^T x, x_1, \ldots, x_d)$ with respect to the lexicographical order for all feasible points (of course we call a point x* feasible *when $Ax \leq b + \lambda \tilde{b}$ and for all i: $0 \leq x_i \leq \lambda$).*

We can describe the behaviour of the optimal solutions $x(\lambda)$ of $(LP)\,|^\lambda$ for large values of λ:

Theorem 2.5 *Either there exists $\lambda_0 \geq 0$, such that for all $\lambda \geq \lambda_0$ the problem $(LP)\,|^\lambda$ has no feasible solution, or there exist $\lambda_0 \geq 0$ and (unique) $u, w \in \mathbb{R}^d$, such that for all $\lambda \geq \lambda_0$ the problem $(LP)\,|^\lambda$ has the optimal solution $x(\lambda) = u + \lambda w$.*

Proof: Assume that there exist $0 \leq \lambda_1 < \lambda_2$ and $x^1, x^2 \in \mathbb{R}^d$, such that for $k = 1, 2$ the point x^k is feasible for $(LP)\,|^{\lambda_k}$, i.e. $Ax^k \leq b + \lambda_k \tilde{b}$ and for all $i \in \{1, \ldots, d\} : 0 \leq x_i^k \leq \lambda_k$. Then for any ϑ with $0 < \vartheta < 1$ we consider $\lambda = \vartheta\lambda_1 + (1-\vartheta)\lambda_2$ and $x = \vartheta x^1 + (1-\vartheta)x^2$, and remark that x is feasible for $(LP)\,|^\lambda$ ($Ax = \vartheta Ax^1 + (1-\vartheta)Ax^2 \leq \vartheta(b + \lambda_1\tilde{b}) + (1-\vartheta)(b + \lambda_2\tilde{b}) = b + \lambda\tilde{b}$ and for

all $i \in \{1,\dots,d\} : 0 \le \vartheta x_i^1 + (1-\vartheta)x_i^2 = x_i \le \vartheta\lambda_1 + (1-\vartheta)\lambda_2 = \lambda$); this implies: if there does not exist a $\lambda_0 \ge 0$, such that for all $\lambda \ge \lambda_0$ the problem $(LP)\mid^\lambda$ has no feasible solution, then there exists a $\lambda^0 \ge 0$ such that for all $\lambda \ge \lambda^0$ the problem $(LP)\mid^\lambda$ has a feasible solution. We assume now to be in this second case; it remains to show that (for large λ) the optimal solution of $(LP)\mid^\lambda$ is of the form $x(\lambda) = u + \lambda w$ for fix u, w.
For fixed λ is $x(\lambda)$ a vertex of the feasible region, i.e. the intersection of some d linearly independent λ-constraints, where we define a λ-constraint to be either one of the constraints given by $Ax \le b + \lambda\tilde{b}$ or one of the box constraints ($0 \le x_i$ or $x_i \le \lambda$); we collect all λ-constraints in one system of linear inequalities $A^\square x \le b^\square + \lambda\tilde{b}^\square$, where it is important to remark that $A^\square$, $b^\square$, and $\tilde{b}^\square$ are independent from λ. For any $\lambda \ge \lambda^0$ there exists a subset B_λ of d row indices such that x_λ is the only solution of $A^\square_{B_\lambda} x = b^\square_{B_\lambda} + \lambda\tilde{b}^\square_{B_\lambda}$, i.e. $x_\lambda = (A^\square_{B_\lambda})^{-1} b^\square_{B_\lambda} + \lambda(A^\square_{B_\lambda})^{-1}\tilde{b}^\square_{B_\lambda} \equiv u_\lambda + \lambda w_\lambda$. We have to show that there exists a $\lambda_0 \ge \lambda^0$ such that for all $\lambda \ge \lambda_0$ the vectors u_λ, w_λ are the same. We assume that this is not true. Since there are only finitely many sets B_λ, there are also only finitely many pairs (u_λ, w_λ) which occur for $\lambda \ge \lambda^0$; this implies that there are at least two *different* pairs, let us say (u_1, w_1) and (u_2, w_2), which occur infinitely often in alternation, i.e. there exists a sequence $\lambda^0 \le \lambda_1^1 < \lambda_2^1 < \lambda_1^2 < \lambda_2^2 < \dots$ with $\lambda_i^k \to \infty$ for $k \to \infty$ and $i \in \{1,2\}$ and with $(u_{\lambda_i^k}, w_{\lambda_i^k}) = (u_i, w_i)$ for all k and for $i \in \{1,2\}$. In the same way as at the beginning of this proof we see that $u_1 + \lambda_2^k w_1$ is feasible for $A^\square x \le b^\square + \lambda_2^k\tilde{b}^\square$ and $u_2 + \lambda_1^k w_2$ is feasible for $A^\square x \le b^\square + \lambda_1^k\tilde{b}^\square$ for every $k > 1$. Since $\lambda_i^k \to \infty$, the optimality of $u_i + \lambda_i^k w_i$ implies (with $j \in \{1,2\}, j \ne i$) $c^T(u_i + \lambda_i^k w_i) \ge c^T(u_j + \lambda_i^k w_j)$ for arbitrary large λ_i^k, i.e. with respect to the lexicographical order $(c^T w_i, c^T u_i) \ge (c^T w_j, c^T u_j)$ and, by symmetry in i and j, $c^T w_1 = c^T w_2$ and $c^T u_1 = c^T u_2$; then, in the same way: $u_i + \lambda_i^k w_i \ge u_j + \lambda_i^k w_j$, i.e. $(w_i, u_i) \ge (w_j, u_j)$, so (by symmetry) $w_1 = w_2, u_1 = u_2$, a contradiction. ■

In the following we present the algorithm of Seidel, which solves problems of the form (2.3). We do not discuss the algorithm **SeidelLP** in detail; we give the main ideas:

The algorithm incrementally adds one constraint after the other (in random order) and solves each time the linear program with respect to the current constraint set. Assume we have solved the problem with respect to the constraints $h_1, \dots, h_{i-1}$ with optimal solution x^{i-1}, the next constraint to add is h_i: First we check whether h_i is violated by x^{i-1}; if not, we return $x^i := x^{i-1}$ as the optimal solution for the constraint set $\{h_1, \dots, h_i\}$; otherwise the optimal solution x^i has to lie on h_i (here we have to use convexity and linearity arguments), so we project the linear program (i.e. c and the constraints $h_1, \dots, h_{i-1}$) on h_i and solve this problem in $d-1$ dimensions. The probability for a violation of x^{i-1} by h_i is at most $\frac{d}{n}$ (x^i is optimal solution of $(LP)\mid_B$, where B is a basis with $|B| = d$; if $h_i \notin B$, there is not violation). For the recursion this means: A call of **SeidelLP** will cause in general (i.e. when not $d = 1$ or $n = 0$) one call of **SeidelLP** with one constraint h_i removed, and with probability $\frac{d}{n}$ one further call of **SeidelLP** where the linear program is projected on h_i. For the expected run time $T(d,n)$ we have (for the original Seidel algorithm; **SeidelLP** has a term $T(d-1, n+1)$ instead of $T(d-1, n-1)$):

$$T(d,n) \le \begin{cases} O(n) & \text{if } d = 1, \\ O(d) & \text{if } n = 0, \\ T(d, n-1) + O(d) + \frac{d}{n}(T(d-1, n-1) + O(dn)) & \text{otherwise.} \end{cases}$$

This leads to an expected run time of $O(d!n)$ (for more explanation see [9]).

Algorithm SeidelLP:
Input: Dimension d, $d > 0$, $d \in \mathbb{Z}$; $c \in \mathbb{R}^d$; $b, \hat{b} \in \mathbb{R}^n$; $A \in \mathbb{R}^{n \times d}$, $A = (A_{i,j})_{\substack{i=1,\dots,n \\ j=1,\dots,d}}$.
Output: Either "infeasible" or $(w,u) \in \mathbb{R}^d \times \mathbb{R}^d$, such that $u + \lambda w$ is the optimal solution for large values of λ (see theorem 2.5).
Remark: We use for comparisons $(\le, \min, \max)$ the lexicographical order defined as in (2.1).

begin SeidelLP$(d, c, b, \hat{b}, A)$;
  if $d = 1$ **then**
    $(h_\lambda, h) := \min\left(\left\{(\frac{\hat{b}_i}{a_{i,1}}, \frac{b_i}{a_{i,1}}) | i \in \{1, \ldots, n\} : a_{i,1} > 0\right\} \cup \{(1,0)\}\right)$;
    $(\ell_\lambda, \ell) := \max\left(\left\{(\frac{\hat{b}_i}{a_{i,1}}, \frac{b_i}{a_{i,1}}) | i \in \{1, \ldots, n\} : a_{i,1} < 0\right\} \cup \{(-1,0)\}\right)$;
    $(z_\lambda, z) := \min\left(\left\{(\hat{b}_i, b_i) | i \in \{1, \ldots, n\} : a_{i,1} = 0\right\} \cup \{(0,0)\}\right)$;
    if $((z_\lambda, z) < (0,0)$ **or** $(h_\lambda, h) < (\ell_\lambda, \ell))$ **then return** "infeasible"
    else if $c \geq 0$ **then return** (h_λ, h) **else return** (ℓ_λ, ℓ) **endif**
    endif
  else
    if $A = \emptyset$ (i.e. $n = 0$) **then**
      for $j \in \{1, \ldots, d\}$ **do**
        if $c_j \geq 0$ **then** $w_j := +1$ **else** $w_j := -1$ **endif**;
      endfor;
      return $((w_1, \ldots, w_d), (0, \ldots, 0))$
    else
      choose $i \in \{1, \ldots, n\}$ at random;
      $A^i := A$ with row i removed;
      $b^i := b$ with component i removed; $\hat{b}^i := \hat{b}$ with component i removed;
      $x^i :=$ **SeidelLP**$(d, c, b^i, \hat{b}^i, A^i)$;
      if $x^i =$ "infeasible" **then return** "infeasible" **else** $(w, u) := x^i$ **endif**;
      if $(a_i^T w, a_i^T u) \leq (\hat{b}_i, b_i)$ **then return** (w, u)
      else
        if $\{j \in \{1, \ldots, d\} | A_{i,j} \neq 0\} = \emptyset$ **then return** "infeasible" **endif**;
        $k := \max\{j \in \{1, \ldots, d\} | A_{i,j} \neq 0\}$;
        $\bar{A}^i := (\bar{A}^i_{r,s})_{\substack{r=1,\ldots,n, r\neq i \\ s=1,\ldots,d, s\neq k}}$, where $\bar{A}^i_{r,s} := A_{r,s} - \frac{A_{r,k}}{A_{i,k}} A_{i,s}$;
        $\bar{c} := c - \frac{c_k}{A_{i,k}}(A_{i,1}, \ldots, A_{i,d})$ with component k removed;
        $\bar{A} := \bar{A}^i$ with additional rows $\frac{1}{A_{i,k}}(A_{i,1}, \ldots, A_{i,d})$ and $-\frac{1}{A_{i,k}}(A_{i,1}, \ldots, A_{i,d})$
          (both with component k removed) at the bottom of the matrix;
        $\bar{x}^i :=$ **SeidelLP**$(d-1, \bar{c}, b^i \circ (0,0), \hat{b}^i \circ (1,1), \bar{A})$;
        if $\bar{x}^i =$ "infeasible" **then return** "infeasible"
        else $((w_1, \ldots, w_{k-1}, w_{k+1}, \ldots, w_d), (u_1, \ldots, u_{k-1}, u_{k+1}, \ldots, u_d)) := \bar{x}^i$
        endif;
        return $((w_1, \ldots, w_d), (u_1, \ldots, u_d))$,
          where $w_k := \frac{1}{A_{i,k}}(\hat{b}_i - \sum_{s\neq k} A_{i,s} w_s)$ and $u_k := \frac{1}{A_{i,k}}(b_i - \sum_{s\neq k} A_{i,s} u_s)$
      endif
    endif
  endif
end SeidelLP.

For a given linear program (LP) we consider as usual the set of constraints H and for any $G \subseteq H$ the problem $(LP)\,|_G$, which will be transformed into a problem of the form (2.3) as described above; now we can define (H, v):

Definition 2.6 *For any $G \subseteq H$ we consider $(LP)\,|_G$: We call $u + \lambda w$* the optimal solution of

$(LP)\,|_G$, *when* $u + \lambda w$ *is optimal for large* λ *(cf. theorem 2.5). We define the value* $v(G)$ *by*

$$v(G) := \begin{cases} (-\infty) & \text{if } (LP)\,|_G \text{ is infeasible,} \\ (c^T u) \circ u & \text{if } u + \lambda \cdot 0 \text{ is the optimal solution of } (LP)\,|_G, \\ (+\infty) \circ w \circ u & \text{if } u + \lambda w, w \neq 0, \text{ is the optimal solution of } (LP)\,|_G. \end{cases}$$

Theorem 2.7 *For any linear programming problem specifies* (H, v) *an optimization problem of LP-type (where* v *is defined as in definition 2.6). The combinatorial dimension of* (H, v) *is bounded by* $\dim(H, v) \leq d + 1$ *(in the case of feasibility even* $\dim(H, v) \leq d$*).*

Proof: For any given linear programming problem (LP) we define the value function as in definition 2.6; then we have to show monotonicity, locality, and the bound for the combinatorial dimension.

- **Monotonicity:** Given F, G with $F \subseteq G \subseteq H$:
  - If $(LP)\,|_F$ is infeasible, then $(LP)\,|_G$ is infeasible, too, i.e. $v(F) = v(G) = (-\infty)$.
  - If $v(F) = (c^T u^F) \circ u^F$ (i.e. $w^F = 0$: the problem $(LP)\,|_F$ has a finite optimum), then $(LP)\,|_G$ is either infeasible (so $v(G) = (-\infty) < v(F)$), or has also a finite optimum, i.e. $v(G) = (c^T u^G) \circ u^G$ (then $(c^T u^G) \circ u^G \leq (c^T u^F) \circ u^F$, because u^G is feasible for $(LP)\,|_F$); obviously $(LP)\,|_G$ can not have an unbounded optimum (i.e. $w \neq 0$), since $(LP)\,|_F$ has not an unbounded optimum.
  - If $v(F) = (+\infty) \circ w^F \circ u^F$ (i.e. $w^F \neq 0$: the problem $(LP)\,|_F$ has an unbounded optimum), then $(LP)\,|_G$ is either infeasible or $v(G) = (c^T u^G) \circ u^G$ (both cases are trivial), or $v(G) = (+\infty) \circ w^G \circ u^G$ (then $w^G \circ u^G \leq w^F \circ u^F$, since (for any λ) every $u + \lambda w$ which is feasible for $(LP)\,|^{\lambda}_G$ is also feasible for $(LP)\,|^{\lambda}_F$).
- **Locality:** Given F, G with $F \subseteq G \subseteq H$, $v(F) = v(G)$, and a constraint $h \in H$, such that $v(G) > v(G \cup \{h\})$:
  - If $(LP)\,|_F$ would be infeasible, then $(LP)\,|_G$ would be infeasible, too; but then $(-\infty) = v(G) > v(G \cup \{h\}) \geq (-\infty)$, a contradiction.
  - If $v(F) = (c^T u^F) \circ u^F$, then (with $v(F) = v(G)$) u^F is the optimal solution of $(LP)\,|_G$, too. $v(G) > v(G \cup \{h\})$ implies, that u^F is not feasible for the constraint h, so $v(F) \neq v(F \cup \{h\})$, i.e. (with monotonicity) $v(F) > v(F \cup \{h\})$.
  - If $v(F) = (+\infty) \circ w^F \circ u^F$, then (for large λ) $u^F + \lambda w^F$ is the optimal solution of $(LP)\,|^{\lambda}_F$ and of $(LP)\,|^{\lambda}_G$. $v(G) > v(G \cup \{h\})$ implies, that (for large λ) $u^F + \lambda w^F$ is not feasible for the constraint h, so as above $v(F) > v(F \cup \{h\})$.
- **Dimension:** Let be B any basis:
  - If $(LP)\,|_B$ is infeasible, then (see e.g. in [1], theorem 9.4) there exists a subset $\tilde{B} \subseteq B$ of at most $d + 1$ constraints, such that $(LP)\,|_{\tilde{B}}$ is infeasible; the definition of a basis implies $B = \tilde{B}$, so $|B| \leq d + 1$.
  - If $v(B) = (c^T u^B) \circ u^B$, then the optimal solution u^B of $(LP)\,|_B$ is a vertex of the feasible region of $(LP)\,|_B$; this vertex can be defined as the optimal solution of $(LP)\,|_{\tilde{B}}$ for $\tilde{B} \subseteq B$ with $|\tilde{B}| = d$; in the same way as above follows $|B| = d$.
  - If $v(B) = (\infty) \circ w^B \circ u^B$: Consider $(LP)\,|^{\lambda}_B$ for large values of λ (i.e. $\lambda \geq \lambda_0$): The optimal solution of $(LP)\,|^{\lambda}_B$ is a vertex of the feasible region of $(LP)\,|^{\lambda}_B$, so we have a set $\tilde{B} \subseteq B \cup X$ of d constraints, where X is the set of the box constraints

$0 \leq x_i \leq \lambda$, such that $(LP)\mid_{\tilde{B}}$ and $(LP)\mid_{B \cup X}$ (regarded as linear programs without additional box constraints) have the same optimal solution. Then: $\tilde{B} \cap B$ has the desired property $|\tilde{B} \cap B| \leq d$ and $v(\tilde{B}) = v(B)$, i.e. we can conclude (since B is a basis) $\tilde{B} = B$ and $|B| \leq d$.

■

2.3.2 Lexicographical Bounding Box

In this subsection we refine the technique of the combinatorial bounding box: We use, in order to get nonnegativity constraints, a better transformation than shifting, and we change the form of the bounding box (for the advantage of this see the remark at the end of chapter 3).

Again we start with a linear programming problem of the form (1.1). Without loss of generality we assume that the matrix $A \in \mathbb{R}^{n \times d}$ has full rank (otherwise we can solve the problem in a proper subspace of $\mathbb{R}^d$), so we assume that A is of the form

$$A = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix},\ A_1 \in \mathbb{R}^{d \times d} \text{ a regular matrix},\ A_2 \in \mathbb{R}^{(n-d) \times d};$$

the given problem is then of the form

$$\max c^T x,\ \text{such that } A_1 x \leq b_1,\ A_2 x \leq b_2. \tag{2.4}$$

Because A_1 is regular, we can substitute $x = (A_1)^{-1}(b_1 - y)$ in (2.4) which leads to

$$\max(-c^T(A_1)^{-1})y,\ \text{such that } -A_2(A_1)^{-1}y \leq b_2 - A_2(A_1)^{-1}b_1,\ y \geq 0,\ y \in \mathbb{R}^d;$$

so we can assume that the given linear programming problem is of the form

$$\max c^T x,\ \text{such that } Ax \leq b,\ x \geq 0,\ x \in \mathbb{R}^d. \tag{2.5}$$

From now on we assume to have linear programming problems of the form (2.5) (instead of the previous form (1.1)), i.e. with nonnegativity constraints. Remark that the transformation has reduced the number of constraints in $Ax \leq b$ by d; nevertheless we will denote the (reduced) number of rows of the (new) matrix A by n as before.

Now we box a given problem of the form (2.5) by constraints of the form $0 \leq x_i \leq L^i$ (the L stands for "Lexicographical"; the i in L^i is an exponent and not only an index):

$$\max c^T x,\ \text{such that } Ax \leq b,\ \text{for all } i \in \{1, \ldots, d\} : 0 \leq x_i \leq L^i. \tag{2.6}$$

We call this box a *lexicographical bounding box* or in short terms a *L-box*.

In the same way as for the λ-box, we can give the following definitions and theorems:

Definition 2.8 (Optimal Solution $x(L)$ of LP with L-box)
For a given (generalized) linear programming problem (LP) of the form (2.6) we write $(LP)\mid^L$ for the same problem with fixed $L \geq 0$. A feasible point $x(L) \in \mathbb{R}^d$ is called the optimal solution of $(LP)\mid^L$ *when it maximizes the vector $(c^T x, x_1, \ldots, x_d)$ with respect to the lexicographical order for all feasible points (of course we call a point x* feasible *when $Ax \leq b$ and for all i: $0 \leq x_i \leq L^i$).*

Theorem 2.9 *Either the problem (2.6) has for all $L \geq 0$ no feasible solution, or there exist $L_0 \geq 0$ and (unique) $u, w^1, \ldots, w^d \in \mathbb{R}^d$, such that for all $L \geq L_0$ the problem $(LP)\mid^L$ has the optimal solution $x(L) = u + \sum_{i=1}^{d} L^i w^i$.*

Proof: Assume that there exists $L^* \geq 0$ and $x \in \mathbb{R}^d$ such that x is feasible for $(LP)\,|^{L^*}$. Then it is trivial that for any $\tilde{L} \geq L^*$ the point x is feasible for $(LP)\,|^{\tilde{L}}$; this implies: if not for all $L \geq 0$ the problem (2.6) has no feasible solution, then there exists a $L^* \geq 0$ such that for all $L \geq L^*$ the problem $(LP)\,|^L$ has a feasible solution. We assume now to be in this second case; it remains to show that (for large L) $x(L) = u + \sum_{i=1}^d L^i w^i$ for fix $u, w^1, \ldots, w^d$.
For fixed L is $x(L)$ a vertex of the feasible region, i.e. the intersection of some d linearly independent L-constraints, where we define a L-constraint to be either one of the constraints given by $Ax \leq b$ or one of the box constraints ($0 \leq x_i$ or $x_i \leq L^i$); we collect all L-constraints in one system of linear inequalities $A^\square x \leq b^\square + \sum_{i=1}^d L^i b^i$, where it is important to remark that $A^\square$, $b^\square$, and $b^1, \ldots b^d$ are independent from L. For any $L \geq L^*$ there exists a subset B_L of d row indices such that x_L is the only solution of $A^\square_{B_L} x = b^\square_{B_L} + \sum_{i=1}^d L^i b^i_{B_L}$, i.e. $x_L = (A^\square_{B_L})^{-1} b^\square_{B_L} + \sum_{i=1}^d L^i (A^\square_{B_L})^{-1} b^i_{B_L} \equiv u_L + \sum_{i=1}^d L^i w^i_L$. We have to show that there exists a $L_0 \geq L^*$ such that for all $L \geq L_0$ the vectors $u_L, w^1_L, \ldots, w^d_L$ stay unchanged. We assume that this is not true. Since there are only finitely many sets B_L, there are also only finitely many families of vectors $(u_L, w^1_L, \ldots, w^d_L)$ which occur for $L \geq L^*$; this implies that there are at least two *different* families, let us say $(u_1, w^1_1, \ldots, w^d_1)$ and $(u_2, w^1_2, \ldots, w^d_2)$, which occur infinitely often in alternation, i.e. there exists a sequence $L^* \leq L_{1,1} < L_{2,1} < L_{1,2} < L_{2,2} < \ldots$ with $L_{j,k} \to \infty$ for $k \to \infty$ and $j \in \{1,2\}$ and with $(u_{L_{j,k}}, w^1_{L_{j,k}}, \ldots, w^d_{L_{j,k}}) = (u_j, w^1_j, \ldots, w^d_j)$ for all k and for $j \in \{1,2\}$. We set for $j \in \{1,2\}$ and $L \geq L^*$: $x_j(L) := u_j + \sum_{i=1}^d L^i w^i_j$. Then consider any constraint $A^\square_s x \leq b^\square_s + \sum_{i=1}^d L^i b^i_s$ and the sequence $x_j(L_{j,k})$ for $k \to \infty$: When we substitute $x_j(L)$ for x in $A^\square_s x \leq b^\square_s + \sum_{i=1}^d L^i b^i_s$, we have at both sides of the inequality a polynomial in L; since $L_{j,k} \to \infty$ for $k \to \infty$ and $x_j(L_{j,k})$ is feasible for $A^\square_s x \leq b^\square_s + \sum_{i=1}^d L^i b^i_s$, there exists a L_{js} such that $x_j(L)$ is feasible for the constraint $A^\square_s x \leq b^\square_s + \sum_{i=1}^d L^i b^i_s$ for all $L \geq L_{js}$. Since there are only finitely many constraints in $A^\square x \leq b^\square + \sum_{i=1}^d L^i b^i$, we conclude that there exists a L_0 such that for all $L \geq L_0$ and for $j \in \{1,2\}$ the point $x_j(L)$ is feasible for all constraints in $A^\square x \leq b^\square + \sum_{i=1}^d L^i b^i$, i.e. the points $x_1(L)$ and $x_2(L)$ are feasible for $L \geq L_0$. For $j_1, j_2 \in \{1,2\}$, $j_1 \neq j_2$: Since $L_{j_1,k} \to \infty$, the optimality of $u_{j_1} + \sum_{i=1}^d L^i_{j_1,k} w^i_{j_1}$ implies $c^T(u_{j_1} + \sum_{i=1}^d L^i_{j_1,k} w^i_{j_1}) \geq c^T(u_{j_2} + \sum_{i=1}^d L^i_{j_1,k} w^i_{j_2})$ for arbitrary large $L_{j_1,k}$, i.e. with respect to the lexicographical order $(c^T w^d_{j_1}, \ldots, c^T w^1_{j_1}, c^T u_{j_1}) \geq (c^T w^d_{j_2}, \ldots, c^T w^1_{j_2}, c^T u_{j_2})$ and, by symmetry in j_1 and j_2, $c^T w^d_1 = c^T w^d_2$, $\ldots$, $c^T w^1_1 = c^T w^1_2$, and $c^T u_1 = c^T u_2$; then, in the same way: $u_{j_1} + \sum_{i=1}^d L^i_{j_1,k} w^i_{j_1} \geq u_{j_2} + \sum_{i=1}^d L^i_{j_1,k} w^i_{j_2}$, i.e. $(w^d_{j_1}, \ldots, w^1_{j_1}, u_{j_1}) \geq (w^d_{j_2}, \ldots, w^1_{j_2}, u_{j_2})$, so (by symmetry) $w^d_1 = w^d_2, \ldots, w^1_1 = w^1_2$, $u_1 = u_2$, a contradiction. ∎

Definition 2.10 *For a given (LP) of the form (2.5) and for any $G \subseteq H$, where H is the set of constraints in $Ax \leq b$, we consider $(LP)\,|_G$: We call $(LP)\,|_G$* infeasible *if the corresponding problem of the form (2.6) is infeasible for any L; similarly we call $u + \sum_{i=1}^d L^i w^i$* the optimal solution of $(LP)\,|_G$*, when for large L the corresponding problem $(LP)\,|^L_G$ of the form (2.6) has the optimal solution $u + \sum_{i=1}^d L^i w^i$ (cf. theorem 2.9). We define the value $v(G)$ by*

$$v(G) := \begin{cases} (-\infty) & \text{if } (LP)\,|_G \text{ is infeasible,} \\ (c^T u) \circ u & \text{if } u + \sum_{i=1}^d L^i \cdot 0 \text{ is the optimal solution of } (LP)\,|_G, \\ (+\infty) \circ w^d \circ \ldots \circ w^1 \circ u & \text{if } u + \sum_{i=1}^d L^i w^i, \text{ not all } w^i = 0, \text{ is the optimal solution of } (LP)\,|_G. \end{cases}$$

Theorem 2.11 *For any linear programming problem specifies (H, v) an optimization problem of LP-type (where v is defined as in definition 2.10). The combinatorial dimension of (H, v) is bounded by $\dim(H, v) \leq d + 1$ (in the case of feasibility even $\dim(H, v) \leq d$).*

Proof: Similar to the proof of theorem 2.7. ∎

Chapter 3

Pivot Algorithms

The goal of this chapter is to design pivot algorithms (Seidel and Matoušek, Sharir, Welzl) which use the technique of combinatorial bounding boxes (see section 2.3). We start with a section about dictionaries, a setting for linear programming problems in which we will describe our pivot algorithms. The second section brings the algorithm of Seidel in pivot form (with the λ-box), but we do not pursue this at large. Section 3.3 discusses the pivot form of the algorithm of Matoušek, Sharir, Welzl (with the L-box), where the analysis of chapter 1 is applied again.

3.1 Dictionaries

3.1.1 Definitions and Elementary Properties

We shortly present the notation of dictionaries and some elementary properies (without proofs); for more details see [4].

We will use the following matrix notation:

Definition 3.1 (Matrix Notation) *For two finite and nonempty sets I_R, I_C we consider a* matrix $M \in \mathbb{R}^{I_R \times I_C}$ *which consists of the* elements *(or* entries*)* M_{ij} *for $i \in I_R$, $j \in I_C$.*
For nonempty subsets $S_R \subseteq I_R$, $S_C \subseteq I_C$ we denote the submatrix *corresponding to the entries M_{ij} for $i \in S_R$, $j \in S_C$ by $M_{S_R S_C}$. We use the following abbreviations: $M_{S_R} := M_{S_R I_C}$ and $M_{\cdot S_C} := M_{I_R S_C}$. For index sets with only one element we omit the braces: $M_i := M_{\{i\}}$ is the row of M with index $i \in I_R$, $M_{\cdot j} := M_{\cdot \{j\}}$ is the column of M with index $j \in I_C$.*

In the next two definitions we introduce basis and dictionary:

Definition 3.2 (Basis and Nonbasis) *Given a matrix $M \in \mathbb{R}^{I_R \times I_C}$ of rank $|I_R|$.*
A basis (of M) *is a subset $B \subseteq I_C$, such that $|B| = |I_R|$ and $M_{\cdot B}$ is regular.*
A nonbasis (of M) *is a subset $N \subseteq I_C$, such that $B = I_C \setminus N$ is a basis. When we use in the same context B and N, then B is a basis and $N = I_C \setminus B$.*

We think that there can hardly be confusion of this definition of a basis and the definition in the context of optimization problems of LP-type in chapter 1: The context should clear up the situation.

Definition 3.3 (Dictionary of a Basis) *Given a matrix $M \in \mathbb{R}^{I_R \times I_C}$ of rank $|I_R|$ and a basis $B \subseteq I_C$. Then we call*

$$D \equiv D(B) := -(M_{.B})^{-1} M_{.N} \in \mathbb{R}^{B \times N}$$

the dictionary of B.

We introduce the null space $N(M)$ and the row space $R(M)$ of a matrix M:

$$\begin{aligned} N(M) &:= \{\xi \in \mathbb{R}^{I_C} | M\xi = 0\}, \\ R(M) &:= \{M^T \eta \in \mathbb{R}^{I_C} | \eta \in \mathbb{R}^{I_R}\}. \end{aligned}$$

As elementary properties we have the following relations:

Lemma 3.4 *Given $M \in \mathbb{R}^{I_R \times I_C}$ of rank $|I_R|$, B a basis of M, $D = D(B)$, $\xi \in \mathbb{R}^{I_C}$:*

$$\begin{aligned} \xi \in N(M) &\Leftrightarrow \xi_B = D\xi_N, \\ \xi \in R(M) &\Leftrightarrow \xi_N = -D^T \xi_B. \end{aligned}$$

The next definition is fundamental for the algorithms:

Definition 3.5 (Pivot Operation on (i,j)) *Given $M \in \mathbb{R}^{I_R \times I_C}$ of rank $|I_R|$, B a basis of M, $D = D(B)$, $i \in B$, $j \in N$, such that $D_{ij} \neq 0$:*
The replacement of B by $B \setminus \{i\} \cup \{j\}$ is called the pivot operation on (i,j).
(i,j) is called the pivot, *D_{ij} is called* the pivot element of the dictionary.

Lemma 3.6 *Given $M \in \mathbb{R}^{I_R \times I_C}$ of rank $|I_R|$, B a basis of M, $D = D(B)$, $i \in B$, $j \in N$, such that $D_{ij} \neq 0$: Then is $B \setminus \{i\} \cup \{j\}$ again a basis of M.*

If $D_{ij} = 0$, then it is not possible to pivot on (i,j).

In the next subsection we give formulae for the computation of $D(B \setminus \{i\} \cup \{j\})$ from $D(B)$.

3.1.2 Linear Programs in Dictionary Form

Since we like to discuss problems with λ-box and problems with L-box (see (2.3) and (2.6)), we consider problems of the form

$$\max c^T x, \text{ such that } Ax \leq b + \lambda \tilde{b}, \text{ for all } i \in \{1, \ldots, d\} : 0 \leq x_i \leq \lambda_i; \tag{3.1}$$

for problems with λ-box we can set $\lambda_i := \lambda$, for problems with L-box we set $\tilde{b} := 0$ and $\lambda_i := L^i$. In the second case we can transform (3.1) exactly into the notation of [4], for the first case (with $\tilde{b} \neq 0$) the dictionary has an additional column for λ.

We introduce new variables $y_1, \ldots, y_n$; together with the variables $x_1, \ldots, x_d$ we have a set of $d+n$ variables ξ_i: Via the correspondence

$$\xi_1 \equiv x_1, \ \ldots, \ \xi_d \equiv x_d, \ \xi_{d+1} \equiv y_1, \ \ldots, \ \xi_{d+n} \equiv y_n$$

we have two possibilities to denote the variables. Furthermore we set

$$X := \{1, \ldots, d\} \text{ and } Y := \{d+1, \ldots, d+n\}.$$

We can write now (3.1) in the form

$$\begin{aligned} \max c^T x, \quad \text{such that} \quad & Ax + y = b + \lambda\tilde{b}, \\ & \text{for all } i \in X : 0 \le x_i \le \lambda_i, \\ & \text{for all } i \in Y : 0 \le y_{i-d}. \end{aligned} \tag{3.2}$$

When we introduce $\xi_{d+n+1} \equiv z := c^T x$ and $\xi_0 \equiv 1$, $\xi_{-1} \equiv \lambda$, then we can write with the enlarged vector of variables $\xi = (\xi_{-1}, \xi_0, \ldots, \xi_{d+n+1})$ and with the matrix

$$M = \left(\begin{array}{c|c|c|c} -\tilde{b} & -b & A & \mathbb{1} \\ \hline 0 & 0 & -c & \end{array} \right) \in \mathbb{R}^{\{d+1,\ldots,d+n+1\} \times \{-1,\ldots,d+n+1\}}$$

the problem (3.2) in the form

$$\begin{aligned} \max z, \quad \text{such that} \quad & M\xi = 0, \\ & \text{for all } i \in X : 0 \le \xi_i \le \lambda_i, \\ & \text{for all } i \in Y : 0 \le \xi_i. \end{aligned} \tag{3.3}$$

We will usually identify a variable ξ_i and its index i, so we can write e.g. $B = Y \cup \{z\}$ instead of $B = Y \cup \{d+n+1\}$ or D_{zi} instead of $D_{d+n+1\,i}$. The only possible confusion could be caused by $\xi_0 \equiv 1$ and ξ_1, but this should never be a problem.

With lemma 3.4 we can write (3.3) for any basis B of M in *the dictionary form*:

$$\begin{aligned} \max z, \quad \text{such that} \quad & \xi_B = D\xi_N, \\ & \text{for all } i \in X : 0 \le \xi_i \le \lambda_i, \\ & \text{for all } i \in Y : 0 \le \xi_i. \end{aligned} \tag{3.4}$$

For $i \in B$ and $j \in N$ with $D_{ij} \neq 0$ we can compute the new dictionary $\tilde{D} = D(B \setminus \{i\} \cup \{j\})$ after a pivot operation on (i, j): We use the relation

$$\xi_i = D_i \xi_N = D_{ij}\xi_j + \sum_{\ell \in N \setminus \{j\}} D_{i\ell}\xi_\ell$$

and substitute

$$\xi_j = \frac{1}{D_{ij}} \left(\xi_i - \sum_{\ell \in N \setminus \{j\}} D_{i\ell}\xi_\ell \right);$$

when $\tilde{B} = B \setminus \{i\} \cup \{j\}$ denotes the new basis, this leads to

$$\begin{aligned} \tilde{D}_{ji} &= \frac{1}{D_{ij}}, \\ \tilde{D}_{j\ell} &= -\frac{D_{i\ell}}{D_{ij}} \text{ for } \ell \in \tilde{N} \setminus \{i\}, \\ \tilde{D}_{ki} &= \frac{D_{kj}}{D_{ij}} \text{ for } k \in (\tilde{B} \setminus \{j\}), \\ \tilde{D}_{k\ell} &= D_{k\ell} - \frac{D_{kj}D_{i\ell}}{D_{ij}} \text{ for } k \in \tilde{B} \setminus \{j\} \text{ and } \ell \in \tilde{N} \setminus \{i\}. \end{aligned}$$

The computation of $\tilde{D}$ with these formulae causes costs of $O(dn)$.

Example: We give a linear programming problem (cf. figure 3.1) in the form (1.1):

$$\begin{aligned} \max x_2, \text{ such that } -2x_1 + x_2 &\leq -2, \\ 2x_1 - x_2 &\leq 4, \\ -x_1 - x_2 &\leq -1. \end{aligned}$$

We will apply the technique of the λ-box; the converted problem of the form (2.3) then is

$$\begin{aligned} \max x_2, \text{ such that } -4x_1 + 2x_2 &\leq -2 - \lambda, \\ 4x_1 - 2x_2 &\leq 4 + \lambda, \\ -2x_1 - 2x_2 &\leq -1 - 2\lambda, \\ 0 \leq x_1 \leq \lambda, \quad 0 \leq x_2 &\leq \lambda; \end{aligned}$$

in dictionary form for the usual starting basis $B = Y \cup \{z\}$ we write

	λ	1	x_1	x_2
y_1	-1	-2	4	-2
y_2	1	4	-4	2
y_3	-2	-1	2	2
z	0	0	0	1

or, for another basis (after a pivot operation on (y_2, x_1)):

	λ	1	y_2	x_2
y_1	0	2	-1	0
x_1	$\frac{1}{4}$	1	$-\frac{1}{4}$	$\frac{1}{2}$
y_3	$-\frac{3}{2}$	1	$-\frac{1}{2}$	3
z	0	0	0	1

•

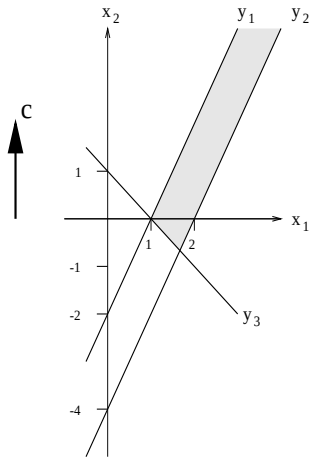

Figure 3.1: Linear Program in the previous example

Given a dictionary D, we usually like to identify it with a certain point $x(D) \in \mathbb{R}^d$, namely the intersection of the (binding) nonbasis constraints. Since a variable $i \in X$ stands for two

constraints, i.e. $0 \le x_i$ and $x_i \le \lambda_i$, we have to introduce a new feature for the dictionary which handles this information: For every $i \in N \cap X$ we will use a flag which has either the value "i-bound = lowerbound" or "i-bound = upperbound"; in order to have simpler notations we will also use flags "i-bound = lowerbound" for $i \in N \cap Y$. From now we assume that a dictionary is always given with such flags. Since the relation $\xi_B = D\xi_N$ determines the value of all variables whenever the nonbasis variables are determined, we can define:

Definition 3.7 ($\xi_i^=$; $x(D)$) *Given a dictionary $D = D(B)$, then we define for $i \in N$*

$$\xi_i^= := \begin{cases} 0 & \text{if } i \in X \cup Y \text{ with flag "}i\text{-bound = lowerbound"}, \\ \lambda_i & \text{if } i \in X \cup Y \text{ with flag "}i\text{-bound = upperbound"}, \\ 1 & \text{if } i = 0, \text{ i.e. } \xi_i \equiv 1, \\ \lambda & \text{if } i = -1, \text{ i.e. } \xi_i \equiv \lambda. \end{cases}$$

When we define now $\xi_B^= := D\xi_N^=$, then $x(D) := (x_1^=, \ldots, x_d^=)$.

The point $x(D)$ is well defined when $z \notin N$ (but this will always be the case: For our algorithms we will always have $\{-1, 0\} \subseteq N$ and $\{z\} \subseteq B$). We usually identify the dictionary D and the corresponding point $x(D)$.

3.1.3 Dual Dictionary Representation

In this subsection we present a technique which allows to compute a pivot operation in $O(d^2)$ instead of $O(dn)$: We will update not the dictionary itself but a representation matrix; each entry of the dictionary can be computed by this representation in $O(d)$.

Given a matrix $M \in \mathbb{R}^{I_R \times I_C}$ with rank $|I_R|$, basis B and nonbasis N, we like to find a matrix $C \in \mathbb{R}^{N \times I_C}$, such that

$$R(M) = N(C); \tag{3.5}$$

with lemma 3.4 we can write condition (3.5) in the equivalent form

$$\xi_N = -D^T \xi_B \Leftrightarrow \xi_{B_C} = D(B_C)\xi_{N_C}; \tag{3.6}$$

as suggested by (3.6) we set the basis B_C of C to be the nonbasis N of M and the nonbasis N_C of C to be the basis B of M.

When we have a basis B^*, such that $M_{\cdot B^*} = \mathbb{1}$, then we can set

$$C = (-(M_{\cdot N^*})^T | \mathbb{1}) \in \mathbb{R}^{N^* \times (B^* \cup N^*)}.$$

In fact, for the setting as in the previous subsection, we have a basis with this desired property, namely $B^* = Y \cup \{z\} \equiv \{d+1, \ldots, d+n+1\}$.

Once defined C, we can compute for any basis B of M the dictionary $D(B)$ via the relation

$$D^T = -D(B_C) = (D_{\cdot N})^{-1} C_{\cdot B},$$

i.e. for $i \in B$ and $j \in N$

$$D_{ij} = \left((C_{\cdot N})^{-1}\right)_j C_{\cdot i}.$$

At the beginning we will have $C_{.N} = \mathbb{1}$, but how can we update $C_{.N}$ efficiently in a pivot operation? When we set $R := (C_{.N})^{-1}$ and $\tilde{R} := (C_{.\tilde{N}})^{-1}$ for $\tilde{N} := N \cup \{i\} \setminus \{j\}$ (i.e. $\tilde{N}$ is the nonbasis after a pivot operation on (i,j)), then we have the relations

$$\begin{aligned} D_{k\ell} &= R_\ell C_{.k} \text{ for } k \in B, \ell \in N, \\ \tilde{D}_{k\ell} &= \tilde{R}_\ell C_{.k} \text{ for } k \in \tilde{B}, \ell \in \tilde{N}. \end{aligned}$$

By definition we know $RC_{.N} = \mathbb{1}$ and $\tilde{R}C_{.\tilde{N}} = \mathbb{1}$, so for $k \in N \setminus \{j\}$

$$RC_{.k} = \tilde{R}C_{.k} = \mathbb{1}_k, \text{ i.e. } (\tilde{R} - R)C_{.k} = 0. \tag{3.7}$$

Since $|N \setminus \{j\}| = d-1$, the matrix $\tilde{R} - R$ has rank 1, i.e. $\tilde{R} - R = uv^T$ for $u, v \in \mathbb{R}^N$, especially (3.7) implies that $v^T C_{.k} = 0$ for all $k \in N \setminus \{j\}$. As long as u is not determined, the scaling of v is free, i.e. we can set $v := (R_j)^T$. With this definition we have

$$\tilde{R} = R + uR_j, \tag{3.8}$$

which leads to the following equations:

$$\mathbb{1} = \tilde{R}C_{.\tilde{N}} = RC_{.\tilde{N}} + uR_jC_{.\tilde{N}},$$

$$uR_jC_{.\tilde{N}} = \mathbb{1} - RC_{.\tilde{N}}. \tag{3.9}$$

Since R and $C_{.\tilde{N}}$ are regular, we know that $w^T := R_jC_{.\tilde{N}}$ has at least one nonzero component, say w_k for $k \in \tilde{N}$, which can be computed in at most $O(d^2)$ time. Then (3.9) allows the computation of u in $O(d^2)$, namely

$$\begin{aligned} u_k &= \frac{1}{w_k}\left(1 - \sum_{t \in N} R_{kt}C_{tk}\right), \\ u_\ell &= -\frac{1}{w_k} \cdot \sum_{t \in N} R_{\ell t}C_{tk} \text{ for } \ell \in \tilde{N} \setminus \{k\}. \end{aligned}$$

It remains to compute $\tilde{R}$ by (3.8) in $O(d^2)$.

3.2 Seidel's Algorithm in Pivot Form

We present now **SeidelPivotLP**, a slightly modified version of **SeidelLP** in pivot form (the modification is: We can now choose an arbitrary k for the pivot (i,k) instead of $k := \max \ldots$); we will work with dictionaries as described above with the following additions:
As the algorithm of Seidel incrementally adds constraints, we mark one variable after another and solve the linear program every time with respect to the "marked" constraints. We will sometimes fix nonbasis variables; this fixed variables will stay nonbasis and are never pivoted into the basis as long as they are fixed.

Algorithm SeidelPivotLP:
Input: A dictionary D (including a x_i-flag "i-bound = lowerbound" or "i-bound = upperbound" for every $x_i \in N \cap (X \cup Y)$) with basis B and nonbasis N; the set $M \subseteq X \cup Y$ of the marked constraints; the set $F \cap (X \cup Y) \subseteq N \subseteq X \cup Y$ of the fixed variables.

Output: Either "infeasible" or a dictionary $\tilde{D}$ which is optimal (in the sense that $x(\tilde{D})$ is maximizing $(c^T x, x_1, \ldots, x_d)$ with respect to the lexicographical order) for the constraint set

$$\begin{aligned}
&\text{for all} \quad x_i \in F \cap X: \quad x_i = 0 \text{ or } x_i = \lambda \text{ (corresponding to } i\text{-bound)},\\
&\text{for all} \quad y_i \in F \cap Y: \quad y_i = 0,\\
&\text{for all} \quad x_i \in M \cap X: \quad 0 \le x_i \le \lambda,\\
&\text{for all} \quad y_i \in M \cap Y: \quad y_i \ge 0.
\end{aligned}$$

Remark: A pivot step changes the dictionary D (and by this also B and N), but the new dicionary is denoted by D, too. For comparisons (min, max, etc.) we use the lexicographical order.

begin SeidelPivotLP(D, M, F);
  if $|(N \cap (X \cup Y)) \setminus F| = 1$ **then**
    $\xi_j :=$ the only element in $(N \cap (X \cup Y)) \setminus F$; *(Remark: $\xi_j \in X$)*
    for all $i \in B \cap M$ **do** $\bar{b}_i^B := D_{i\lambda} + \sum_{\substack{x_k \in F \cap X \text{ with} \\ \text{“}k\text{-bound = upperbound”}}} D_{ik}$ **endfor**;
    $(h_\lambda, h) := \min\Big(\Big\{(-\frac{\bar{b}_i^B}{D_{ij}}, -\frac{D_{i0}}{D_{ij}}) | i \in B \cap M\text{: } D_{ij} < 0\Big\}$
      $\cup \Big\{(\frac{1-\bar{b}_i^B}{D_{ij}}, -\frac{D_{i0}}{D_{ij}}) | i \in B \cap M \cap X\text{: } D_{ij} > 0\Big\} \cup \{(1,0)\}\Big)$;
    $(\ell_\lambda, \ell) := \max\Big(\Big\{(-\frac{\bar{b}_i^B}{D_{ij}}, -\frac{D_{i0}}{D_{ij}}) | i \in B \cap M\text{: } D_{ij} > 0\Big\}$
      $\cup \Big\{(\frac{1-\bar{b}_i^B}{D_{ij}}, -\frac{D_{i0}}{D_{ij}}) | i \in B \cap M \cap X\text{: } D_{ij} < 0\Big\} \cup \{(0,0)\}\Big)$;
    $(z_\lambda, z) := \min\Big(\Big\{(\bar{b}_i^B, D_{i0}) | i \in B \cap M\text{: } D_{ij} = 0\Big\}$
      $\cup \Big\{(1 - \bar{b}_i^B, -D_{i0}) | i \in B \cap M \cap X\text{: } D_{ij} = 0\Big\} \cup \{(0,0)\}\Big)$;
    if $((z_\lambda, z) < (0,0)$ **or** $(h_\lambda, h) < (\ell_\lambda, \ell))$ **then return** "infeasible"
    else
      if $D_{zj} \neq 0$ **then** $s := z$
      else $s := \min(\{t \in B \cap X | D_{tj} \neq 0\} \cup \{j\})$;
      endif;
      if $(s = j$ **or** $D_{sj} > 0)$ **then** *(make ξ_j as large as possible)*
        if $(h_\lambda, h) = (1,0)$ **then** set flag "j-bound = upperbound"
        else
          pivot on (i,j), where ξ_i is binding for (h_λ, h);
          set ξ_i-flag corresponding to the violated contraint h_i; clear ξ_j-flag;
        endif
      else *(make ξ_j as small as possible)*
        if $(\ell_\lambda, \ell) = (0,0)$ **then** set flag "j-bound = lowerbound"
        else
          pivot on (i,j), where ξ_i is binding for (ℓ_λ, ℓ);
          set ξ_i-flag corresponding to the violated contraint h_i; clear ξ_j-flag;
        endif
      endif;
      return D
    endif
  else
    if $B \cap M = \emptyset$ **then**
      for all $j \in (N \cap (X \cup Y)) \setminus F$ **do** *(Remark: $(N \cap (X \cup Y)) \setminus F \subseteq X$)*
        if $D_{zj} \neq 0$ **then** $s := z$

```
                else s := min ({t ∈ B ∩ X | D_tj ≠ 0} ∪ {j});
                endif;
                if (s = j or D_sj > 0) then
                set flag "j-bound = upperbound"
                else set flag "j-bound = lowerbound"
                endif;
            endfor;
            return D
        else
            choose ξ_i ∈ B ∩ M at random;
            r^i := SeidelPivotLP(D, M \ {i}, F);
            if r^i = "infeasible" then return "infeasible" else D̃ := r^i endif;
            if (ξ_i ≡ D̃_i ξ_N ≥ 0 and (ξ_i ∈ X ⇒ ξ_i ≤ λ)) then return D̃
            else
                if D_i = 0 then return "infeasible" endif;
                choose k ∈ {j ∈ (N ∩ (X ∪ Y)) \ F | D_ij ≠ 0} (at random or with any criteria);
                pivot on (i, k);
                if ξ_i < 0 then set flag "i-bound = lowerbound"
                else (i.e. ξ_i > λ) set flag "i-bound = upperbound"
                endif;
                clear ξ_k-flag;
                return SeidelPivotLP(D, M, F ∪ {i})
            endif
        endif
    endif
end SeidelPivotLP.
```

The pivot form of Seidel's algorithm is not very efficient, since the dictionaries which are returned as the value of subcalls are not used but for a violation test; if the expected run time of $O(d!n)$ should be attained, the pivot operations should not be computed at once when they appear in **SeidelPivotLP** but only partially (i.e. compute an entry of the new dictionary not before you need it). We omit a detailed analysis and proceed to a more promising pivot algorithm.

3.3 The Algorithm of Matoušek, Sharir, Welzl in Pivot Form

We discuss in this section the pivot form of the algorithm of Matoušek, Sharir, Welzl (for the original algorithm see **BasisLP** in section 1.3). For the linear programs we use the notation of dictionaries (see section 3.1); since we will use the L-box as combinatorial bounding box (cf. subsection 2.3.2), we make no use of the variable $\xi_{-1} \equiv \lambda$, so we eliminate the corresponding column in M (see subsection 3.1.2).

In the first subsection we study the way, in which we can apply the techniques of chapter 1 and 2 (i.e. the axiomatic framework of LP-type problems and the concept of the L-box). The second subsection discusses the problem of the pivot search, i.e. the equivalent of the computation of **Basis**$(\tilde{F} \cup \{h\})$ in **BasisLP**. Subsection 3.3.3 finally contains the analysis of the algorithm.

3.3.1 Application of LP-Type Framework and Bounding Box

Even if the pivot variant of **BasisLP** will behave very similar to the original algorithm, we have to take care that we describe the relations between **BasisPivotLP** and **BasisLP** in a proper way. First we define a mapping from the (input) data of **BasisPivotLP** to the (input) data of **BasisLP**, then (after some additional notations) we describe how we will apply the L-box in **BasisPivotLP**.

The input of **BasisLP** consists of two sets of constraints G and F with $F \subseteq G \subseteq H$ and F a basis; the input of **BasisPivotLP** are a dictionary D and a set $M \subseteq X \cup Y$ of socalled *marked* constraints. The following definition defines a mapping from D and M to G and F:

Definition 3.8 ($G(M)$ and $F(D)$)
We define for a given dictionary D and a set $M \subseteq X \cup Y$ (M is called the set of the marked *constraints), where $N \cap (X \cup Y) \subseteq M$, two sets of constraints F and G as follows:*

- *$G = G(M)$ consists of the following constraints:*

$$\begin{aligned} &\text{for } i \in M \cap X: && 0 \le \xi_i \le L^i, \\ &\text{for } i \in M \cap Y: && 0 \le \xi_i. \end{aligned}$$

- *$F = F(D)$ consists of the following constraints:*

$$\begin{aligned} &\text{for } i \in N \cap (X \cup Y) \text{ with flag ``i-bound = lowerbound'':} && 0 \le \xi_i, \\ &\text{for } i \in N \cap (X \cup Y) \text{ with flag ``i-bound = upperbound'':} && \xi_i \le L^i. \end{aligned}$$

Remark 3.9 *The point $x(D)$ which is identified with the dictionary D (see definition 3.7)is exactly the intersection of the constraints in $F(D)$.*

Of course (i.e. because $N \cap (X \cup Y) \subseteq M$) we have always $F \subseteq G \subseteq H$, where H is the set of all constraints, i.e. $H = G(X \cup Y)$.

Remark that with definition 3.8 the $d+n$ variables in $X \cup Y$ represent $2d+n$ constraints, since every $i \in X$ stands for two constraints. The following definition determines (for the situations where we need it) for a $i \in X \cup Y$ a unique constraint h_i:

Definition 3.10 ($i \in X \cup Y$ violated by D; h_i)
Given a dictionary D with basis B, where $z \in B$, and any variable $i \in X \cup Y$.
The variable i is called violated by D, *when either $\xi_i^= < 0$ or $i \in X$ with $\xi_i^= > L^i$ for large L (for $\xi_i^=$ see definition 3.7).*
If i is violated by D, then we define h_i as follows:

$$h_i := \begin{cases} \xi_i \ge 0 & \text{if } \xi_i^= < 0, \\ \xi_i \le L^i & \text{otherwise.} \end{cases}$$

If $i \in N$ (and hence not violated by D), then we define h_i as follows:

$$h_i := \begin{cases} \xi_i \ge 0 & \text{if ``i-bound = lowerbound'',} \\ \xi_i \le L^i & \text{if ``i-bound = upperbound''.} \end{cases}$$

Now we describe how the axiomatic framework of chapter 1 and the L-box of chapter 2 will be used in the following:

We consider a given linear programming problem (LP) in the dictionary form (3.4), i.e. the problem is of the form (2.5). When we add the L-box constraints as in (2.6), we get a linear program $(LP)\ |^L$ with $2d+n$ constraints which is considered for $L \geq L_0$, where L_0 is as in theorem 2.9 (if (LP) is infeasible, then consider L_0 to be large enough, such that every point $x \geq 0$ is contained in the box $0 \leq x_i \leq L_0^i$, when the point is the intersection of d linearly independent constraints in $Ax \leq b$ and $x \geq 0$). We recall that we have not to know L_0: We will treat L not as a real number but as a parameter that represents an (unknown) real number which is larger than every real number which might occur in the computations. So let be $(LP)\ |^L$ a fixed linear program for the moment.

The following considerations might be somehow confusing, but they are necessary for our treatment of the problem: We have to define for $(LP)\ |^L$ a corresponding LP-type optimization problem (H, v). The confusing point is that we have used already a bounding L-box, but we will not take this box for the definition of (H, v) since we do not like to work with a constant bounding box $0 \leq x_i \leq L^i$ in the algorithm **BasisPivotLP**. In fact we define (H, v) by boxing $(LP)\ |^L$ with another (larger) bounding box $0 \leq x_i \leq \tilde{L}^i$: H is the set of the $2d+n$ constraints in $M(X \cup Y)$, v is defined as in definition 2.10 with $\tilde{L}$ instead of L (consider L as a parameter and $\tilde{L}_0 = \tilde{L}_0(L)$ depending on L); the vectors $u, w^1, \ldots, w^d$ depend on L.

In the algorithm **BasisPivotLP** we consider $(LP)\ |^L \equiv (LP)\ |_H^L$ and relaxed problems $(LP)\ |_S^L$ for $S \subseteq H$ (S will be a set like $F(D)$, $G(M)$, $G(M \setminus \{i\})$). It will be a key property of the algorithm that, when it is started with an appropriate dictionary D, then for all $S \subseteq H$ which occur in **BasisPivotLP** the problem $(LP)\ |_S^L$ is either infeasible or bounded. This means: The additional box $0 \leq x_i \leq \tilde{L}^i$ which was introduced for the definition of (H, v) does never occur in the computation, we can forget it again; even the form of the box is not relevant, a λ-box $0 \leq x_i \leq \tilde{\lambda}$ for large $\tilde{\lambda}$ leads to the same result (for the computation only the first bounding box and its form is important). We recall the meaning of the value $v(S)$ for a $S \subseteq H$ that can occur in **BasisPivotLP**:

- If $v(S) = (-\infty)$: $(LP)\ |_S^L$ is infeasible (and also (LP) will be infeasible).
- If $v(S) = (c^T u) \circ u$: $(LP)\ |_S^L$ is feasible and $u = u(L)$ is the optimal solution, i.e. the point $u(L)$ maximizes $(c^T x, x_1, \ldots, x_d)$ with respect to the lexicographical order (see subsection 2.1.1) for all points x which are feasible for all constraints in S.

In the algorithm the value $v(S) = (-\infty)$ is represented by "infeasible", the value $v(S) = (c^T u) \circ u$ has its equivalent in a dictionary D with $x(D) = u$. Hence the original problem (LP) has a bounded feasible solution if and only if the return value of **BasisPivotLP** is a dictionary D such that for all $i \in N \cap (X \cup Y)$ the flag is "i-bound = lowerbound".

3.3.2 Pivot Search in BasisPivotLP

In this subsection we study the following problem 3.11 (which corresponds to the computation of **Basis**$(\tilde{F} \cup \{h\})$ in the algorithm **BasisLP**); for the notation (as $(LP)\ |_{F(D)}^L$ of h_i) we refer to subsection 3.3.1.

Problem 3.11 (Pivot Search) *Given D and i: D a dictionary with basis B and nonbasis N, such that $x(D)$ is the optimal solution of $(LP)\ |_{F(D)}^L$ (i.e. $(LP)\ |_{F(D)}^L$ is bounded), and $i \in B \cap (X \cup Y)$ such that i is violated by D.*

Then: Determine whether $(LP)\ |^L_{F(D)\cup\{h_i\}}$ *is infeasible or not; if not, then find the (unique)* $j \in N \cap (X \cup Y)$ *such that* $x(\tilde{D})$ *is optimal for* $(LP)\ |^L_{F(\tilde{D})}$ *and feasible for* h_j*, where* $\tilde{D}$ *denotes the dictionary after a pivot operation on* (i,j)*; obviously* $x(\tilde{D})$ *is then optimal for* $(LP)\ |^L_{F(D)\cup\{h_i\}}$ *(this is a characterization of* j*: When after a pivot operation on* (i,j) $x(\tilde{D})$ *is optimal for* $(LP)\ |^L_{F(D)\cup\{h_i\}}$*, then is* j *the variable we want to find in the pivot search).*

We describe first a trivial (but unefficient) method which solves the pivot search:

When we have a criteria which decides for any dictionary D whether $x(D)$ is optimal for $(LP)\ |^L_{F(D)}$, then we can just compute for every $j \in N \cap (X \cup Y)$ the dictionary $\tilde{D}$ after a pivot operation on (i,j): If j is not violated by $\tilde{D}$ and $\tilde{D}$ is optimal for $(LP)\ |^L_{F(\tilde{D})}$ (here we need the criteria), then j is the answer for the pivot search. The costs then are $O(d) \cdot (T_{\text{PivotOperation}} + T_{\text{Criteria}})$. In fact we have such an optimality criteria:

Theorem 3.12 (Optimality Criteria for a Dictionary D**)** *Given a dictionary* D *with basis* B*; in order to make the criteria simpler we assume that for all* $k \in N \cap Y$ *the variable* ξ_k *is not constant, i.e. depends on the variables in* X *(it is easy to remove this assumption). For all* $k \in N \cap (X \cup Y)$ *we define*

$$s_k := \begin{cases} z & \text{if } D_{zk} \neq 0, \\ \min(\{t \in B \cap X | D_{tk} \neq 0\} \cup \{k\}) & \text{otherwise;} \end{cases}$$

then: $x(D)$ *is optimal for* $(LP)\ |^L_{F(D)}$ *if and only if for all* $k \in N \cap (X \cup Y)$

$$\begin{aligned} \text{when } s_k = k \text{ or } D_{s_k k} > 0: &\quad \text{“}k\text{-bound} = \text{upperbound”}, \\ \text{otherwise}: &\quad \text{“}k\text{-bound} = \text{lowerbound”}. \end{aligned}$$

Proof: Consider $\xi_B = D\xi_N$: We have to discuss the influence of a change of a nonbasis variable ξ_k on the vector $(c^T x, x_1, \ldots, x_d)$ which should be maximized. $x(D)$ is optimal for $(LP)\ |^L_{F(D)}$ if and only if for every $k \in N \cap (X \cup Y)$ a possible change of ξ_k (i.e. a change where ξ_k remains feasible for h_k) does not increase (and hence decrease) the vector $(c^T x, x_1, \ldots, x_d)$; for $k \in N \setminus (X \cup Y)$ is $\xi_k \equiv 1$ fix anyway. For any $k \in N \cap (X \cup Y)$ define s_k as above.

- If $s_k = z$: A change of ξ_k does also change the value of $z = c^T x$. We can not increase z if and only if the corresponding change of ξ_k is not possible, i.e. if

$$\begin{aligned} \text{when } D_{zk} > 0: &\quad \text{“}k\text{-bound} = \text{upperbound”}, \\ \text{when } D_{zk} < 0: &\quad \text{“}k\text{-bound} = \text{lowerbound”}. \end{aligned}$$

- If $s_k \neq z$: We have to find the x_s with $s =$ the minimal index such that a change of ξ_k does also change x_s:
  - If $k \notin X$, then $\{t \in B \cap X | D_{tk} \neq 0\} \neq \emptyset$ (otherwise ξ_k would be a constant; this case was excluded in the assumptions), and since the indices in X are smaller than the indices in Y we have

$$\min(\{t \in B \cap X | D_{tk} \neq 0\} \cup \{k\}) = \min\{t \in B \cap X | D_{tk} \neq 0\}$$

    and $s_k < k$; the rest follows in the same way as for $s_k = z$.

– If $k \in X$, then we have to remark that possibly the change of $\xi_k \equiv x_k$ itself is the most important influence on $(c^T x, x_1, \ldots, x_d)$ (hence we have to consider k when min... is computed); the rest is similar as we have seen it above.

■

We remark that this criteria can also be useful for the design of other pivot algorithms which use a L-box. Furthermore we see that for the optimality test of the dictionary it suffices to check the signs of dictionary elements.

An optimality test as in this theorem would cause costs of $O(d^2)$, when every dictionary element is available in constant time; the total costs of the pivot search in the described manner would be $O(d) \cdot (O(dn) + O(d^2)) = O(d^2 n)$; with the dual dictionary representation (see subsection 3.1.3) we have costs of $O(d) \cdot (O(d^2) + O(d^3)) = O(d^4)$.

We present in the following an algorithm which solves the pivot search by a (dual) Simplex step; the costs are in $O(d^2)$ even if each element of the dictionary has to be computed with costs of $O(d)$.

We consider a dictionary D and $i \in B$ as in problem 3.11: The infeasible case occurs if and only if for every $j \in N$ after the pivot step ξ_j is infeasible. Otherwise there exists at least one $j \in N$ for which after the pivot step on (i, j) the value of ξ_j is feasible (we call such a $j \in N$ *feasible*); then we have to find among the feasible $j \in N$ the one for which the vector $(c^T x, x_1, \ldots, x_d)$ is maximal.

We denote by $\tilde{\xi}_i^=$ the value of ξ_i after the pivot operation, i.e.

$$\tilde{\xi}_i^= := \begin{cases} 0 & \text{if } h_i \equiv \xi_i \geq 0, \\ L^i & \text{if } h_i \equiv \xi_i \leq L^i. \end{cases}$$

For the following we denote by $T_{\text{DictEntry}}$ the costs to compute an entry of the current dictionary (when we think of a dual dictionary representation as in subsection 3.1.3, then $T_{\text{DictEntry}}$ is in $O(d)$).

For any $j \in N$ the value of ξ_j after the pivot step on (i, j) is

$$\tilde{\xi}_j^= := \frac{1}{D_{ij}} \left(\tilde{\xi}_i^= - \sum_{\ell \in N \setminus \{j\}} D_{i\ell} \xi_\ell^= \right). \tag{3.10}$$

When we precompute (in $O(d) \cdot T_{\text{DictEntry}}$)

$$w_i := \tilde{\xi}_i^= - \sum_{\ell \in N} D_{i\ell} \xi_\ell^=,$$

then

$$\tilde{\xi}_j^= = \frac{1}{D_{ij}} \left(w_i + D_{ij} \xi_j^= \right) = \frac{w_i}{D_{ij}} + \xi_j^=$$

leads (in $O(1) \cdot T_{\text{DictEntry}}$ for every $j \in N$) with total costs of $O(d) \cdot T_{\text{DictEntry}}$ to the set of all feasible j.

We have now to discuss the effect of the choice of j on the vector $(c^T x, x_1, \ldots, x_d)$. For the objective value $c^T x$ we have after the pivot step on (i, j) (where $\tilde{\xi}_j^=$ is the (already known) value after the pivot step)

$$c^T x = z = D_{zj} \tilde{\xi}_j^= + \sum_{\ell \in N \setminus \{j\}} D_{z\ell} \xi_\ell^=.$$

When we precompute (in $O(d) \cdot T_{\text{DictEntry}}$)

$$w_z := \sum_{\ell \in N} D_{z\ell} \xi_\ell^=,$$

then, depending on the choice of $j \in N$:

$$z = D_{zj}\tilde{\xi}_j^= + w_z - D_{zj}\xi_j^= = w_z + D_{zj}(\tilde{\xi}_j^= - \xi_j^=) = w_z + \frac{D_{zj}}{D_{ij}} w_i;$$

so the total costs for collecting the set of the feasible $j \in N$ for which $c^T x$ is maximal are $O(d) \cdot T_{\text{DictEntry}}$. If this set contains more than one element, we compare the x_1-coordinates after the pivot step; if this comparison does not lead to a unique maximum, we go to the comparison of x_2 etc. For any $t \in \{1, \ldots, d\}$ we have to discuss the effect of the choice of j on x_t. The value of x_t after a pivot step on (i, j) is

$$x_t = \begin{cases} \xi_t^= & \text{if } t \in N \setminus \{j\}, \\ \tilde{\xi}_j^= & \text{if } t = j, \\ \tilde{\xi}_i^= & \text{if } t = i, \\ w_t + \frac{D_{tj}}{D_{ij}} w_i, \text{ where } w_t := \sum\limits_{\ell \in N} D_{t\ell} \xi_\ell^= & \text{otherwise.} \end{cases}$$

We will show that for every t the costs for the comparison of x_t are $O(1) \cdot T_{\text{DictEntryX}}$, where $T_{\text{DictEntryX}}$ are the costs to compute an entry D_{tj} of the dictionary with $t \in X$ ($T_{\text{DictEntryX}}$ will be in $O(1)$). When we have to compare $\tilde{d} \in \{0, 1, \ldots, d\}$ components of $(x_1, \ldots, x_d)$, the costs for the whole pivot search (without the computation of the new dictionary) are

$$O(d) \cdot T_{\text{DictEntry}} + \tilde{d} \cdot O(d) \cdot T_{\text{DictEntryX}} = O(d) \cdot T_{\text{DictEntry}} + O(d^2) \cdot T_{\text{DictEntryX}}.$$

The following algorithm solves the pivot search as we have described it so far:

Algorithm FindPivot1:
Input: A dictionary D with basis B and nonbasis N, such that $x(D)$ is the optimal solution of $(LP)\,|_{F(D)}$; $i \in B \cap (X \cup Y)$, such that i is violated by D.
Output: Either "infeasible" (if and only if $(LP)\,|_{F(D) \cup \{h_i\}}$ is infeasible) or $j \in N$, such that after a pivot step on (i, j) the new dictionary $\tilde{D}$ has the property: $x(\tilde{D})$ is optimal for $(LP)\,|_{F(\tilde{D})}$ and feasible for h_j.

begin FindPivot1(D, i);
  $w_i := \tilde{\xi}_i^= - \sum_{\ell \in N} D_{i\ell} \xi_\ell^=$;
  $S := \emptyset$;
  for all $j \in N$ with $D_{ij} \neq 0$ **do**
    $\tilde{\xi}_j^= := \frac{w_i}{D_{ij}} + \xi_j^=$;
    if $\tilde{\xi}_j^=$ is feasible for h_j **then** $S := S \cup \{j\}$ **endif**
  endfor;
  if $S = \emptyset$ **then return** "infeasible"
  else
    $q := \max\left\{\frac{D_{zj}}{D_{ij}} w_i | j \in S\right\}$;
    $S := \left\{j \in S | \frac{D_{zj}}{D_{ij}} w_i = q\right\}$;
    $t := 1$;

while $|S| > 1$ **do**
if $t \in S$ **then**
if $\tilde{\xi}_t^{=} > \xi_t^{=}$ **then return** t **else** *(remark* $\tilde{\xi}_t^{=} < \xi_t^{=}$*)* $S := S \setminus \{t\}$ **endif**
else
if $t \in B \setminus \{i\}$ **then**
$q := \max\left\{\frac{D_{tj}}{D_{ij}} w_i | j \in S\right\}$;
$S := \left\{j \in S | \frac{D_{tj}}{D_{ij}} w_i = q\right\}$
endif
endif;
$t := t + 1$
endwhile;
$j :=$ the only element in S;
return j
endif
end FindPivot1.

Some of the computations in **FindPivot1** are not necessary; we explain this in the following and present then a simpler version **FindPivot2**.

Instead of w_i we need only the sign of w_i: $\tilde{\xi}_j^{=}$ is feasible if

$$\begin{aligned} \text{when } \tilde{\xi}_j^{=} = 0: \quad & \frac{w_i}{D_{ij}} > 0, \\ \text{when } \tilde{\xi}_j^{=} = L^j: \quad & \frac{w_i}{D_{ij}} < 0; \end{aligned}$$

for the maximization $q := \max \ldots$ it is clear that w_i can be replaced by its sign.

Remark that the sign of w_i is already known from the violation test: $w_i = \tilde{\xi}_i^{=} - \xi_i^{=}$, so (since i is violated by D)

$$\begin{aligned} w_i > 0 \Leftrightarrow \quad & \xi_i^{=} < 0 \quad \Leftrightarrow \tilde{\xi}_i^{=} = 0, \\ w_i < 0 \Leftrightarrow \quad & \xi_i^{=} > L^i \quad \Leftrightarrow \tilde{\xi}_i^{=} = L^i. \end{aligned}$$

The test $\tilde{\xi}_t^{=} > \xi_t^{=}$ for $t \in S$ can be computed without $\tilde{\xi}_t^{=}$ by use of the following relations (remark that $\tilde{\xi}_t^{=}$ is feasible for h_t):

$$\begin{aligned} \tilde{\xi}_t^{=} > \xi_t^{=} \quad & \Leftrightarrow \quad \frac{w_i}{D_{it}} > 0 \Leftrightarrow \xi_t^{=} = 0 \\ & \Leftrightarrow \quad \text{“}t\text{-bound} = \text{lowerbound”}. \end{aligned}$$

With these simplifications the algorithm has the following form:

Algorithm FindPivot2:
Input: A dictionary D with basis B and nonbasis N, such that $x(D)$ is the optimal solution of $(LP)\,|_{F(D)}$; $i \in B \cap (X \cup Y)$, such that i is violated by D; the sign s_i of $\xi_i^{=}$ (either 1 or -1; remark that $s_i = -\text{sign}(w_i)$).
Output: Either “infeasible” (if and only if $(LP)\,|_{F(D)\cup\{h_i\}}$ is infeasible) or $j \in N$, such that after a pivot step on (i, j) the new dictionary $\tilde{D}$ has the property: $x(\tilde{D})$ is optimal for $(LP)\,|_{F(\tilde{D})}$ and feasible for h_j.

begin FindPivot2(D, i, s_i);
  $S := \emptyset$;
  for all $j \in N$ **do**
    compute D_{ij};
    if $D_{ij} \neq 0$ **and** $((\xi_j^{=} = 0$ **and** $\frac{s_i}{D_{ij}} < 0)$ **or** $(\xi_j^{=} = L^j$ **and** $\frac{s_i}{D_{ij}} > 0))$ **then**
      $S := S \cup \{j\}$
    endif
  endfor;
  if $S = \emptyset$ **then return** "infeasible"
  else
    $q := \min\left\{\frac{D_{zj}}{D_{ij}} s_i | j \in S\right\}$;
    $S := \left\{j \in S | \frac{D_{zj}}{D_{ij}} s_i = q\right\}$;
    $t := 1$;
    while $|S| > 1$ **do**
      if $t \in S$ **then**
        if "t-bound = lowerbound" **then return** t **else** $S := S \setminus \{t\}$ **endif**
      else
        if $t \in B \setminus \{i\}$ **then**
          $q := \min\left\{\frac{D_{tj}}{D_{ij}} s_i | j \in S\right\}$;
          $S := \left\{j \in S | \frac{D_{tj}}{D_{ij}} s_i = q\right\}$
        endif
      endif;
      $t := t + 1$
    endwhile;
    $j :=$ the only element in S;
    return j
  endif
end FindPivot2.

The costs of **FindPivot2** are the same as for **FindPivot1**, namely

$$O(d) \cdot T_{\text{DictEntry}} + O(d^2) \cdot T_{\text{DictEntryX}}.$$

For a direct use of a dictionary (where an update of the dictionary will cost $O(dn)$) we have of course $T_{\text{DictEntry}}$ and $T_{\text{DictEntryX}}$ in $O(1)$, so the total costs for the pivot search are in $O(d^2)$. For the dual representation of the dictionary (see subsection 3.1.3) we have $T_{\text{DictEntry}}$ in $O(d)$ and $T_{\text{DictEntryX}}$ in $O(1)$; the latter is true since for $t \in X \cap B$ we have

$$D_{tj} = \left((C_{\cdot N})^{-1}\right)_j C_{\cdot t} = \left((C_{\cdot N})^{-1}\right)_{jt} \equiv R_{jt}.$$

By this we have even for the dual representation technique costs of only $O(d^2)$.

3.3.3 Analysis of BasisPivotLP

We have seen in subsection 3.3.1 the main ideas how to map the pivot form of the algorithm of Matoušek, Sharir, Welzl into the framework of LP-type problems; there we have an analysis (see section 1.3) which we like to apply also for the pivot form. We will show that by the definition

of (H, v) and the mapping from D and M to G and F as in subsection 3.3.1 we have a relation between the two algorithms such that their behaviour is almost identical.

First we present the algorithm:

Algorithm BasisPivotLP:
Input: Dictionary D with basis B and nonbasis N; the set $M \subseteq X \cup Y$ of the marked constraints with $N \cap (X \cup Y) \subseteq M$; furthermore there is given the algorithm **FindPivot2** (see subsection 3.3.2).
Output: Either "infeasible" (if and only if $(LP)\mid^L_{G(M)}$ is infeasible) or a dictionary D which is optimal for M (this means: For large values of L is $x(D)$ the optimal solution of $(LP)\mid^L_{G(M)}$ and also the optimal solution of $(LP)\mid^L_{F(D)}$).

begin BasisPivotLP(D, M);
  if $B \cap M = \emptyset$ **then return** D
  else
    choose $i \in B \cap M$ at random;
    $r^i :=$ **BasisPivotLP**$(D, M \setminus \{i\})$;
    if $r^i =$ "infeasible" **then return** "infeasible" **else** $D := r^i$ **endif**;
    if i not violated by D **then return** D
    else
      if $h_i \equiv \xi_i \geq 0$ **then** $s_i := -1$ **else** $s_i := 1$;
      $j :=$ **FindPivot2**(D, i, s_i);
      if $j =$ "infeasible" **then return** "infeasible"
      else
        pivot on (i, j);
        if $h_i \equiv \xi_i \geq 0$ **then** set flag "i-bound = lowerbound"
        else set flag "i-bound = upperbound"
        endif;
        clear ξ_j-flag;
        return BasisPivotLP(D, M)
      endif
    endif
  endif
end BasisPivotLP.

Before we discuss the analysis in detail (theorem 3.13), we give some remarks about the behaviour of **BasisPivotLP**:

We start **BasisPivotLP** with D and M such that $x(D)$ is the optimal solution of $(LP)\mid_{F(D)}$ (i.e. the constraint set $F(D)$ defines a feasible region which is bounded with respect to the maximization of $(c^T x, x_1, \ldots, x_d)$), and $N \cap (X \cup Y) \subseteq M$. Then all further calls of **BasisPivotLP**$(\bar{D}, \bar{M})$ have again the property, that $(LP)\mid_{F(\bar{D})}$ is bounded (with optimal solution $x(\bar{D})$) and also that $\bar{N} \cap (X \cup Y) \subseteq \bar{M}$.

When we choose $i \in B \cap M$, this corresponds to the choice of $h \in G \setminus F$ in **BasisLP**; the difference is that for $i \in X$ we choose two constraints at once (namely the constraints $0 \leq \xi_i$ and $\xi_i \leq L^i$), but for at most one of these two can $x(D)$ be infeasible (this will be the constraint h_i).

Theorem 3.13 (Analysis of BasisPivotLP) *For a dictionary D and $M \subseteq X \cup Y$, where $x(D)$ is the optimal solution of $(LP)\mid_{F(D)}$ and where $(N \cap (X \cup Y)) \subseteq M$:*

(i) **BasisPivotLP**(D, M) *terminates after at most* $2^{|G(M)|-|\mathcal{E}(G(M),F(D))|+1} - 2$ *further calls of* **BasisPivotLP** *and returns either "infeasible" (if and only if* $(LP)\,|_{G(M)}$ *is infeasible) or a dictionary which is optimal for* M*.*

(ii) **BasisPivotLP**(D, M) *has an expected run time of at most*

$$e^{4\cdot\sqrt{d\ln(n+1)}} \cdot (O(d^2) + T_{Pivot\ Operation} + O(n) \cdot T_{Violation\ Test}),$$

where $d = |X|$ *and* $n = |Y|$*.*

Proof:

(i) We use the mentioned mapping from **BasisPivotLP**(D, M) to **BasisLP**$(G(M), F(D))$. We proof the finiteness (i.e. the upper bound for the number of subcalls) by induction over $m_{D,M} := |G(M)| - |\mathcal{E}(G(M), F(D))|$ (as in theorem 1.13 (iii)): If there are no subcalls (i.e. we are in the case $B \cap M = \emptyset$), then the claim is trivial (since we have always $m_{D,M} \geq 0$). If there are subcalls, we have to show that the inequality $m_{\bar{D},\bar{M}} \leq m_{D,M} - 1$ holds for any possible arguments $\bar{D}, \bar{M}$ of a subcall of **BasisPivotLP**.
For a subcall of **BasisPivotLP**$(D, M \setminus \{i\})$ we have $|G(M \setminus \{i\})| = |G(M)| - \ell$ for $\ell \in \{1, 2\}$, and with $\mathcal{E}(G(M), F(D)) \subseteq \mathcal{E}(G(M \setminus \{i\}), F(D))$ (cf. proof of theorem 1.13 (ii)) we conclude

$$\begin{aligned} m_{D,M\setminus\{i\}} &\leq |G(M)| - 1 - |\mathcal{E}(G(M \setminus \{i\}), F(D))| \\ &\leq |G(M)| - |\mathcal{E}(G(M), F(D))| - 1 = m_{D,M} - 1. \end{aligned}$$

For a possible subcall of **BasisPivotLP**$(\tilde{D}, M)$, where $\tilde{D}$ denotes the dictionary after a pivot step on (i, j), we remark that $F(\tilde{D})$ is an optimal basis of $F(D) \cup \{h_i\}$, where $D := r^i$ and $F(D)$ is an optimal basis of $G(M \setminus \{i\})$ and of $G(M) \setminus \{h_i\}$ for $h_i \in G(M) \setminus F(D)$; furthermore h_i is violated by $F(D)$, so we can conclude $m_{\tilde{D},M} \leq m_{D,M} - 1$ in the same way as we did it for the proof of inequality (1.5).
Correctness of **BasisPivotLP**: If $B \cap M = \emptyset$, then since $x(D)$ is optimal for $(LP)\,|_{F(D)}$ and there is obviously no constraint in $G(M) \setminus F(D)$ which is violated by $x(D)$, $x(D)$ is also optimal for $(LP)\,|_{G(M)}$. If $B \cap M \neq \emptyset$ we compute r^i. If $r^i =$ "infeasible", then $(LP)\,|_{G(M\setminus\{i\})}$ is infeasible and hence also $(LP)\,|_{G(M)}$, so we can return "infeasible". Otherwise the return value is "infeasible" if we detect in the pivot search that $(LP)\,|_{F(D)\cup\{h_i\}}$ for $D := r^i$ is infeasible (and then in fact $(LP)\,|_{G(M)}$ is infeasible, too), or we return **BasisPivotLP**$(\tilde{D}, M)$ (here the correctness follows from induction in the recursion tree).

(ii) We denote with $\beta_p(\ell, r)$ the maximal expected number of calls of **BasisPivotLP** caused by a call of **BasisPivotLP**(D, M) which occur immediately after a pivot step (i.e. we count all "second" calls from the same node in the recursion tree), where the maximum is taken over all D, M where $x(D)$ is optimal solution of $(LP)\,|_{F(D)}$, $N \subseteq M$, $r = |B \cap M|$, $d - \ell \leq |\mathcal{E}(G(M), F(D))|$ (if there are no such D and M for given ℓ and r, then we set $\beta_p(\ell, r) := 0$). With the same mapping technique as in (i) and in the same way as in theorem 1.15 we can show:

- For ℓ, r with $0 \leq \ell \leq d$, $0 \leq r \leq n$: $\beta_p(\ell, 0) = \beta_p(0, r) = 0$:
  If $r = 0$, we are in the case $B \cap M = \emptyset$, so $\beta_p(\ell, 0) = 0$ is trivial.
  If $\ell = 0$, we conclude $\mathcal{E}(G(M), F(D)) = F(D)$ (by lemma 1.11 (i) we have the inclusion $\mathcal{E}(G(M), F(D)) \subseteq F(D)$; furthermore remark $|F(D)| = d$). If $(LP)\,|_{G(M)}$ is feasible, then $F(D)$ is an optimal basis of $G(M)$ (by lemma 1.11 (ii) and since $G(M)$ has at most d extreme constraints, cf. lemma 1.8 (i)); we easily see that we have

never a subcall of "second type" (i.e. after a pivot step). If $(LP)\mid_{G(M)}$ is infeasible, then either $(LP)\mid_{(G(M\setminus\{i\}))}$ is infeasible (then we proof by induction in the recursion tree that there is no further "second type" call), or $(LP)\mid_{(G(M\setminus\{i\}))}$ is feasible and has the same optimal solution as $(LP)\mid_{F(D)}$ (where $D := r^i$); then $(LP)\mid_{F(D)\cup\{h_i\}}$ has to be infeasible (in detail: Assume that $(LP)\mid_{F(D)\cup\{h_i\}}$ is feasible. Then we can find $F_x \subseteq F(D)\cup\{h_i\}$ with $|F_x| = d$, such that $(LP)\mid_{F_x}$ and $(LP)\mid_{F(D)\cup\{h_i\}}$ have the same solution: F_x is an optimal basis of $F(D)\cup\{h_i\}$. We recall that $F(D)$ is an optimal basis of $G(M)\setminus\{h_i\}$, and that h_i is extreme in $G(M)$, so lemma 1.12 (iii) implies $|\mathcal{E}(G(M), F_x)| > d$, a contradiction to $|F_x| = d$ and lemma 1.11 (i)), so we have no further call.

- For ℓ, r with $1 \le \ell \le d$, $1 \le r \le n$: $\beta_p(\ell, r) \le \beta_p(\ell, r-1) + \frac{1}{r}\sum_{i=1}^{\min\{\ell,r\}}(1+\beta_p(\ell-i, r))$:
  If $(LP)\mid_{G(M)}$ is feasible: In the same way as in theorem 1.15 (ii) we consider the r possibilities to choose $i \in B\cap M$. There are $s \le \min\{\ell, r\}$ extreme constraints in $G(M)$ which are not in $F(D)$; just remark that for every $i \in B\cap M$ there is at most one constraint in $G(M)\setminus G(M\setminus\{i\})$ which is extreme in $G(M)$, and follow the arguments in theorem 1.15 (ii). Since the arguments of a "second type" call of **BasisPivotLP** match (after the mapping to $G(M)$ and $F(D)$) possible arguments of a "second type" call of **BasisLP** (see (i)), we can conclude $|\mathcal{E}(G(M), F(D))| + t \le |\mathcal{E}(G(M), F(\tilde{D}))|$ for $t \in \{1, \ldots, s\}$ and, by putting everything together, the inequality of the claim follows.
  If $(LP)\mid_{G(M)}$ is feasible, then all arguments remain true, only we have to remark that $s = \ell + 1$ is possible (but then the only additional case $t = \ell + 1$ does not lead to a call of **BasisPivotLP** because we detect infeasibility when we search for a pivot; compare with the detailed argumentation at the end of the proof of $\beta_p(0, r) = 0$).

This two properties of β_p imply as in theorem 1.15 (iv) (consider $\tilde{\beta}_p(\ell, r) := \beta_p(\ell, r) + 1$):

$$\text{For } \ell, r \text{ with } 0 \le \ell \le d,\ 0 \le r \le n: \quad \beta_p(\ell, r) \le e^{4\cdot\sqrt{\ell \ln(r+1)}} - 1.$$

As in theorem 1.15 (v) we can bound the expected run time of the algorithm with the use of $\beta_p(\ell, r)$. Since we have at most once a pivot search without a following call of **BasisPivotLP** (because then we detect infeasibility), we have for the expected number of pivot searches an upper bound of $e^{4\cdot\sqrt{\ell \ln(r+1)}}$, where every pivot search has costs of $O(d^2)$ and is followed (if not infeasibility is detected) by a pivot operation; for the expected number violation tests we have an upper bound of $r\cdot(e^{4\cdot\sqrt{\ell \ln(r+1)}}+1)$, and for the remaining arithmetics we have expected costs of at most $O(r\cdot e^{4\cdot\sqrt{\ell \ln(r+1)}})$. By this follows the claim of (ii).

■

We give a short discussion of the run time of **BasisPivotLP** depending on the representation of the dictionary:

When we use a direct dictionary representation and for the pivot operation the formulae of subsection 3.1.2, then we know that $T_{\text{Pivot Operation}}$ is in $O(dn)$ and $T_{\text{Violation Test}}$ is in $O(d)$, so the expected run time of **BasisPivotLP** is at most

$$e^{4\cdot\sqrt{d \ln(n+1)}} \cdot O(dn).$$

When we use the dual dictionary representation of subsection 3.1.3, then is $T_{\text{Pivot Operation}}$ in $O(d^2)$; the expected run time of **BasisPivotLP** is at most

$$e^{4\cdot\sqrt{d \ln(n+1)}} \cdot (O(d^2) + O(n) \cdot T_{\text{Violation Test}}).$$

$T_{\text{Violation Test}}$ causes at least costs of $O(d)$ and at most costs of $O(d^2)$: If we do not want to have a worse order for the run time than $e^{4\cdot\sqrt{d\ln(n+1)}}\cdot O(dn)$, we should have $T_{\text{Violation Test}}$ in $O(d)$ (for $T_{\text{Violation Test}}$ in $O(d^2)$ we have an expected run time of at most $e^{4\cdot\sqrt{d\ln(n+1)}}\cdot(O(d^2n))$).

How do we perform a violation test when we use the dual dictionary representation? We have to test, whether

$$\xi_i^= = D_i\xi_N^= = \sum_{k\in N} D_{ik}\xi_k^= \tag{3.11}$$

violates $\xi_i \geq 0$ or (only when $i \in X$) $\xi_i \leq L^i$. For the lexicographical box this can be done as follows (without computation of $\xi_i^=$):

Consider $S := \{k \in N | D_{ik}\xi_k^= \neq 0\}$; if $S = \emptyset$, then $\xi_i^= = 0$, i.e. is feasible; when $S \neq \emptyset$, then with $t := \max S$: The dominating term $D_{it}\xi_t^=$ determines the sign of $\xi_i^=$:

$$\text{sign}(\xi_i^=) = \text{sign}(D_{it}\xi_t^=) = \text{sign}(D_{it}).$$

In the worst case there are $O(d)$ indices $k \in N$ with $\xi_k^= > 0$ and $D_{ik} = 0$, so we could have total costs of $O(d^2)$ for the violation test; in a normal case we have only to compute one D_{ik} with costs of $O(d)$.

This leads to the following result:

The use of the dual dictionary representation gives (with our analysis) a higher subexponential bound of

$$O(d^2n\cdot e^{4\cdot\sqrt{d\ln(n+1)}})$$

instead of

$$O(dn\cdot e^{4\cdot\sqrt{d\ln(n+1)}}).$$

In practice we do not expect a slower performance: The faster computation of the dictionary in $O(d^2)$ instead of $O(dn)$ should be much more important than the rare cases where we need to compute a violation test with more than one dictionary element. Our implementation (see appendix A) will use the dual dictionary representation.

A last remark to the computational advantage of the L-box compared with the λ-box: It is obvious that the computation of a violation test is simpler for the L-box (see above); with the λ-box we have to compute the sum in (3.11) since there is no dominating term. An other advantage lies in the behaviour of the algorithm: An affine transformation of the linear program in $\mathbb{R}^d$ does not affect the combinatorial structure of the constraints (i.e. the given constraints of the LP together with the box constraints have the same adjacency relations); this is not true for the λ-box with its square form.

Chapter 4

Conclusions

4.1 Experimental Results

In this section we report some experiences which we have made with our implementation of the algorithm **BasisPivotLP** (for the implementation in C see appendix A).

In the following we consider a test serie of 50 examples; each example is a linear program (a dual Kuhn-Quandt problem) of dimension d with n constraints (plus d nonnegativity constraints), where $d \in \{10, 20, \ldots, 50\}$ and $n \in \{100, 200, \ldots, 1000\}$. We have used this test set for the comparison of the following three algorithms:

- **BasisPivotLP:** Here each example was solved with 50 or 100 different seeds for the random number generator. This led for each example to a mean number of pivot operations, a standard deviation for this number and a mean run time of the algorithm.
- **Dual Simplex:** Here we solved each example once and counted the number of pivot operations for this.
- **Criss-Cross:** Again only one run of the algorithm per example (with random strategy) and its number of pivot operations was considered.

For the Simplex and the Criss-Cross tests we used an implementation of Komei Fukuda (the C program `cdd`; see [3]). For the Criss-Cross algorithm see e.g. [2].

The following plots show a comparison of the number of pivot operations. For each algorithm we have two plots: At the left a plot where the number of constraints n is on the x-axis and with a line for each dimension d; the plot at the right contains the same information with n and d interchanged. The y-axis is scaled logarithmically. For comparison we give a plot with the subexponential upper bound $e^{4 \cdot \sqrt{d \ln(n+1)}}$ for the expected number of pivot operations.

Our tests do not allow to give a detailed comment about the practical behavior of the algorithms; but we can state the main tendencies. Remark that the scaling of the y-axis is different; we give in the following table the approximate range of the number of pivots for each algorithm and the whole test set:

	Minimum	Maximum
Dual Simplex (figure 4.2)	20	300
BasisPivotLP (figure 4.1)	100	6000
Criss-Cross (figure 4.3)	300	750000
Subexponential Bound (figure 4.4)	$6 \cdot 10^{11}$	$2 \cdot 10^{32}$

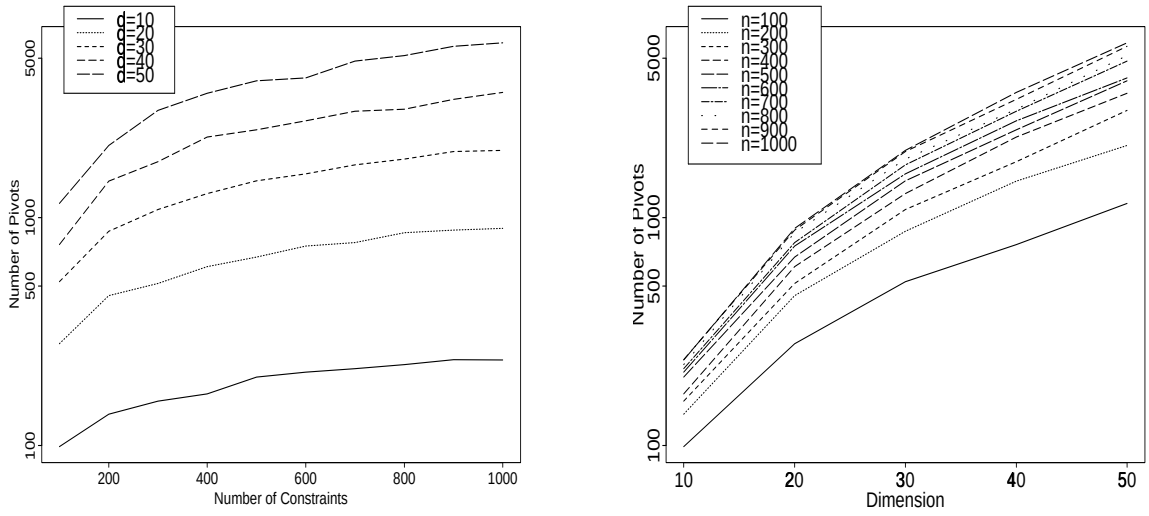

Figure 4.1: Number of Pivots with **BasisPivotLP**

For the number of pivot operations we seem to have the following relation:

$$\text{Dual Simplex} < \textbf{BasisPivotLP} < \text{Criss-Cross} \ll \text{Subexponential Bound}.$$

In figures 4.5 and 4.6 we show for **BasisPivotLP** the measured standard deviation and run time. For our examples the standard deviation is rather small, i.e. the number of pivot operations is more or less stable. The run time measurements are not very precise, the machine influences are quite big (the machine was a HP 9000 735/125).

Finally figure 4.7 shows the ratio of run time per pivot operation; in terms of the analysis this is at most $d \cdot n$. For a suggestion of the ratio in practice are our tests not powerful enough.

4.2 Questions and Suggestions for the Further Research

As we have seen in the previous section there is an enormous gap between the analysis of **BasisPivotLP** and its behaviour (i.e. computational costs) in practice; we think that this is not only caused by a particular test set of linear programs which are "easy to be solved". It seems that the framework of LP-type problems is not powerful enough for the design and analysis of algorithms which are "fast" and where the practical behaviour is near to the analysis.

In the following we suggest directions for the further research; especially we consider the connections to the framework and algorithms in this diploma thesis:

- **Axiomatics for linear programming:**
  The aim is to define and investigate structures (as e.g. here enforcing constraints, the combinatorial dimension) in axiomatic frameworks where linear programming is contained as a special case. Possibly one could take the LP-type framework as a starting point and try to modify or enlarge it, as e.g.
  - Modify the axiom of locality such that it is "nearer" to linear programming (we think that this is rather difficult; anyway it would seriously affect the whole analysis which was discussed it in this thesis)

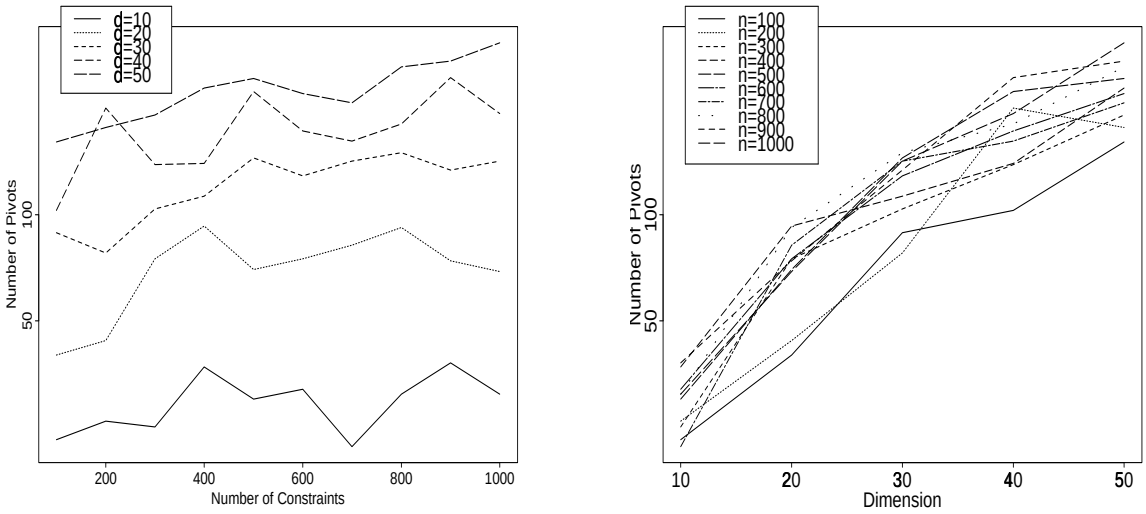

Figure 4.2: Number of Pivots with Dual Simplex

- Add a dual structure, which reflects in some sense the duality of linear programs. Especially the framework should be able not only to deal with deletion of constraints but also with restriction to constraints (e.g. the analysis of Seidel's algorithm should then be possible in this framework). Maybe also a new definition of the combinatorial dimension (which is not a global constant but depends on a constraint set or some dual structure) could lead to better results.

We remark that our technique of combinatorial bounding boxes, especially the lexicographical box, will perhaps be useful for other, more general optimization problems than linear programming.

- **Algorithms:**
  New axiomatic frameworks can give the means for the design and analysis of new algorithms; probably the other direction will be more important in further research: The study of known algorithms can lead to new axiomatics. So far the geometric point of view was the origin for randomized algorithms for linear programming (as the algorithms of Seidel or Kalai or also **BasisPivotLP**); we think that in the future the investigation of pivot algorithms (like Criss-Cross or also **BasisPivotLP** and variants) could lead to more insight in the structures of linear programming.
  Another source of inspiration might be the study of the practical behavior of implemented algorithms. Large test series can lead to a serious comparison of several algorithms and to a prediction of their practical behaviour, well chosen examples may give hints for critical decisions during the search for the optimal solution. Also for theory examples can be fruitful when they reveal lower bounds for the computational costs of an algorithm (as **BasisLP** or Criss-Cross).

Acknowledgements

This diploma thesis was written under the supervision of *Prof. H.-J. Lüthi* and *Prof. K. Fukuda.* I thank them especially for the discussions and suggestions which were inspiring and useful for my work. In particular the replacement of the λ-box by the L-box is due to Komei Fukuda.

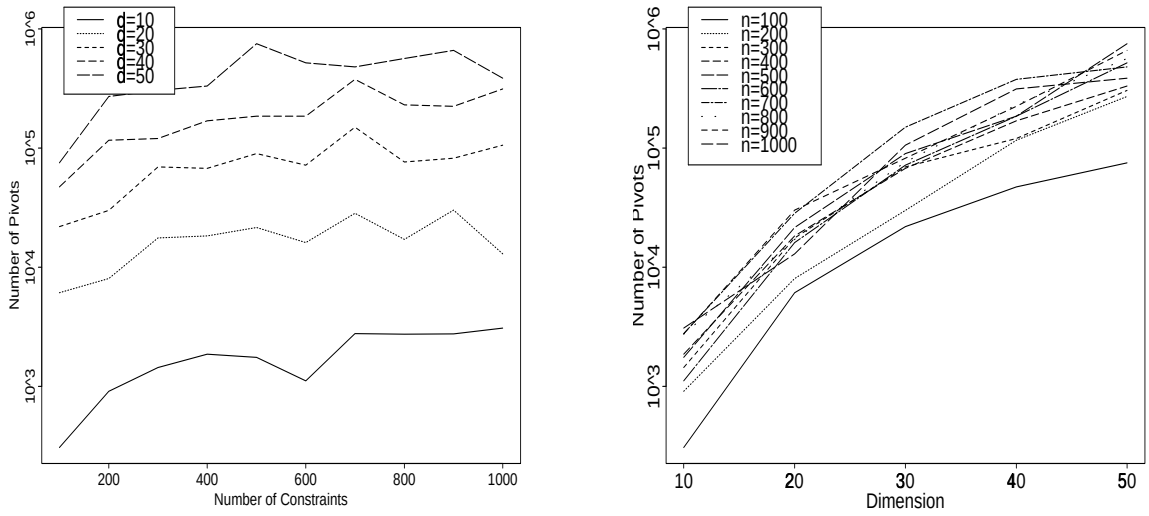

Figure 4.3: Number of Pivots with Criss-Cross

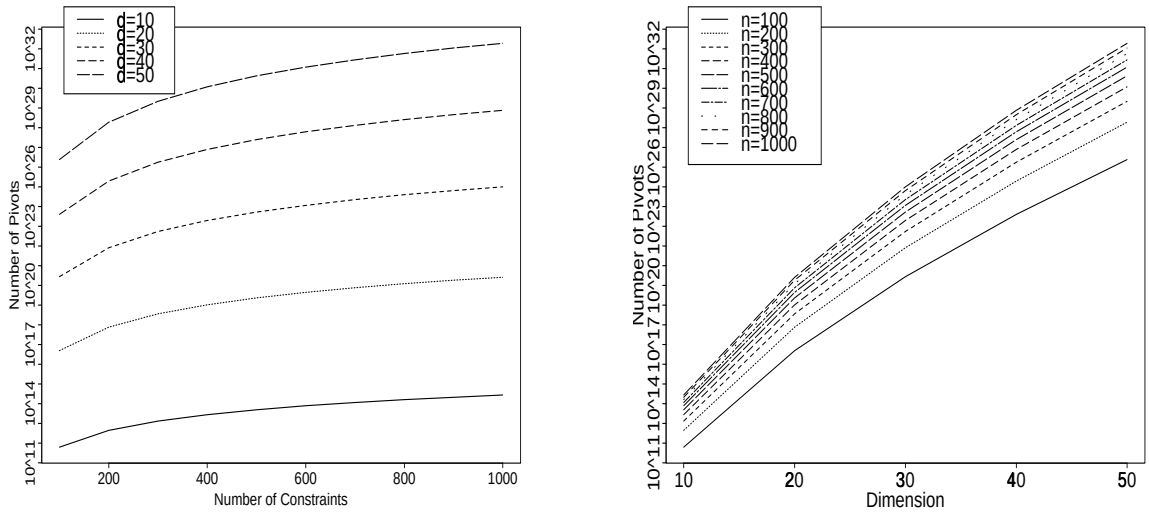

Figure 4.4: Subexponential Bound for Number of Pivots in **BasisPivotLP**

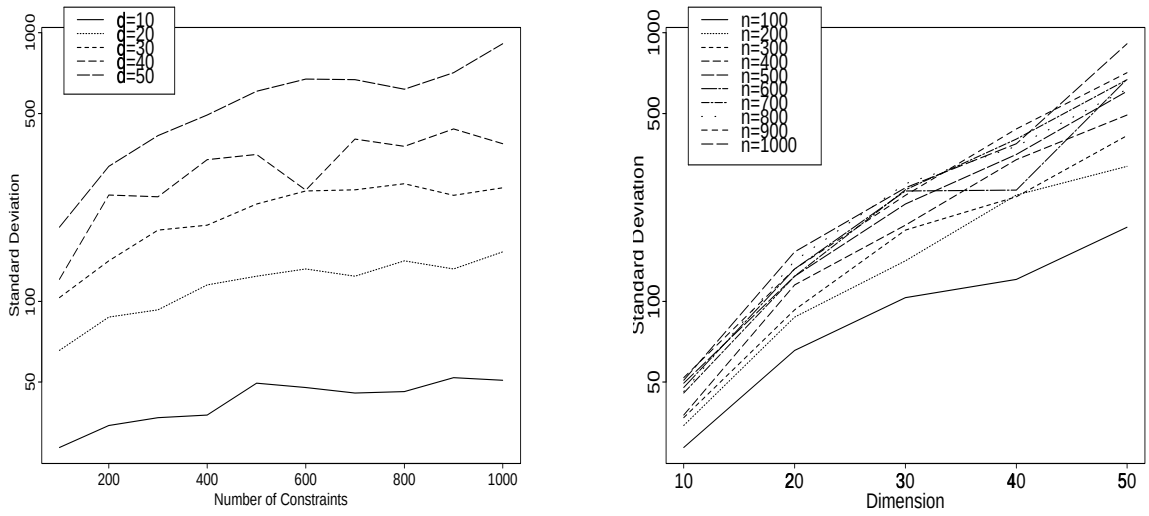

Figure 4.5: Standard Deviation of Number of Pivots with **BasisPivotLP**

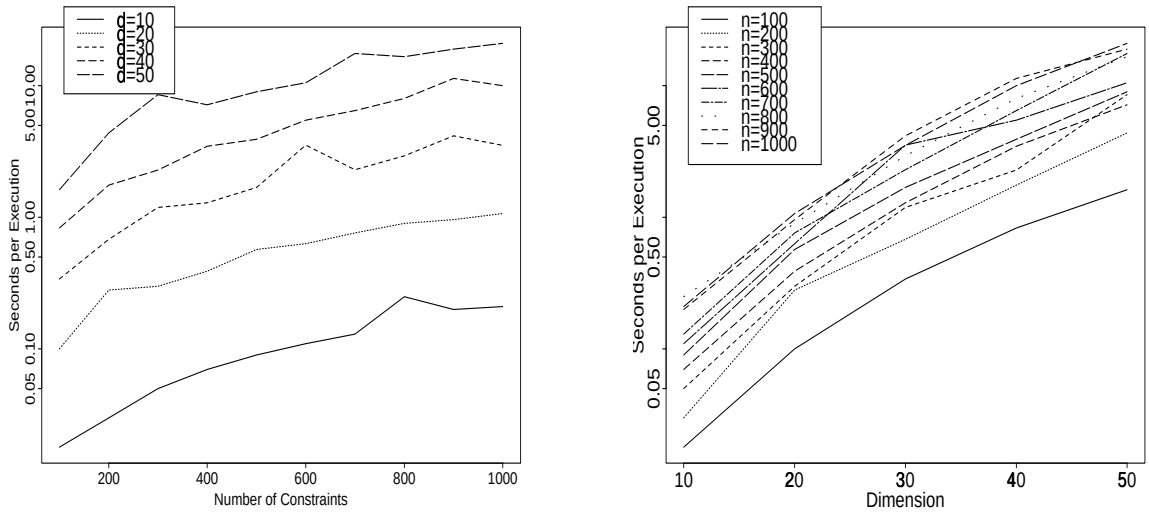

Figure 4.6: Run Time of **BasisPivotLP**

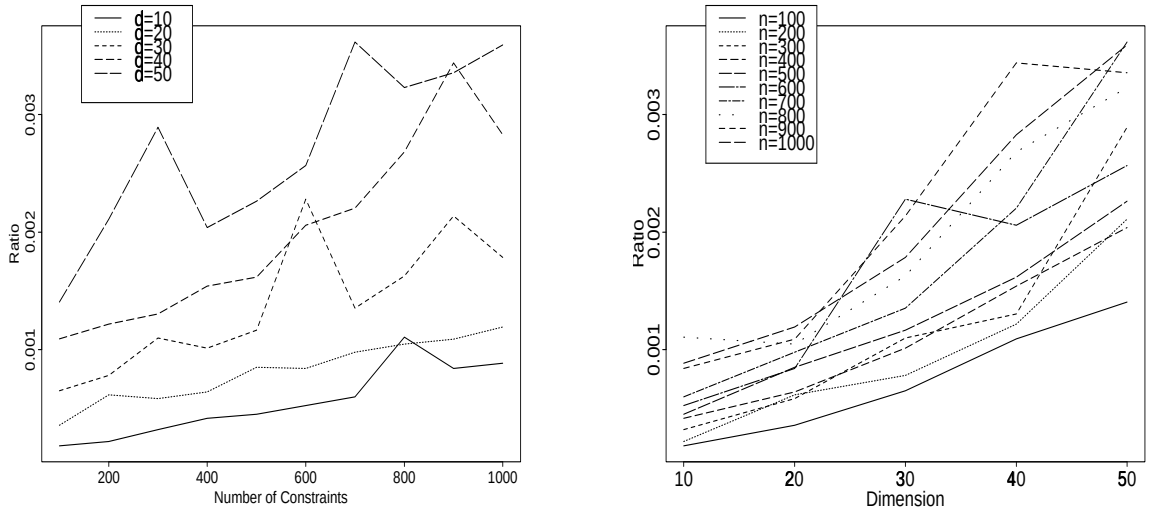

Figure 4.7: Ratio of Run Time per Pivot Operation in **BasisPivotLP**

Appendix A

Source Code of the Implementation of BasisPivotLP

```
1    /* Eidgenoessische Technische Hochschule Zuerich
2
3       Institut fuer Operations Research
4       Clausiusstrasse 45
5       CH-8092 Zuerich
6
7       bp6.c: BasisPivotLP in C (Implementation 6)
8
9       Lukas Finschi, 19.2.1997 - 28.2.1997
10
11
12      Procedures in bp6.c: - read_input
13                           - initialize_data
14                           - zero
15                           - dict_entry
16                           - find_pivot
17                           - pivot_operation
18                           - basis_pivot_lp
19                           - main
20
21      Numerical instability is possible for some input data (especially when
22      d is small and n is large).
23   */
24
25   /* ========================================================================= */
26   /* Includings:                                                               */
27
28   #include <stdio.h>
29   #include <stdlib.h>
30   #include <string.h>
31   #include <float.h>
32   #include <time.h>
33
34   /* ========================================================================= */
35
36   #define MAXLINE        1003
37   #define MAXSTRING      200
38   #define BEGIN_STR      "begin"
39   #define END_STR        "end"
```

```
40  #define INTEGER_STR    "integer"
41  #define MAX_STR        "maximize"
42  #define NEXEC_DEFAULT  10
43
44  #define OUT            2            /* 0: stdout; 1: fix name; 2: var. ext. */
45  #define OUT_FILE       "bp.bps"
46  #define PIV_FILE       "bp.bpp"
47  #define IN_EXT         ".ine"
48  #define OUT_EXT        ".bps"
49  #define PIV_EXT        ".bpp"
50
51  #define EPS            0.00001
52
53  #define TRUE           0==0
54  #define FALSE          0==1
55
56  #define FEASIBLE       0          /* FEASIBLE has to be a nonnegative number */
57  #define INFEASIBLE     -1          /* INFEASIBLE has to be a negative number */
58  #define IN_BASIS       -1            /* IN_BASIS has to be a negative number */
59  #define LOWERBOUND     0
60  #define UPPERBOUND     1
61
62  /* ========================================================================== */
63  /* Global Data:                                                               */
64
65     int n, d, cols, *marked_in_basis, *nonbasis, *x_to_nonbasis, *bound,\
66         *sel, pivot_count, *old_nonbasis, infeas_constr;
67     double *mc, *r, *d_i_nb;
68     FILE *fpout, *fppiv;
69
70  /*
71     Remarks:
72
73        n:               Number of constraint variables (= |Y|).
74
75        d:               Dimension (= |X|).
76
77        cols:            Number of columns in matrix mc (= n+d+2).
78
79        marked_in_basis: Pointer to vector of length n+1:
80                         marked_in_basis[0] = the number of marked basis
81                         constraints; the elements marked_in_basis[1],...,
82                         marked_in_basis[marked_in_basis[0]] are (in any order)
83                         the indices of the marked basis constraints (indices:
84                         for correspondence see mc below); the elements
85                         marked_in_basis[marked_in_basis[0]+1],...,
86                         marked_in_basis[n] are undefined.
87
88        nonbasis:        Pointer to vector of length d+1:
89                         The elements nonbasis[0],...,nonbasis[d] are (in order
90                         corresponding to the rows of r) the indices of the
91                         nonbasis constraints (indices: for correspondence see
92                         mc below); always nonbasis[0] == 0.
93
94        x_to_nonbasis:   Pointer to vector of length d+1:
95                         If for 0 <= t <= d there exists a 0 <= s <= d such that
96                         nonbasis[s] == t, then x_to_nonbasis[t] = s; otherwise
```

```
                        x_to_nonbasis[t] = -1.

        bound:          Pointer to vector of length d+1:
                        The elements bound[1],...,bound[d] are (in order
                        corresponding to nonbasis[1],...,nonbasis[d])
                           - either LOWERBOUND (if i = xi_nonbasis[s] is a
                             variable with flag "i-bound = lowerbound"),
                           - or     UPPERBOUND (if i = xi_nonbasis[s] is a
                             variable with flag "i-bound = upperbound").
                        bound[0] is undefined.

        sel:            Vectors of length d+1 used in find_pivot;
                        sel[0] = number of elements in the selection.

        pivot_count:    Contains the number of pivot operations used to reach
                        the solution.

        old_nonbasis:   nonbasis of the previous test (for correctness check)

        infeas_constr:  In the case of infeasibility the index of a constraint
                        which defines together with the nonbasis constraints
                        an empty feasibility region.

        mc:             Matrix C of dimension (d+1) x (n+d+2) of the form

                               ( 1 0 0 ... 0   b_1   b_2  ...  b_n    0  )
                               ( 0 1 0 ... 0  -A_11 -A_21 ... -A_n1  c_1 )
                               ( . . . ... .    .     .    ...   .     .  )
                          mc = ( . . . ... .    .     .    ...   .     .  )
                               ( . . . ... .    .     .    ...   .     .  )
                               ( 0 0 0 ... 1  -A_1d -A_2d ... -A_nd  c_d )

                         or, in compact writing,

                               (  1   b^T  0 )   } 1 row
                          mc = (  |  -A^T  c )   } d rows

                                    v    v   v
                                  d+1,  n,  1 column(s)

                         The rows are labelled by the indices 0,1,...,d,
                         the columns by 0,1,...,d, d+1,...,d+n, d+n+1.
                         The correspondence of the indices to the variables is:
                                0 <=> 1 (the unit column of the dictionary),
                                1 <=> x_1 = xi_1
                                .  .  .
                                d <=> x_d = xi_d
                              d+1 <=> y_1 = xi_{d+1}
                                .  .  .
                              d+n <=> y_n = xi_{d+n}
                            d+n+1 <=> z (= c^T x, the objective value)

        r:              Regular matrix of dimension (d+1) x (d+1);
                        r is the inverse matrix of S, where

                           S =  (mc_nonbasis[0] ... mc_nonbasis[d]),
```

```
                        where mc_t is the column with index t of mc.

       d_i_nb:          Pointer to vector of length d+1 used in find_pivot;
                        contains entries D_i_nonbasis of the current dictionary
                        (in order corresponding to nonbasis).
                        Also used as vector u in pivot_operation.

       fpout:           Pointer for output file (solution).

       fppiv:           Pointer for output file (list of number of pivot steps).
*/

/* ========================================================================== */
/* Procedure: int read_input(char *filenamep)

   Input:     filename:   Pointer to string which contains the name of
                          the input file

   Output:    If no error occurs: 0;
              if input file can not be opened: 1;
              if input file can not be read correctly: 2;
              if in the input file a line is too long (> MAXLINE): 3;
              if the format of the input file is wrong: 4;
              if there is not enough free memory: 5;
              if input file can not be closed correctly: 6.

   Comments:  Reads data from input file; the format of the input file has to
              be as follows:

                 various comments [which are ignored]
                 begin
                 <n> <d+1> <numbertype>
                 0 1 0 0 ... 0 [d+1 columns]
                 0 0 1 0 ... 0
                 . . . . ... .
                 0 0 0 0 ... 1
                 <b_1> <-A_11> ... <-A_1d>
                   .      .    ...    .
                 <b_n> <-A_1d> ... <-A_nd>
                 end
                 various comments [which are ignored]
                 maximize <c_0> <c_1> ... <c_d>
                 various comments [which are ignored]

              <n> and <d+1> have to be positive integers.
              <numbertype> has to be "integer" (in this program version),
              all elements of A, b, and c have to be integers.

              This procedure allocates memory for mc, r, marked_in_basis,
              nonbasis, x_to_nonbasis, bound, sel, d_i_nb.

              This procedure initializes d, n, cols, mc.

              For the error reports see "output".
*/

int read_input(char *filenamep)
```

```
{
   int i, j, tmp_int;
   char line[MAXLINE], tmp_string[MAXLINE], *linep;
   FILE *fp;

   fp = fopen(filenamep, "r");                          /* open input file  */
   if(fp==NULL) return 1;

   do {                                                 /* scan for "begin" */
      linep = fgets(line, MAXLINE, fp);
      if(linep==NULL) return 2;
      if(strlen(line) == MAXLINE - 1) return 3;
      sscanf(line, "%s", tmp_string);
   } while(strcmp(tmp_string, BEGIN_STR)!=0);

   linep = fgets(line, MAXLINE, fp);             /* scan "n d+1 numbertype" */
   if(linep==NULL) return 2;
   if(strlen(line) == MAXLINE - 1) return 3;
   if(sscanf(line, "%d %d %s", &n, &d, tmp_string)!=3) return 4;
   d -= 1; n -= d;
   if(d<=0 || n<=0) return 4;
   mc = (double *) malloc((d+1)*(n+d+2)*sizeof(double));
   if(mc==NULL) return 5;
   cols = (n+d+2);
   r = (double *) malloc((d+1)*(d+1)*sizeof(double));
   if(r==NULL) return 5;
   marked_in_basis = (int *) malloc((n+1)*sizeof(int));
   if(marked_in_basis==NULL) return 5;
   nonbasis = (int *) malloc((d+1)*sizeof(int));
   if(nonbasis==NULL) return 5;
   old_nonbasis = (int *) malloc((d+1)*sizeof(int));
   if(old_nonbasis==NULL) return 5;
   x_to_nonbasis = (int *) malloc((d+1)*sizeof(int));
   if(x_to_nonbasis==NULL) return 5;
   bound = (int *) malloc((d+1)*sizeof(int));
   if(bound==NULL) return 5;
   sel = (int *) malloc((d+1)*sizeof(int));
   if(sel==NULL) return 5;
   d_i_nb = (double *) malloc((d+1)*sizeof(double));
   if(d_i_nb==NULL) return 5;

   if(strcmp(tmp_string, INTEGER_STR)==0) {             /* scan integer data */
      for(i=1; i<=d; i++) {                       /* first d rows == box (?) */
         for(j=0; j<=d; j++)
            if((fscanf(fp, "%d", &tmp_int)!=1) || (tmp_int!=(j==i? 1.0: 0.0)))
                  return 4;
      }
      for(i=0; i<=d; i++)                              /* write identity part */
          for(j=0; j<=d; j++) mc[j*cols+i] = (j==i? 1.0: 0.0);
      for(i=d+1; i<n+d+1; i++) {                       /* scan and write data */
         for(j=0; j<=d; j++) {
            if(fscanf(fp, "%d", &tmp_int)!=1) return 4;
            mc[j*cols+i] = (double) tmp_int;
         }
      }
      linep = fgets(line, MAXLINE, fp);                    /* scan for "\n" */
      linep = fgets(line, MAXLINE, fp);                   /* scan for "end" */
```

```
268       if(linep==NULL) return 2;
269       if(strlen(line) == MAXLINE - 1) return 3;
270       sscanf(line, "%s", tmp_string);
271       if(strcmp(tmp_string, END_STR)!=0) return 4;
272       do {                                          /* scan for "maximize" */
273          if(fscanf(fp, "%s", tmp_string)!=1) return 4;
274       } while(strcmp(tmp_string, MAX_STR)!=0);
275       for(j=0; j<=d; j++) {                    /* scan for objective function */
276          if(fscanf(fp, "%d", &tmp_int)!=1) return 4;
277          mc[j*cols+(n+d+1)] = (double) tmp_int;
278       }
279    }
280    else return 4;
281    if(fclose(fp)!=NULL) return 6;
282
283    return 0;
284 }
285
286 /* ========================================================================= */
287 /* Procedure: void initialize_data(void)
288
289    Input:
290
291    Output:
292
293    Comments:  This procedure intializes r, marked_in_basis, nonbasis,
294               x_to_nonbasis, bound.
295 */
296
297 void initialize_data(void)
298 {
299    int i, j;
300
301    for(i=0; i<=d; i++)                              /* write r (identity) */
302       for(j=0; j<=d; j++) r[j*(d+1)+i] = (j==i? 1.0: 0.0);
303
304    marked_in_basis[0] = n;                     /* initialize marked_in_basis */
305    for(j=1; j<=n; j++) marked_in_basis[j] = d+j;
306
307    for(j=0; j<=d; j++) {                                    /* initialize */
308       nonbasis[j] = j;                                       /* nonbasis */
309       x_to_nonbasis[j] = j;                             /* and x_to_nonbasis */
310    }
311
312    for(j=1; j<=d; j++) {                              /* initialize bound */
313       if(mc[j*cols+(n+d+1)]>=0) bound[j] = UPPERBOUND;
314       else bound[j] = LOWERBOUND;
315    }
316
317    return;
318 }
319
320 /* ========================================================================= */
321 /* Procedure: int zero(double x)
322
323    Input:     x: a double float.
324
```

```
325     Output:    If x == 0 (with tolerance EPS): TRUE; otherwise: FALSE.
326
327     Comments:  If x is near to zero (-EPS < x < EPS), then the return value is
328                TRUE (x is considered to represent numerically 0.0);
329                otherwise x is considered to be different from zero.
330  */
331
332  int zero(double x)
333  {
334     return (int) (-EPS<x && x<EPS);
335  }
336
337  /* ========================================================================= */
338  /* Procedure: double dict_entry(int i, int j_ind)
339
340     Input:     i:     index of a variable in the current basis (1<=i<=d+n+1).
341                j_ind: number of element of nonbasis (1<=j_ind<=d+1).
342
343     Output:    The element D_ij of the current dictionary, where
344                j = nonbasis[j_ind].
345
346     Comments:  We use the relation D_ij = R_j C_.i, i.e.
347                D_ij = sum_{k in N}{R_jk * C_ki}.
348
349                If i in X, then D_ij = R_ji.
350  */
351
352  double dict_entry(int i, int j_ind)
353  {
354     int k;
355     double d_ij;
356
357     if(i<=d) return r[j_ind*(d+1)+i];                          /* case i in X */
358
359     d_ij = 0;
360     for(k=0; k<=d; k++) d_ij += r[j_ind*(d+1)+k]*mc[k*cols+i];
361     return d_ij;
362  }
363
364  /* ========================================================================= */
365  /* Procedure: int find_pivot(int i, int si)
366
367     Input:     i:  pivot row (= violated constraint)
368                si: sign of i, i.e. -1 (if xi_i < 0) or 1 (if xi_i > lambda_i)
369
370     Output:    if feasible: j; if infeasible: INFEASIBLE (see comments).
371
372     Comments:  Determines whether the d+1 constraints in F(D) + {h_i} define
373                a feasible LP or not. In the first case the return value j is
374                such that after a pivot operation on (i,j) the new dictionary
375                is optimal for all d+1 constraints; otherwise the return value
376                is INFEASIBLE.
377
378                This procedure is the implementation of FindPivot2.
379  */
380
381  int find_pivot(int i, int si)
```

```
{
   int t, j_ind, sel_ind, sel_length, t_ind;
   double d_zj, q, q_temp;

   sel[0] = 0;                                          /* selection = emtpyset */
   for(j_ind=1; j_ind<=d; j_ind++) {                   /* feasibility selection */
      d_i_nb[j_ind] = dict_entry(i, j_ind);
      if(!zero(d_i_nb[j_ind]) \
            && ((bound[j_ind]==LOWERBOUND) ^ (si<0) ^ (d_i_nb[j_ind]>0))) {
         sel[0] += 1;
         sel[sel[0]] = j_ind;
      }
   }
   if(sel[0] == 0) return INFEASIBLE;

   q = DBL_MAX;                                            /* value optimality */
   sel_length = sel[0];
   for(sel_ind=1; sel_ind<=sel_length; sel_ind++) {
      j_ind = sel[sel_ind];
      d_zj = dict_entry(n+d+1, j_ind);
      q_temp = (d_zj/d_i_nb[j_ind])*si;
      if(q_temp < q) { q = q_temp; sel[0] = 1; sel[1] = j_ind; }
      else if(q_temp == q) { sel[0] += 1; sel[sel[0]] = j_ind; }
   }

   t = 1;                                        /* lexicographical optimality */
   while(sel[0] > 1) {
      t_ind = x_to_nonbasis[t];
      for(sel_ind=1; sel_ind<=sel[0] && sel[sel_ind]!=t_ind; sel_ind++);
      if(sel_ind<=sel[0]) {                                          /* t in S */
         if(bound[t_ind]==LOWERBOUND) return t;
         else {
            sel[sel_ind] = sel[sel[0]];
            sel[0] -= 1;
         }
      }
      else {
         if(t_ind==IN_BASIS && t!=i) {
            q = DBL_MAX;
            sel_length = sel[0];
            for(sel_ind=1; sel_ind<=sel_length; sel_ind++) {
               j_ind = sel[sel_ind];
               q_temp = (dict_entry(t, j_ind)/d_i_nb[j_ind])*si;
               if(q_temp < q) { q = q_temp; sel[0] = 1; sel[1] = j_ind; }
               else if(q_temp == q) { sel[0] += 1; sel[sel[0]] = j_ind; }
            }
         }
      }
      t += 1;
   }

   return nonbasis[sel[1]];
}

/* ========================================================================= */
/* Procedure: int pivot_operation(int i, int k, int si)
```

```
439     Input:     i, k:  pivot, where i in B and k in N
440                si:    if xi_i < 0: -1; if xi_i > lambda_i: 1
441
442     Output:    if no error occurs: 0; if an error occurs: 1.
443
444     Comments:  Modifies r, marked_in_basis, nonbasis, x_to_nonbasis, bound,
445                pivot_table. This procedure does not run correctly if not
446                marked_in_basis[marked_in_basis[0]] == i at beginning.
447  */
448
449  int pivot_operation(int i, int k, int si)
450  {
451     int s, j, k_ind, ell, ell_ind;
452     double w_ell;
453
454     marked_in_basis[marked_in_basis[0]] = k;
455
456     for(k_ind=1; nonbasis[k_ind]!=k; k_ind++);
457     nonbasis[k_ind] = i;
458
459     if(i<=d) x_to_nonbasis[i] = k_ind;
460     if(k<=d) x_to_nonbasis[k] = -1;
461
462     if(si==-1) bound[k_ind] = LOWERBOUND; else bound[k_ind] = UPPERBOUND;
463
464     ell_ind = 0;
465     do {
466        ell_ind += 1;
467        ell = nonbasis[ell_ind];
468        w_ell = 0;
469        for(j=0; j<=d; j++) w_ell += r[k_ind*(d+1)+j]*mc[j*cols+ell];
470     } while(ell_ind<d && w_ell>-EPS && w_ell<EPS);
471     if(w_ell>-EPS && w_ell<EPS) return 1;
472
473     for(s=0; s<=d; s++) {
474        d_i_nb[s] = (s==ell_ind? 1.0: 0.0);
475        for(j=0; j<=d; j++) d_i_nb[s] -= r[s*(d+1)+j]*mc[j*cols+ell];
476        d_i_nb[s] /= w_ell;
477     }
478
479     for(s=0; s<=d; s++)
480        if (s!=k_ind)
481           for(j=0; j<=d; j++) r[s*(d+1)+j] += d_i_nb[s]*r[k_ind*(d+1)+j];
482     for(j=0; j<=d; j++) r[k_ind*(d+1)+j] *= (1+d_i_nb[k_ind]);
483
484     pivot_count += 1;
485
486     return 0;
487  }
488
489  /* ========================================================================= */
490  /* Procedure: int basis_pivot_lp(void)
491
492     Input:
493
494     Output:    FEASIBLE or INFEASIBLE (see comments).
495
```

```
496     Comments:  Determines whether the constraints in F(D) + marked_in_basis
497                define a feasible LP or not. In the first case the return value
498                is FEASIBLE and the current dictionary defines the optimal
499                solution; otherwise the return value is INFEASIBLE and the
500                constraints in the nonbasis together with the constraint in
501                infeasible_constr suffice for the infeasibility (i.e. there is
502                no feasible point for these d+1 constraints).
503
504                This procedure is the implementation of BasisPivotLP.
505  */
506
507  int basis_pivot_lp(void)
508  {
509     int i, i_ind, j, t, t_ind, si;
510     double d_it;
511
512     if(marked_in_basis[0]==0) return FEASIBLE;
513     else {
514        i_ind = (rand() % marked_in_basis[0]) + 1;   /* choose i at random and */
515        i = marked_in_basis[i_ind];                                /* unmark it */
516        marked_in_basis[i_ind] = marked_in_basis[marked_in_basis[0]];
517        marked_in_basis[0] -= 1;
518        if(basis_pivot_lp()==INFEASIBLE) return INFEASIBLE; /* 1st recur. call */
519        marked_in_basis[0] += 1;                          /* recover new set of */
520        marked_in_basis[marked_in_basis[0]] = i;   /* marked basis constraints */
521
522        t = d;                     /* find t = max {k in N | D_ik xi_k^= <> 0} */
523        do {
524           d_it = 0;
525           t_ind = x_to_nonbasis[t];
526           if(t_ind>0 && bound[t_ind]==UPPERBOUND)
527              d_it = dict_entry(i, t_ind);
528           if(zero(d_it)) { t_ind = IN_BASIS; t -= 1; }
529        } while(t_ind==IN_BASIS && t>0);
530        if(t_ind==IN_BASIS) d_it = dict_entry(i, 0);
531
532        if(d_it>=0) si = 1; else si = -1;     /* si := sign(xi_i) = sign(d_it) */
533        if(si==1 && t<i) return FEASIBLE;             /* return if no violation*/
534
535        j = find_pivot(i, si);
536        if(j==INFEASIBLE) { infeas_constr = i; return INFEASIBLE; }
537        if(pivot_operation(i, j, si)!=0) {
538           printf("basis_pivot_lp: pivot_operation: Fatal Error.\n");
539           exit(1);
540        }
541        return basis_pivot_lp();                      /* second recursive call */
542     }
543  }
544
545  /* ========================================================================== */
546  /* Procedure: int main(int argc, char *argv[])
547
548     Input:     argc:       number of arguments in the command line
549                            (argc >= 2)
550                argv:       argv[0] is the name of the program,
551                            argv[1] is the name of the input file;
552                            argv[2] (optional) is the seed of the random
```

```
553                           generator (has to be unsigned int);
554                           further arguments will be ignored
555
556      Output:    if no error occurs: 0; otherwise: 1.
557                 The program generates output files.
558
559      Comments:  Later...
560
561      Mnemonics: bp6: BasisPivotLP, Version 6
562   */
563
564   int main(int argc, char *argv[])
565   {
566      int i, s, t, err, samesolution, nexec, *pivots;
567      double val, mean, var, sum;
568      char datestr[30], timestr[30], *ext_str,\
569         outstr[MAXSTRING], pivstr[MAXSTRING];
570      time_t actualtime, start_time;
571
572      if(argc < 2) {
573         printf("bp6: main: Too few arguments in command line.\n");
574         exit(1);
575      }
576
577      err = read_input(argv[1]);
578      if(err==1) {
579         printf("bp6: read_input: Error opening file \"%s\".\n", argv[1]);
580         exit(1);
581      }
582      if(err==2) {
583         printf("bp6: read_input: Error reading file \"%s\".\n", argv[1]);
584         exit(1);
585      }
586      if(err==3) {
587         printf("bp6: read_input: Too long (> %d) line in file \"%s\".\n",\
588                (int) MAXLINE-3, argv[1]);
589         exit(1);
590      }
591      if(err==4) {
592         printf("bp6: read_input: Unexpected format of file \"%s\".\n", argv[1]);
593         exit(1);
594      }
595      if(err==5) {
596         printf("bp6: read_input: Not enough memory.\n");
597         exit(1);
598      }
599      if(err==6) {
600         printf("bp6: read_input: Error closing file \"%s\".\n", argv[1]);
601         exit(1);
602      }
603
604      printf("bp6: read_input: ok.\n");
605
606      if(OUT==0) {
607         fpout = stdout;
608         fppiv = stdout;
609         printf("bp6: Output on stdout.\n");
```

```
610     }
611     else if(OUT==1) {
612        fpout = fopen(OUT_FILE, "w");
613        if(fpout==NULL) {
614           printf("bp6: Error opening file \"%s\".\n", OUT_FILE);
615           exit(1);
616        }
617        else printf("bp6: Opened file \"%s\" for output of solution.\n",\
618              OUT_FILE);
619        fppiv = fopen(PIV_FILE, "w");
620        if(fppiv==NULL) {
621           printf("bp6: Error opening file \"%s\".\n", PIV_FILE);
622           exit(1);
623        }
624        else printf("bp6: Opened file \"%s\" for output of pivot numbers.\n",\
625              PIV_FILE);
626     }
627     else {                                                        /* OUT==2 */
628        if(strlen(argv[1])>MAXSTRING-6) {
629           printf("bp6: read_input: Too long (> %d) input file name \"%s\".\n",\
630                  (int) MAXSTRING-6, argv[1]);
631           exit(1);
632        }
633        ext_str = strrchr(argv[1], (int) '.');
634        if(ext_str==NULL) strcpy(outstr, argv[1]);  /* no extension -> add ext */
635        else strncpy(outstr, argv[1], ext_str-argv[1]);   /* replace extension */
636        strcpy(pivstr, outstr);
637        strcat(outstr, OUT_EXT);
638        strcat(pivstr, PIV_EXT);
639        fpout = fopen(outstr, "w");
640        if(fpout==NULL) {
641           printf("bp6: Error opening file \"%s\".\n", outstr);
642           exit(1);
643        }
644        else printf("bp6: Opened file \"%s\" for output of solution.\n", outstr);
645        fppiv = fopen(pivstr, "w");
646        if(fppiv==NULL) {
647           printf("bp6: Error opening file \"%s\".\n", pivstr);
648           exit(1);
649        }
650        else printf("bp6: Opened file \"%s\" for output of pivot numbers.\n",\
651              pivstr);
652     }
653
654     fprintf(fpout, "\n************************************************\n");
655     fprintf(fpout, "*** Output of BasisPivotLP                    ***\n");
656     fprintf(fpout, "***                                           ***\n");
657     fprintf(fpout, "*** Version 6 (bp6.c)                         ***\n");
658     if (time(&actualtime)!=-1
659         && strftime(datestr, 30, "%a %b %d %Y", localtime(&actualtime))!=0
660         && strftime(timestr, 30, "%H:%M:%S", localtime(&actualtime)) != 0 )
661        fprintf(fpout, "*** Date: %s   Time: %s    ***\n", datestr, timestr);
662     fprintf(fpout, "************************************************\n\n");
663
664     start_time = actualtime;
665
666     if(argc>=3 && sscanf(argv[2], "%d", &nexec)==1)
```

```
667        fprintf(fpout, "Number of executions: %d\n\n", nexec);
668     else {
669        nexec = NEXEC_DEFAULT;
670        fprintf(fpout, "Default number of executions: %d\n\n", nexec);
671     }
672
673     fprintf(fpout, "Dimensions:  %i\n", d);
674     fprintf(fpout, "Constraints: %i  (+ nonnegativity constraints)\n\n", n);
675     fprintf(fpout, "Input file:  \"%s\".\n\n", argv[1]);
676
677     pivots = (int *) malloc((nexec+1)*sizeof(int));
678     if(pivots==NULL) {
679        printf("bp6: Error allocating memory for pivot counts.\n");
680        exit(1);
681     }
682     pivots[0] = nexec;
683
684     samesolution = 0;
685     for(i=1; i<=nexec; i++) {
686        srand((unsigned int) i);
687        initialize_data();
688        pivot_count = 0;
689        err = basis_pivot_lp();
690        pivots[i] = pivot_count;
691        if(i==1) {
692           if(err==0) for(s=0; s<=d; s++) old_nonbasis[s] = nonbasis[s];
693           else old_nonbasis[0] = -1;
694           printf(" %d: First Result ", i);
695           if(err==0) printf("(feasible).\n"); else printf("(infeasible).\n");
696        }
697        else {
698           if(err==0) {
699              for(s=0; s<=d; s++) {
700                 for(t=0; t<=d && nonbasis[s]!=old_nonbasis[t]; t++);
701                 if(t>d) {
702                    printf("%d: CHANGE OF NONBASIS.\n", i);
703                    samesolution = 1;
704                    s=d+1;
705                 }
706              }
707              if(t<=d && i==nexec) printf("%d: o.k.\n", i);
708           }
709           else {
710              if(old_nonbasis[0]!=-1) {
711                 printf("%d: CHANGE OF NONBASIS.\n", i);
712                 samesolution = 1;
713              }
714              else if(i==nexec) printf("%d: o.k.\n", i);
715           }
716        }
717
718     }
719     if(err==INFEASIBLE) fprintf(fpout, "LP IS INFEASIBLE.\n\n");
720     else fprintf(fpout, "LP is feasible.\n\n");
721
722     fprintf(fpout, "Nonbasis: ");
723     for(i=1; i<=d; i++) fprintf(fpout, "%2d ", nonbasis[i]);
```

```
724     if(err==INFEASIBLE) fprintf(fpout, "\nInfeasible Contraint d+1 = %d\n",\
725           infeas_constr);
726     fprintf(fpout, "\nBound:    ");
727     for(i=1; i<=d; i++) fprintf(fpout, "%2d ", bound[i]);
728     fprintf(fpout, "\n\n");
729
730     if(err!=INFEASIBLE) {
731        fprintf(fpout, "Solution: ");
732        if(samesolution==0) fprintf(fpout, "(for all tests)\n");
733        else fprintf(fpout, "(of last test; NOT ALL TESTS HAVE SAME RESULT!)\n");
734        for(i=1; i<=d; i++) {
735           if(x_to_nonbasis[i]>=0) {
736              if(bound[x_to_nonbasis[i]]==0)
737                 fprintf(fpout, "x[%d] = %15.8f\n", i, 0.0);
738              else fprintf(fpout, "x[%d] = lambda_%d\n", i, i);
739           }
740           else {
741              val=0;
742              for(s=0; s<=d; s++) val += r[s]*mc[s*cols+i];
743              fprintf(fpout, "x[%d] = %15.8f (+ lambda part (?))\n", i, val);
744           }
745        }
746     }
747
748     fprintf(fpout, "\nNumber of Pivot Steps:\n");
749     sum = 0.0;
750     for(nexec = 1; nexec<=pivots[0]; nexec++) sum += pivots[nexec];
751     mean = sum/pivots[0];
752     fprintf(fpout, "     Average: %8.4f\n", mean);
753     sum = 0.0;
754     for(nexec = 1; nexec<=pivots[0]; nexec++)
755        sum += (pivots[nexec]-mean)*(pivots[nexec]-mean);
756     var = sum/pivots[0];
757     fprintf(fpout, "     Variance: %8.4f\n", var);
758
759     for(nexec = 1; nexec<=pivots[0]; nexec++)
760        fprintf(fppiv, "%d\n", pivots[nexec]);
761
762     fprintf(fpout, "\n*************************************************\n");
763     fprintf(fpout, "***             END OF OUTPUT                 ***\n");
764     if (time(&actualtime)!=-1
765         && strftime(datestr, 30, "%a %b %d %Y", localtime(&actualtime))!=0
766         && strftime(timestr, 30, "%H:%M:%S", localtime(&actualtime)) != 0 )
767        fprintf(fpout, "*** Execution Time: %8.2f seconds          ***\n",\
768              difftime(actualtime, start_time));
769        fprintf(fpout, "*** Date: %s   Time: %s    ***\n", datestr, timestr);
770     fprintf(fpout, "*************************************************\n\n");
771
772     free(mc); free(r); free(marked_in_basis); free(nonbasis); free(bound);
773     exit(0);                                        /* exit includes fclose */
774  }
```

Bibliography

[1] Vašek Chvátal:
Linear Programming,
W. H. Freeman and Co., 1983,
ISBN 0-7167-1587-2

[2] Komei Fukuda, Tomomi Matsui:
On the finiteness of the criss-cross method,
European Journal of Operational Research 52, Elsevier Science Publishers B.V. (North-Holland), 1991, pp. 119 - 124

[3] Komei Fukuda:
cdd Reference Manual,
Institute for Operations Research, Zurich, Aug 27, 1996

[4] Komei Fukuda, Hans-Jakob Lüthi, Makoto Namiki:
The existence of a short sequence of admissible pivots to an optimal basis in LP and LCP,
Preprint (submitted to Elsevier Preprint), Institute for Operations Research, Zurich, Nov 19, 1996

[5] Michael Goldwasser:
A Survey of Linear Programming in Randomized Subexponential Time,
SIGACT News 26 no. 2, 1995, pp. 96 - 104

[6] Jiří Matoušek, Micha Sharir, Emo Welzl:
A Subexponential Bound for Linear Programming,
Proceedings of the eighth annual symposium on Computational Geometry, ACM, Berlin, 6/1992, pp. 1 - 8

[7] Jiří Matoušek:
Lower Bounds for a Subexponential Optimization Algorithm,
Random Structures and Algorithms, Vol. 5, No. 4, John Wiley & Sons, 1994, pp. 591 - 607

[8] Rajeev Motwani, Prabhakar Raghavan:
Randomized algorithms,
Cambrigde University Press, 1995

[9] Raimund Seidel:
Small-Dimensional Linear Programming and Convex Hulls Made Easy,
Discrete & Computational Geometry 6, Springer-Verlag, New York, 1991, pp. 423 - 434

[10] Emo Welzl:
LP with Small d - Algorithms and Applications,
Freie Universität Berlin, October 28, 1995