Chapter 11

Linear Programming

This lecture is about a special type of optimization problems, namely *linear programs*. We start with a geometric problem that can directly be formulated as a linear program.

11.1 Linear Separability of Point Sets

Let $P \subseteq \mathbb{R}^d$ and $Q \subseteq \mathbb{R}^d$ be two finite point sets in d-dimensional space. We want to know whether there exists a hyperplane that separates P from Q (we allow non-strict separation, i.e. some points are allowed to be on the hyperplane). Figure 11.1 illustrates the 2-dimensional case.

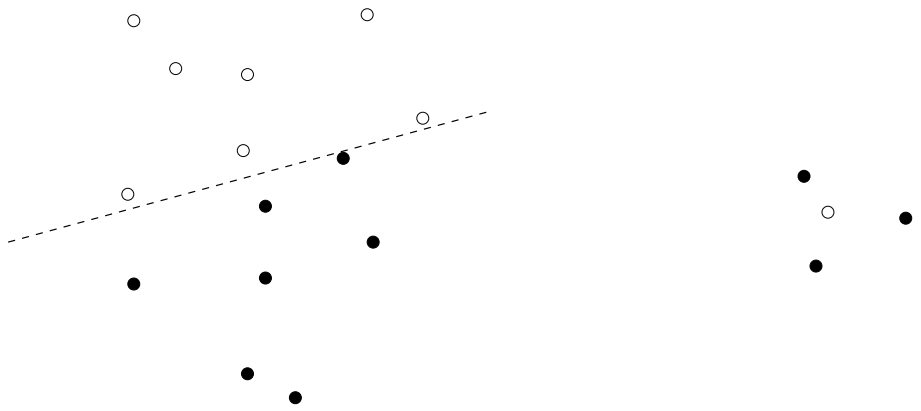


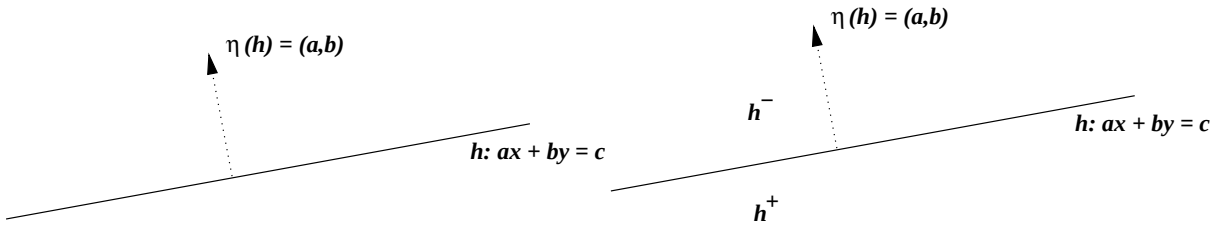
Figure 11.1: *Left: there is a separating hyperplane; Right: there is no separating hyperplane*

How can we formalize this problem? A hyperplane is a set of the form

$$h = \{x \in \mathbb{R}^d : h_1 x_1 + h_2 x_2 + \cdots + h_d x_d = h_0\},$$

where $h_i \in \mathbb{R}, i = 0, \ldots, d$. For example, a line in the plane has an equation of the form $ax + by = c$.

The vector $\eta(h) = (h_1, h_2, \ldots, h_d) \in \mathbb{R}^d$ is called the *normal vector* of h. It is orthogonal to the hyperplane and usually visualized as in Figure 11.2(a).



(a) The normal vector of a hyperplane **(b)** The two halfspaces of a hyperplane

Figure 11.2: *The concepts of hyperplane, normal vector, and halfspace*

Every hyperplane h defines two closed *halfspaces*

$$\begin{aligned} h^+ &= \{x \in \mathbb{R}^d : h_1x_1 + h_2x_2 + \cdots + h_dx_d \leqslant h_0\}, \\ h^- &= \{x \in \mathbb{R}^d : h_1x_1 + h_2x_2 + \cdots + h_dx_d \geqslant h_0\}. \end{aligned}$$

Each of the two halfpsaces is the region of space "on one side" of h (including h itself). The normal vector $\eta(h)$ points into h^-, see Figure 11.2(b). Now we can formally define linear separability.

Definition 11.1 *Two point sets $P \subseteq \mathbb{R}^d$ and $Q \subseteq \mathbb{R}^d$ are called* linearly separable *if there exists a hyperplane h such that $P \subseteq h^+$ and $Q \subseteq h^-$. In formulas, if there exist real numbers $h_0, h_1, \ldots, h_d$ such that*

$$\begin{aligned} h_1p_1 + h_2p_2 + \cdots + h_dp_d &\leqslant h_0, \quad p \in P, \\ h_1q_1 + h_2q_2 + \cdots + h_dq_d &\geqslant h_0, \quad q \in Q. \end{aligned}$$

As we see from Figure 11.1, such $h_0, h_1, \ldots, h_d$ may or may not exist. How can we find out?

11.2 Linear Programming

The problem of testing for linear separability of point sets is a special case of the general linear programming problem.

Definition 11.2 *Given $n, d \in \mathbb{N}$ and real numers*

$$\begin{aligned} b_i &, \quad i = 1, \ldots, n, \\ c_j &, \quad j = 1, \ldots, d, \\ a_{ij} &, \quad i = 1, \ldots, n, \quad j = 1, \ldots, d, \end{aligned}$$

the linear program *defined by these numbers is the problem of finding real numbers* $x_1, x_2, \ldots, x_d$ *such that*

(i) $\sum_{j=1}^{d} a_{ij} x_j \leqslant b_i, \quad i = 1, \ldots, n$, *and*

(ii) $\sum_{j=1}^{d} c_j x_j$ *is as large as possible subject to (i).*

Let us get a geometric intuition: each of the n *constraints* in (i) requires $x = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ to be in the positive halfspace of some hyperplane. The intersection of all these halfspaces is the *feasible region* of the linear program. If it is empty, there is no solution—the linear program is called *infeasible*.

Otherwise—and now (ii) enters the picture—we are looking for a *feasible solution* x (a point inside the feasible region) that maximizes $\sum_{j=1}^{d} c_j x_j$. For every possible value γ of this sum, the feasible solutions for which the sum attains this value are contained in the hyperplane

$$\{x \in \mathbb{R}^d : \sum_{j=1}^{d} c_j x_j = \gamma\}$$

with normal vector $c = (c_1, \ldots, c_d)$. Increasing γ means to shift the hyperplane into direction c. The highest γ is thus obtained from the hyperplane that is most extreme in direction c among all hyperplanes that intersect the feasible region, see Figure 11.3.

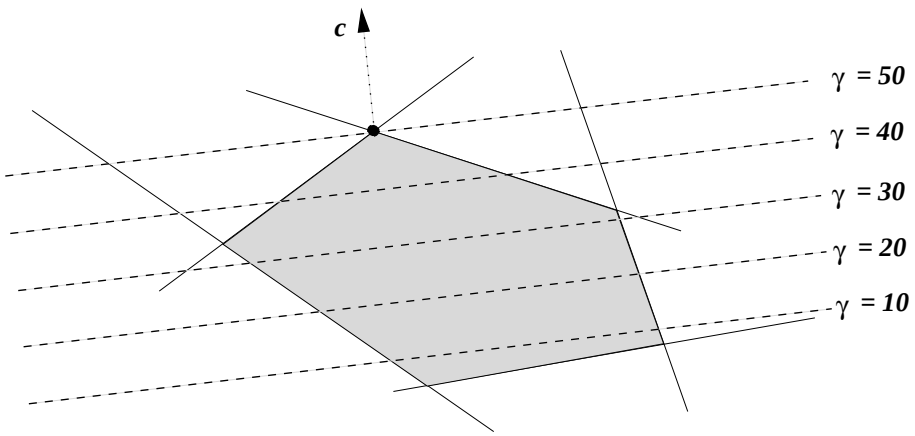


Figure 11.3: *A linear program: finding the feasible solution in the intersection of five positive halfspaces that is most extreme in direction* c *(has highest value* $\gamma = \sum_{j=1}^{d} c_j x_j$*)*

In Figure 11.3, we do have an *optimal solution* (a feasible solution x of highest value $\sum_{j=1}^{d} c_j x_j$), but in general, there might be feasible solutions of arbitrarily high γ-value. In this case, the linear program is called *unbounded*, see Figure 11.4.

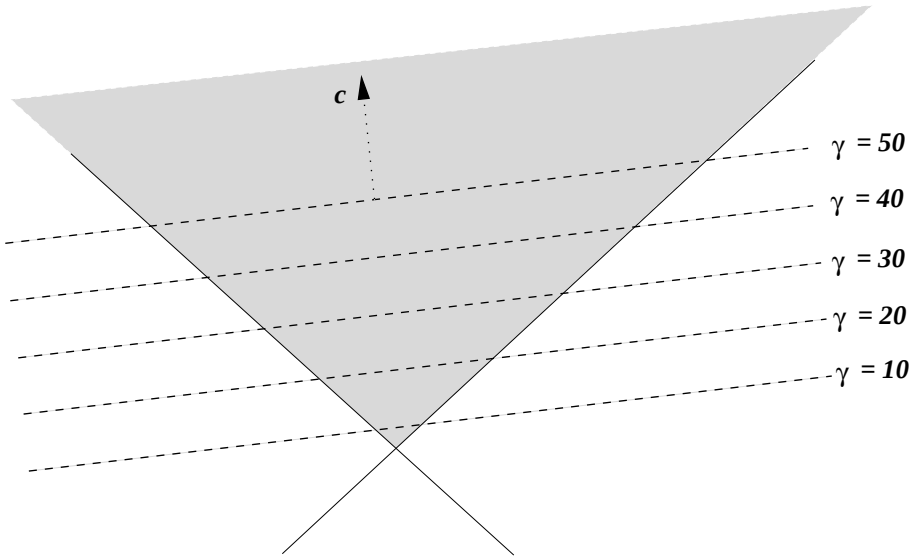


Figure 11.4: *An unbounded linear program*

It can be shown that infeasibility and unboundedness are the only obstacles for the existence of an optimal solution. If the linear program is feasible and bounded, there exists an optimal solution.

This is not entirely trivial, though. To appreciate the statement, consider the problem of finding a point (x, y) that (i) satisfies $y \geqslant e^x$ and (ii) has smallest value of y among all (x, y) that satisfy (i). This is not a linear program, but in the above sense it is feasible (there are (x, y) that satisfy (i)) and bounded (y is bounded below from 0 over the set of feasible solutions). Still, there is no optimal solution, since values of y arbitrarily close to 0 can be attained but not 0 itself.

Even if a linear program has an optimal solution, it is in general not unique. For example, if you rotate c in Figure 11.3 such that it becomes orthogonal to the top-right edge of the feasible region, then every point of this edge is an optimal solution. Why is this called a *linear* program? Because all constraints are linear inequalities, and the objective function is a linear function. There is also a reason why it is called a linear *program*, but we won't get into this here (see [3] for more background).

Using vector and matrix notation, a linear program can succinctly be written as follows.

$$\begin{array}{lll} \text{(LP)} & \text{maximize} & c^T x \\ & \text{subject to} & Ax \leqslant b \end{array}$$

Here, $c, x \in \mathbb{R}^d$, $b \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times d}$, and $\cdot^T$ denotes the transpose operation. The inequality "$\leqslant$" is interpreted componentwise. The vector x represents the *variables*, c is called the *objective function vector*, b the *right-hand side*, and A the *constraint matrix*.

To *solve* a linear programs means to either report that the problem is infeasible or unbounded, or to compute an optimal solution x^*. If we can solve linear programs, we can also decide linear separability of point sets. For this, we check whether the linear program

$$\begin{array}{ll} \text{maximize} & 0 \\ \text{subject to} & h_1 p_1 + h_2 p_2 + \cdots + h_d p_d - h_0 \leqslant 0, \quad p \in P, \\ & h_1 q_1 + h_2 q_2 + \cdots + h_d q_d - h_0 \geqslant 0, \quad q \in Q. \end{array}$$

in the $d+1$ variables $h_0, h_1, h_2, \ldots, h_d$ and objective function vector $c = 0$ is feasible or not. The fact that some inequalities are of the form "$\geqslant$" is no problem, of course, since we can multiply an inequality by -1 to turn "$\geqslant$" into "$\leqslant$".

11.3 Minimum-area Enclosing Annulus

Here is another geometric problem that we can write as a linear program, although this is less obvious. Given a point set $P \subseteq \mathbb{R}^2$, find a *minimum-area annulus* (region between two concentric circles) that contains P; see Figure 11.5 for an illustration.

The optimal annulus can be used to test whether the point set P is (approximately) on a common circle which is the case if the annulus has zero (or small) area.

Let us write this as an optimization problem in the variables $c = (c_1, c_2) \in \mathbb{R}^2$ (the center) and $r, R \in \mathbb{R}$ (the small and the large radius).

$$\begin{array}{ll} \text{minimize} & \pi(R^2 - r^2) \\ \text{subject to} & r^2 \leqslant \|p - c\|^2 \leqslant R^2, \quad p \in P. \end{array}$$

This neither has a linear objective function nor are the constraints linear inequalities. But a variable substitution will take care of this. We define new variables

$$u := r^2 - \|c\|^2, \tag{11.3}$$

$$v := R^2 - \|c\|^2. \tag{11.4}$$

Omitting the factor π in the objective function does not affect the optimal solution (only its value), hence we can equivalently work with the objective function $v - u = R^2 - r^2$. The constraint $r^2 \leqslant \|p - c\|^2$ is equivalent to $r^2 \leqslant \|p\|^2 - 2p^T c + \|c\|^2$, or

$$u + 2p^T c \leqslant \|p\|^2.$$

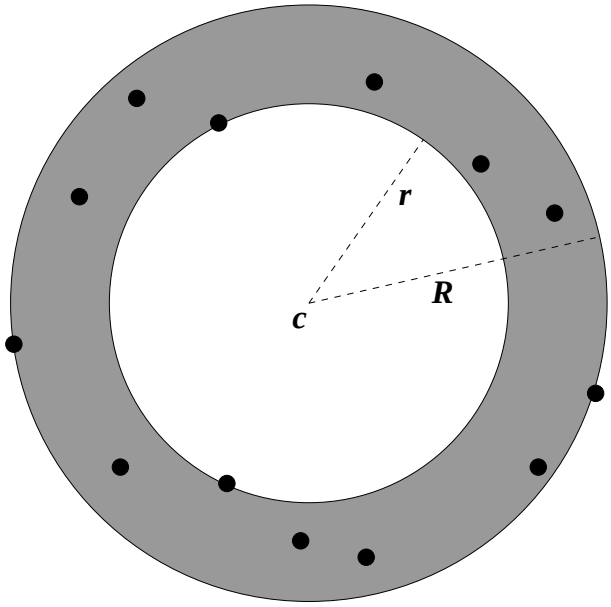


Figure 11.5: *A minimum-area annulus containing* P

In the same way, $\|p - c\| \leqslant R$ turns out to be equivalent to

$$\nu + 2p^T c \geqslant \|p\|^2.$$

This means, we now have a *linear* program in the variables u, ν, c_1, c_2:

$$\begin{array}{ll} \text{maximize} & u - \nu \\ \text{subject to} & u + 2p^T c \leqslant \|p\|^2, \quad p \in P \\ & \nu + 2p^T c \geqslant \|p\|^2, \quad p \in P. \end{array}$$

From optimal values for u, ν and c, we can also reconstruct r^2 and R^2 via (11.3) and (11.4). It cannot happen that r^2 obtained in this way is negative: since we have $r^2 \leqslant \|p - c\|^2$ for all p, we could still increase u (and hence r^2 to at least 0), which is a contradicition to $u - \nu$ being maximal.

Exercise 11.5 *a) Prove that the problem of finding a largest disk inside a convex polygon can be formulated as a linear program! What is the number of variables in your linear program?*

b) Prove that the problem of testing whether a simple polygon is starshaped can be formulated as a linear program.

Exercise 11.6 *Given a simple polygon* P *as a list of vertices along its boundary. Describe a linear time algorithm to decide whether* P *is star-shaped and—if so—to construct the so-called* kernel *of* P*, that is, the set of all star-points.*

11.4 Solving a Linear Program

Linear programming was first studied in the 1930's -1950's, and some of its original applications were of a military nature. In the 1950's, Dantzig invented the *simplex method* for solving linear programs, a method that is fast in practice but is not known to come with any theoretical guarantees [1].

The computational complexity of solving linear programs was unresolved until 1979 when Leonid Khachiyan discovered a polynomial-time algorithm known as the *ellipsoid method* [2]. This result even made it into the New York times.

From a computational geometry point of view, linear programs with a *fixed* number of variables are of particular interest (see our two applications above, with d and 4 variables, respectively, where d may be 2 or 3 in some relavant cases). As was first shown by Megiddo, such linear programs can be solved in time $O(n)$, where n is the number of constraints [4]. In the next lecture, we will describe a much simpler randomized $O(n)$ algorithm due to Seidel [5].

Questions

51. *What is a linear program?* Give a precise definition! How can you visualize a linear program? What does it mean that the linear program is infeasible / unbounded?

52. *Show an application of linear programming!* Describe a geometric problem that can be formulated as a linear program, and give that formulation!

References

[1] G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.

[2] L. G. Khachiyan, Polynomial algorithms in linear programming, *U.S.S.R. Comput. Math. and Math. Phys* **20** (1980), 53–72.

[3] J. Matoušek and B. Gärtner, *Understanding and Using Linear Programming*, Universitext, Springer-Verlag, 2007.

[4] N. Megiddo, Linear programming in linear time when the dimension is fixed, *J. ACM* **31** (1984), 114–127.

[5] R. Seidel, Small-dimensional linear programming and convex hulls made easy, *Discrete Comput. Geom.* **6** (1991), 423–434.

Chapter 12

A randomized Algorithm for Linear Programming

Let us recall the setup from last lecture: we have a linear program of the form

$$\begin{array}{lll} \text{(LP)} & \text{maximize} & c^T x \\ & \text{subject to} & Ax \leqslant b, \end{array} \tag{12.1}$$

where $c, x \in \mathbb{R}^d$ (there are d variables), $b \in \mathbb{R}^n$ (there are n constraints), and $A \in \mathbb{R}^{n \times d}$. The scenario that we are interested in here is that d is a (small) constant, while n tends to infinity.

The goal of this lecture is to present a randomized algorithm (due to Seidel [2]) for solving a linear program whose expected runtime is $O(n)$. The constant behind the big-O will depend exponentially on d, meaning that this algorithm is practical only for small values of d.

To prepare the ground, let us first get rid of the unboundedness issue. We add to our linear program a set of $2d$ constraints

$$-M \leqslant x_i \leqslant M, \quad i = 1, 2, \ldots d, \tag{12.2}$$

where M is a symbolic constant assumed to be larger than any real number that it is compared with. Formally, this can be done by computing with rational functions in M (quotients of polynomials of degree d in the "variable "M), rather than real numbers. The original problem is bounded if and only if the solution of the new (and bounded) problem does not depend on M. This is called the *big-M method*.

Now let H, $|H| = n$, denote the set of original constraints. For $h \in H$, we write the corresponding constraint as $a_h x \leqslant b_h$.

Definition 12.3 *For $Q, R \subseteq H$, $Q \cap R = \emptyset$, let $x^*(Q, R)$ denote the lexicographically largest optimal solution of the linear program*

$$\begin{array}{lll} \textit{LP(Q,R)} & \textit{maximize} & c^T x \\ & \textit{subject to} & a_h x \leqslant b_h, \qquad h \in Q \\ & & a_h x = b_h, \qquad h \in R \\ & & -M \leqslant x_i \leqslant M, \quad i = 1, 2, \ldots, d. \end{array}$$

If this linear program has no feasible solution, we set $x^*(Q, R) = \infty$.

What does this mean? We delete some of the original constraints (the ones not in $Q \cup R$, and we require some of the constraints to hold with equality (the ones in R). Since every linear equation $a_h x = b_h$ can be simulated by two linear inequalities $a_h x \leqslant b_h$ and $a_h x \geqslant b_h$, this again assumes the form of a linear program. By the big-M method, this linear program is bounded, but it may be infeasible. If it is feasible, there may be several optimal solutions, but choosing the lexicographically largest one leads to a unique solution $x^*(Q, R)$.

Our algorithm will compute $x^*(H, \emptyset)$, the lexicographically largest optimal solution of (12.1) subject to the additional bounding-box constraint (12.2). We also assume that $x^*(H, \emptyset) \neq \infty$, meaning that (12.1) is feasible. At the expense of solving an auxiliary problem with one extra variable, this may be assumed w.l.o.g. (Exercise).

Exercise 12.4 *Suppose that you have an algorithm $\mathcal{A}$ for solving* feasible *linear programs of the form*

$$\begin{array}{lll} \textit{(LP)} & \textit{maximize} & c^T x \\ & \textit{subject to} & Ax \leqslant b, \end{array}$$

where feasible means that there exists $\tilde{x} \in \mathbb{R}^d$ such that $A\tilde{x} \leqslant b$. Extend algorithm $\mathcal{A}$ such that it can deal with arbitrary (not necessarily feasible) linear programs of the above form.

12.1 Helly's Theorem

A crucial ingredient of the algorithm's analysis is that the optimal solution $x^*(H, \emptyset)$ is already determined by a constant number (at most d) of constraints. More generally, the following holds.

Lemma 12.5 *Let $Q, R \subseteq H, Q \cap R = \emptyset$, such that the constraints in R are* independent. *This means that the set $\{x \in \mathbb{R}^d : a_h x = b_h, h \in R\}$ has dimension $d - |R|$.*

If $x^*(Q, R) \neq \infty$, then there exists $S \subseteq Q, |S| \leqslant d - |R|$ such that

$$x^*(S, R) = x^*(Q, R).$$

The proof uses *Helly's Theorem*, a classic result in convexity theory.

Theorem 12.6 (Helly's Theorem [1]) *Let $C_1, \ldots C_n$ be $n \geqslant d + 1$ convex subsets of $\mathbb{R}^d$. If any $d + 1$ of the sets have a nonempty common intersection, then the common intersection of* all n *sets is nonempty.*

Even in $\mathbb{R}^1$, this is not entirely obvious. There it says that for every set of intervals with pairwise nonempty overlap there is one point contained in all the intervals. We will not prove Helly's Theorem here but just use it to prove Lemma 12.5.

Proof. (**Lemma 12.5**) The statement is trivial for $|Q| \leqslant d - |R|$, so assume $|Q| > d - |R|$. Let

$$L(R) := \{x \in \mathbb{R}^d : a_h x = b_h, h \in R\}$$

and

$$B := \{x \in \mathbb{R}^d : -M \leqslant x_i \leqslant M, i = 1, \ldots, d\}.$$

For a vector $x = (x_1, \ldots, x_d)$, we define $x_0 = c^T x$, and we write $x > x'$ if $(x_0, x_1, \ldots, x_d)$ is lexicographically larger than $(x'_0, x'_1, \ldots, x'_d)$.

Let $x^* = x^*(Q, R)$ and consider the $|Q| + 1$ sets

$$C_h = \{x \in \mathbb{R}^d : a_h x \leqslant b_h\} \cap B \cap L(R), \quad h \in Q$$

and

$$C_0 = \{x \in \mathbb{R}^d : x > x^*\} \cap B \cap L(R).$$

The first observation (that may require a little thinking in case of C_0) is that all these sets are convex. The second observation is that their common intersection is empty. Indeed, any point in the common intersection would be a feasible solution $\tilde{x}$ of $LP(Q, R)$ with $\tilde{x} > x^* = x^*(Q, R)$, a contradiction to $x^*(Q, R)$ being the lexicographically largest optimal solution of $LP(Q, R)$. The third observation is that since $L(R)$ has dimension $d - |R|$, we can after an affine transformation assume that all our $|Q| + 1$ convex sets are actually convex subsets of $R^{d-|R|}$.

Then, applying Helly's Theorem yields a subset of $d - |R| + 1$ constraints with an empty common intersection. Since all the C_h do have $x^*(Q, R)$ in common, this set of constraints must contain C_0. This means, there is $S \subseteq Q, |S| = d - |R|$ such that

$$x \in C_h \ \forall h \in S \quad \Rightarrow \quad x \notin C_0.$$

In particular, $x^*(S, R) \in C_h$ for all $h \in S$, and so it follows that $x^*(S, R) \leqslant x^* = x^*(Q, R)$. But since $S \subseteq Q$, we also have $x^*(S, R) \geqslant x^*(Q, R)$, and $x^*(S, R) = x^*(Q, R)$ follows. □

12.2 Convexity, once more

We need a second property of linear programs on top of Lemma 12.5; it is also a consequence of convexity of the constraints, but a much simpler one.

Lemma 12.7 *Let* $Q, R \subseteq H, Q \cap R \neq \emptyset$ *and* $x^*(Q, R) \neq \infty$. *Let* $h \in Q$. *If*

$$a_h x^*(Q \setminus \{h\}, R) > b_h,$$

then

$$x^*(Q, R) = x^*(Q \setminus \{h\}, R \cup \{h\}).$$

Before we prove this, let us get an intuition. The vector $x^*(Q \setminus \{h\})$ is the optimal solution of $LP(Q \setminus \{h\}, R)$. And the inequality $a_h x^*(Q \setminus \{h\}, R) > b_h$ means that the constraint h is violated by this solution. The implication of the lemma is that at the optimal solution of $LP(Q, R)$, the constraint h must be satisfied with equality in which case this optimal solution is at the same time the optimal solution of the more restricted problem $LP(Q \setminus \{h\}, R \cup \{h\})$.

Proof. Let us suppose for a contradition that

$$a_h x^*(Q, R) < b_h$$

and consider the line segment s that connects $x^*(Q, R)$ with $x^*(Q \setminus \{h\}, R)$. By the previous strict inequality, we can make a small step (starting from $x^*(Q, R)$) along this line segment without violating the constraint h (Figure 12.1). And since both $x^*(Q, R)$ as well as $x^*(Q \setminus \{h\}, R)$ satisfy all other constraints in $(Q \setminus \{h\}, R)$, convexity of the constraints implies that this small step takes us to a feasible solution of $LP(Q, R)$ again. But this solution is lexicographically larger than $x^*(Q, R)$, since we move towards the lexicographically larger vector $x^*(Q \setminus \{h\}, R)$; this is a contradiction. □

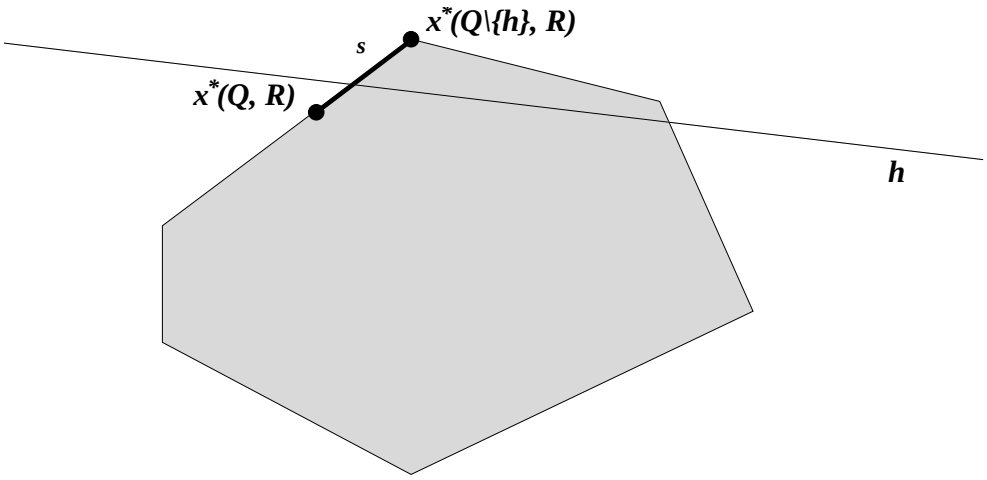


Figure 12.1: *Proof of Lemma 12.7*

12.3 The Algorithm

The algorithm reduces the computation of $x^*(H, \emptyset)$ to the computation of $x^*(Q, R)$ for various sets Q, R, where R is an *independent* set of constraints. Suppose you want to compute $x^*(Q, R)$ (assuming that $x^*(Q, R) \neq \infty$). If $Q = \emptyset$, this is "easy", since we have a constant-size problem, defined by R with $|R| \leqslant d$ and $2d$ bounding-box constraints $-M \leqslant x_i \leqslant M$.

Otherwise, we choose $h \in Q$ and recursively compute $x^*(Q \setminus \{h\}, R) \neq \infty$. We then check whether constraint h is violated by this solution. If not, we are done, since then $x^*(Q \setminus \{h\}, R) = x^*(Q, R)$ (Think about why!). But if h is violated, we can use Lemma 12.7 to conclude that $x^*(Q, R) = x^*(Q \setminus \{h\}, R \cup \{h\})$, and we recursively compute the latter solution. Here is the complete pseudocode.

```
LP(Q, R):
    IF Q = ∅ THEN
        RETURN x*(∅, R)
    ELSE
        choose h ∈ Q uniformly at random
        x* := LP(Q \ {h}, R)
        IF a_h x* ⩽ b_h THEN
            RETURN x*
        ELSE
            RETURN LP(Q \ {h}, R ∪ {h})
        END
    END
```

To solve the original problem, we call this algorithm with $\mathcal{LP}(H, \emptyset)$. It is clear that the algorithm terminates since the first argument Q becomes smaller in every recursive call. It is also true (Exercise) that every set R that comes up during this algorithm is indeed an independent set of constraints and in particular has at most d elements. The correctness of the algorithm then follows from Lemma 12.7.

Exercise 12.8 *Prove that all sets R of constraints that arise during a call to algorithm $\mathcal{LP}(H, \emptyset)$ are independent, meaning that the set*

$$\{x \in \mathbb{R}^d : a_h x = b_h, h \in R\}$$

of points that satisfy all constraints in R with equality has dimension $d - |R|$.

12.4 Runtime Analysis

Now we get to the analysis of algorithm LP, and this will also reveal why the random choice is important.

We will analyze the algorithm in terms of the expected number of *violation tests* $a_h x^* \leqslant b_h$, and in terms of the expected number of *basis computations* $x^*(\emptyset, R)$ that it performs. This is a good measure, since these are the dominating operations of the algorithm. Moreover, both violation test and basis computation are "cheap" operations in the sense that they can be performed in time $f(d)$ for some f.

More specifically, a violation test can be performed in time $O(d)$; the time needed for a basis computation is less clear, since it amounts to solving a small linear program itself. Let us suppose that it is done by brute-force enumeration of all vertices of the bounded polyhedron defined by the at most $3d$ (in)equalities

$$a_h x = b_h, h \in R$$

and

$$-M \leqslant x_i \leqslant M, i = 1, \ldots, d.$$

12.4.1 Violation Tests

Lemma 12.9 *Let $T(n, j)$ be the maximum expected number of violation tests performed in a call to $\mathcal{LP}(Q, R)$ with $|Q| = n$ and $|R| = d - j$. For all $j = 0, \ldots, d$,*

$$\begin{aligned} T(0, j) &= 0 \\ T(n, j) &\leqslant T(n-1, j) + 1 + \frac{j}{n} T(n-1, j-1), \quad n > 0. \end{aligned}$$

Note that in case of $j = 0$, we may get a negative argument on the right-hand side, but due to the factor $0/n$, this does not matter.

Proof. If $|Q| = \emptyset$, there is no violation test. Otherwise, we recursively call $\mathcal{LP}(Q \setminus \{h\}, R)$ for some h which requires at most $T(n-1, j)$ violation tests in expectation. Then there is one violation test within the call to $\mathcal{LP}(Q, R)$ itself, and depending on the outcome, we perform a second recursive call $\mathcal{LP}(Q \setminus \{h\}, R \cup \{h\})$ which requires an expected number of at most $T(n-1, j-1)$ violation tests. The crucial fact is that the probability of performing a second recursive call is at most j/n.

To see this, fix some $S \subseteq Q, |S| \leqslant d - |R| = j$ such that $x^*(Q, R) = x^*(S, R)$. Such a set S exists by Lemma 12.5. This means, we can remove from Q all constraints except the ones in S, without changing the solution.

If $h \notin S$, we thus have

$$x^*(Q, R) = x^*(Q \setminus \{h\}, R),$$

meaning that we have already found $x^*(Q, R)$ after the first recursive call; in particular, we will then have $a_h x^* \leqslant b_h$, and there is no second recursive call. Only if $h \in S$ (and this happens with probability $|S|/n \leqslant j/n$), there can be a second recursive call. □

The following can easily be verified by induction.

Theorem 12.10

$$T(n, j) \leqslant \sum_{i=0}^{j} \frac{1}{i!} j! n.$$

Since $\sum_{i=0}^{j} \frac{1}{i!} \leqslant \sum_{i=0}^{\infty} \frac{1}{i!} = e$, we have $T(n, j) = O(j!n)$. If $d \geqslant j$ is constant, this is $O(n)$.

12.4.2 Basis Computations

To count the basis computations, we proceed as in Lemma 12.9, except that the "1" now moves to a different place.

Lemma 12.11 *Let* $B(n,j)$ *be the maximum expected number of basis computations performed in a call to* $\mathcal{LP}(Q,R)$ *with* $|Q| = n$ *and* $|R| = d-j$*. For all* $j = 0,\ldots,d$*,*

$$\begin{aligned} B(0,j) &= 1 \\ B(n,j) &\leqslant B(n-1,j) + \frac{j}{n}B(n-1,j-1), \quad n > 0. \end{aligned}$$

Interestingly, $B(n,j)$ turns out to be much smaller than $T(n,j)$ which is good since a basic computation is much more expensive than a violation test. Here is the bound that we get.

Theorem 12.12

$$B(n,j) \leqslant (1+H_n)^j = O(\log^j n),$$

where H_n *is the* n*-th Harmonic number.*

Proof. This is also a simple induction, but let's do this one since it is not entirely obvious. The statement holds for $n = 0$ with the convention that $H_0 = 0$. It also holds for $j = 0$, since Lemma 12.11 implies $B(n,0) = 1$ for all n. For $n,j > 0$, we inductively obtain

$$\begin{aligned} B(n,j) &\leqslant (1+H_{n-1})^j + \frac{j}{n}(1+H_{n-1})^{j-1} \\ &\leqslant \sum_{k=0}^{j} \binom{j}{k} (1+H_{n-1})^{j-k} (\frac{1}{n})^k \\ &= (1+H_{n-1}+\frac{1}{n})^j = (1+H_n)^j. \end{aligned}$$

The second inequality uses the fact that the terms $(1+H_{n-1})^j$ and $\frac{j}{n}(1+H_{n-1})^{j-1}$ are the first two terms of the sum in the second line. □

12.4.3 The Overall Bound

Putting together Theorems 12.10 and 12.12 (for $j = d$, corresponding to the case $R = \emptyset$), we obtain the following

Theorem 12.13 *A linear program with* n *constraints in* d *variables (*d *a constant) can be solved in time* $O(n)$*.*

Questions

53. *What is Helly's Theorem?* Give a precise statement and outline the application of the theorem for linear programming (Lemma 12.5).

54. *Outline an algorithm for solving linear programs*! Sketch the main steps of the algorithm and the correctness proof! Also explain how one may achieve the preconditions of feasibility and boundedness.

55. *Sketch the analysis of the algorithm!* Explain on an intuitive level how randomization helps, and how the recurrence relations for the expected number of violation tests and basis computations are derived. What is the expected runtime of the algorithm?

References

[1] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*, Springer-Verlag, Heidelberg, West Germany, 1987.

[2] R. Seidel, Small-dimensional linear programming and convex hulls made easy, *Discrete Comput. Geom.* **6** (1991), 423–434.