Understanding and Using Linear Programming

Jiří Matoušek and Bernd Gärtner

Work in progress. Please do not distribute!

Rev. 22/XII/2005

Contents

1 What Is It, and What For? 4
- 1.1 A Linear Program . . . 4
- 1.2 What Can Be Found in This Book . . . 8
- 1.3 Linear Programming and Linear Algebra . . . 9
- 1.4 Significance and History of Linear Programming . . . 10

2 Examples 12
- 2.1 Optimized Diet: Wholesome and Cheap? . . . 12
- 2.2 Network Flow . . . 14
- 2.3 Separation of Points . . . 16
- 2.4 Fitting a Line . . . 17
- 2.5 Disk in a Convex Polygon . . . 19
- 2.6 Example From Paper Industry . . . 20
- 2.7 The General and the Particular . . . 22

3 Integer Programming and LP Relaxation 23
- 3.1 Integer Programming . . . 23
- 3.2 Maximum-Weight Matching . . . 25
- 3.3 Minimum Vertex Cover . . . 29
- 3.4 Maximum Independent Set . . . 31

4 Theory of Linear Programming: First Steps 33
- 4.1 Equational Form . . . 33
- 4.2 Basic Feasible Solutions . . . 35
- 4.3 ABC of Convexity and Convex Polyhedra . . . 40
- 4.4 Vertices and Basic Feasible Solutions . . . 44

5 The Simplex Method 46
- 5.1 An Introductory Example . . . 46
- 5.2 Exception Handling: Unboundedness . . . 50
- 5.3 Exception Handling: Degeneracy . . . 51
- 5.4 Exception Handling: Infeasibility . . . 52
- 5.5 Simplex Tableaus in General . . . 53
- 5.6 The Simplex Method in General . . . 55
- 5.7 Pivot Rules . . . 58
- 5.8 The Struggle Against Cycling . . . 59

5.9 Efficiency of the simplex method 63
5.10 Summary 65

6 Duality of Linear Programming 67
6.1 The Duality Theorem 67
6.2 Dualization for Everyone 70
6.3 A Proof From the Simplex Method 73
6.4 The Farkas Lemma 74
6.5 Farkas Lemma: A Geometric Proof 78
6.6 Farkas Lemma From the Fourier–Motzkin Elimination 81

7 Not Only the Simplex Method 84
7.1 The Ellipsoid Method 84
7.2 Interior point methods 92

8 More Applications 96
8.1 Zero-Sum Games 96
8.2 Matchings and vertex covers in bipartite graphs 104
8.3 Machine Scheduling 111
8.4 Upper Bounds for Codes 117
8.5 Sparse Solutions of Linear Systems 125
8.6 Transversals of d-Intervals 136
8.7 Smallest Balls and Convex Programming 142

9 Software and Further Reading 149

1

What Is It, and What For?

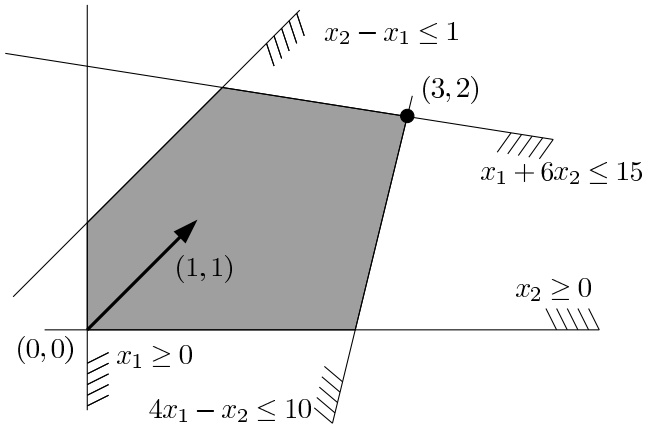
Linear programming, surprisingly, is not directly related to computer programming. The term was introduced in the 1950s when computers were few and mostly top secret, and the word programming was a military term that, at that time, referred to plans or schedules for training, logistical supply, or deployment of men. The word linear suggests that feasible plans are restricted by linear constraints (inequalities), and also that the quality of the plan (e.g., costs or duration) is also measured by a linear function of the considered quantities. In a similar spirit, linear programming soon started to be used for planning all kinds of economic activities, such as transport of raw materials and products among factories, sowing various crop plants, or cutting paper rolls into shorter ones in sizes ordered by customers. The phrase "planning with linear constraints" would perhaps better capture this original meaning of linear programming. However, the term linear programming has been well-established for many years, and at the same time, it has acquired a considerably broader meaning: It doesn't play a role only in mathematical economy, but it appears frequently in computer science and in many other fields.

1.1 A Linear Program

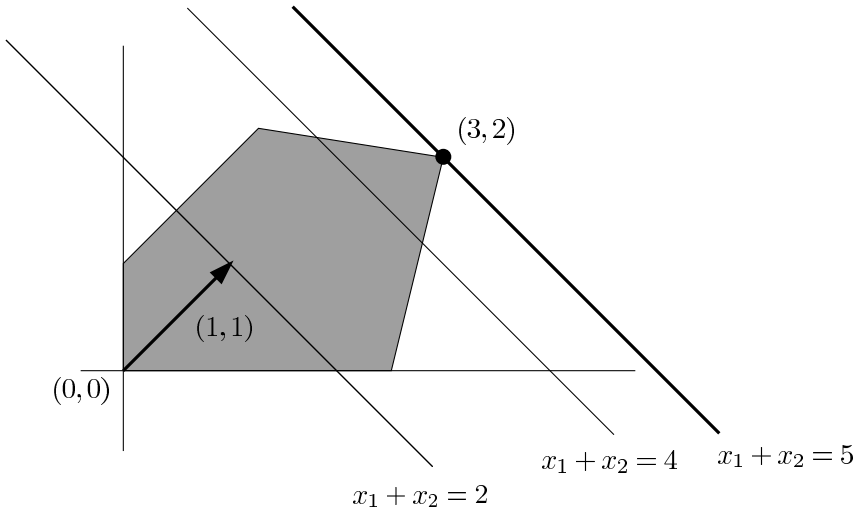
We begin with a very simple linear programming problem (or **linear program** for short):

$$\begin{array}{ll} \text{maximize the value} & x_1 + x_2 \\ \text{among all vectors } (x_1, x_2) \in \mathbb{R}^2 & \\ \text{satisfying the constraints} & x_1 \geq 0 \\ & x_2 \geq 0 \\ & x_2 - x_1 \leq 1 \\ & x_1 + 6x_2 \leq 15 \\ & 4x_1 - x_2 \leq 10. \end{array}$$

For this linear program we can easily draw a picture. The set $\{\mathbf{x} \in \mathbb{R}^2 : x_2 - x_1 \leq 1\}$ is the half-plane lying below the line $x_2 = x_1 + 1$, and similarly, each of the remaining four inequalities defines a half-plane. The set of all vectors satisfying the five constraints simultaneously is a convex polygon:



Which point of this polygon maximizes the value of $x_1 + x_2$? The one lying "farthest in the direction" of the vector $(1,1)$ drawn by the arrow; that is, the point $(3,2)$. The phrase "farthest in the direction" is in quotation marks since it is not quite precise. More precisely, we are seeking a point where a line perpendicular to the arrow intersects our polygon the last time when we translate it in the direction of the arrow. (Note that the function $x_1 + x_2$ is constant on each line perpendicular to the vector $(1,1)$, and as we move the line in the direction of that vector, the value of the function increases.) See the next illustration:



In a general linear program we want to find a vector $\mathbf{x}^* \in \mathbb{R}^n$ maximizing (or minimizing) the value of a given linear function among all vectors $\mathbf{x} \in \mathbb{R}^n$ satisfying a given system of linear equations and inequalities. The linear function to be maximized, or sometimes minimized, is called the **objective function**. It has the form $\mathbf{c}^T\mathbf{x} = c_1x_1 + \cdots + c_nx_n$, where $\mathbf{c} \in \mathbb{R}^n$ is a given vector.[1] The linear equations and inequalities in the linear program are called the **constraints**. It is customary to denote the number of constraints by m.

[1]We regard $\mathbf{c}$ as a column vector, or also as an $n\times1$ matrix, and we make a similar convention for all vectors in this book.

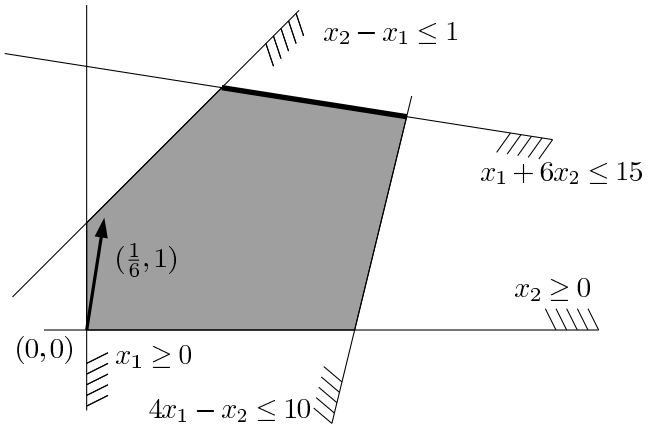
A linear program is often written using matrices and vectors, in a way similar to the notation $A\mathbf{x} = \mathbf{b}$ for a system of linear equations in linear algebra. To make such a notation simpler, we can replace each equation in the linear program by two opposite inequalities. For example, instead of the constraint $x_1 + 3x_2 = 7$ we can put the two constraints $x_1 + 3x_2 \leq 7$ and $x_1 + 3x_2 \geq 7$. Moreover, the direction of the inequalities can be reversed by changing the signs: $x_1 + 3x_2 \geq 7$ is equivalent to $-x_1 - 3x_2 \leq -7$, and thus we can assume that all inequality signs are "$\leq$", say. Finally, minimizing an objective function $\mathbf{c}^T\mathbf{x}$ is equivalent to maximizing $\mathbf{-c}^T\mathbf{x}$, and hence we can always pass to a maximization problem. After such modifications each linear program can be expressed as follows:

$$\begin{array}{ll} \text{maximize the value of} & \mathbf{c}^T\mathbf{x} \\ \text{among all vectors } \mathbf{x} \in \mathbb{R}^n \text{ satisfying} & A\mathbf{x} \leq \mathbf{b}, \end{array}$$

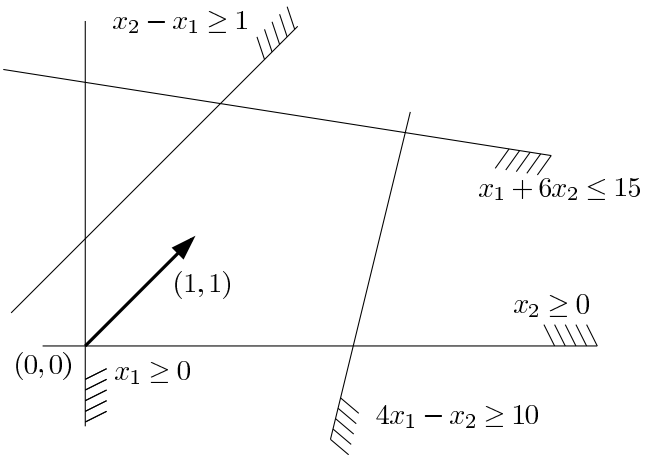
where A is a given $m \times n$ real matrix and $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$ are given vectors. Here the relation $\leq$ holds for two vectors of equal length if and only if it holds componentwise.

Any vector $\mathbf{x} \in \mathbb{R}^n$ satisfying all constraints of a given linear program is a **feasible solution**. Each $\mathbf{x}^* \in \mathbb{R}^n$ that gives the maximum possible value of $\mathbf{c}^T\mathbf{x}$ among all feasible $\mathbf{x}$ is called an **optimal solution** or **optimum** for short. In our linear program above we have $n = 2$, $m = 5$, and $\mathbf{c} = (1, 1)$. The only optimal solution is the vector $(3, 2)$, while, for instance, $(2, \frac{3}{2})$ is a feasible solution that is not optimal.

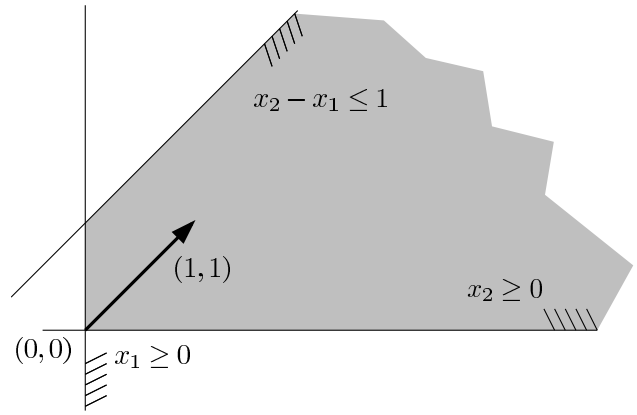
A linear program may in general have a single optimal solution, or infinitely many optimal solutions, or none at all. We have seen a situation with a single optimal solution in the first example of a linear program. If we change the vector $\mathbf{c}$ in the example to $(\frac{1}{6}, 1)$, all points on the side of the polygon drawn thick in the next picture are optimal solutions:



If we reverse the directions of the inequalities in the constraints $x_2 - x_1 \leq 1$ and $4x_1 - x_2 \leq 10$ in our first example, we obtain a linear program that has no feasible solution, and hence no optimal solution either:



Finally, an optimal solution need not exist even when there are feasible solutions. This happens when the objective function can attain arbitrarily large values (such a linear program is called **unbounded**). This is the case when we remove the constraints $4x_1 - x_2 \leq 10$ and $x_1 + 6x_2 \leq 15$ from the initial example:



Later we will prove that no essentially different situations can occur.

We have solved the initial linear program graphically. It was easy since there are only two variables. However, for a linear program with four variables we won't even be able to make a picture, let alone to find an optimal solution graphically. A substantial linear program in practice often has several thousands variables, rather than two or four. A graphical illustration is useful for understanding the notions and procedures of linear programming, but as a computational method it is worthless. Sometimes it may even be misleading, since objects in high dimension may behave in a way quite different from what the intuition gained in the plane or in three-dimensional space suggests.

One of the key pieces of knowledge about linear programming that one should remember forever is this:

A linear program is efficiently solvable, both in theory and in practice.

(In order that the piece of knowledge makes sense, one must not forget what a linear program is.)

- *In practice* a number of software packages are available. They can handle inputs with several thousands of variables and constraints. Linear programs with a special structure, for example, with a small number of nonzero coefficients in each constraint, can often be managed even with a much larger number of variables and constraints.
- *In theory*, algorithms have been developed that provably solve each linear program in time bounded by a certain polynomial function of the input size. The input size is measured as the total number of bits needed to write down all coefficients in the objective function and in all the constraints.

These two statements summarize the results of a long and strenuous research, and efficient methods for linear programming are not simple.

1.2 What Can Be Found in This Book

The rest of Chapter 1 briefly discusses history and importance of linear programming and connects it to linear algebra.

For a large majority of readers it can be expected that, whenever they encounter linear programming in practice or in research, they will be using it as a black box. From this point of view Chapter 2 is crucial, since it describes a number of algorithmic problems that can be solved via linear programming. The closely related Chapter 3 discusses integer programming, in which one also optimizes a linear function over a set of vectors determined by linear constraints, but moreover, the variables have to attain only integer values. In this context we will see how linear programming can help in approximate solution of hard computational problems.

Chapter 4 brings basic theoretical results on linear programming and on the geometric structure of the set of all feasible solutions. Notions introduced there, such as convexity and convex polyhedron, are important in many other branches of mathematics and computer science as well.

The next chapter covers the simplex method, which is a fundamental algorithm for linear programming. With all details it is relatively complicated and from the contemporary point of view it is not indispensable in a first course on linear programming, although some traditional introductions to linear programming are focused almost solely on the simplex method.

In Chapter 6 we will state and prove the Duality Theorem, which is one of the principal theoretical results in linear programming and an extremely useful tool for proofs.

Chapter 7 deals with two other important algorithmic approaches to linear programming: the ellipsoid method and the interior point method. Both of them are rather intricate and we only outline the basic ideas.

Chapter 8 collect several slightly more advanced applications of linear programming from various fields, each with motivation and some background material.

Two levels of text. This discourse should mainly serve as an introductory text for undergraduate and early graduate students, and so we do not want to assume previous knowledge beyond the usual basic undergraduate courses. However, many of the key results in linear programming, which would be a pity to omit, are not easy to prove, and sometimes they use mathematical methods whose knowledge cannot be expected at undergraduate level. Consequently, the text is divided into two levels. On the basic level we are aiming at full and sufficiently detailed proofs.

> The second, more advanced, and "edifying" level is typographically distinguished like this. In such parts, intended chiefly for mathematically more mature readers, say graduate or PhD students, we include sketches of proofs and somewhat imprecise formulations of more advanced results. Whoever finds these passages incomprehensible may freely ignore them; the basic text should also make sense without them.

1.3 Linear Programming and Linear Algebra

The basics of linear algebra can be regarded as a theory of systems of linear equations. Linear algebra considers many other things as well, but systems of linear equations are surely one of the core subjects. A key algorithm is the Gaussian elimination, which efficiently finds a solution of such a system, and even a description of the set of all solutions. Geometrically, the solution set is an affine subspace[2] of $\mathbb{R}^n$, which is an important linear-algebraic notion.

In a similar spirit, the discipline of linear programming can be regarded as a theory of systems of linear *inequalities*.

> In a linear program this is somewhat obscured by the fact that we do not look for an arbitrary solution of the given system of inequalities, but rather a solution maximizing a given objective function. But it can be shown that finding an (arbitrary) feasible solution of a linear program, if one exists, is computationally almost equally difficult as finding an optimal solution. Let us outline how one can gain an optimal solution, provided that feasible solutions can be computed (a different and more elegant way will be described in Section 6.1). If we somehow know in advance that, for instance, the maximum value of the objective function in a given linear program lies between 0 and 100, we can first ask, whether there exists a feasible $\mathbf{x} \in \mathbb{R}^n$ for which the objective function is at least 50. That is, we add to the existing constraints a new constraint requiring that the value of the objective function be at least 50, and we find out whether this auxiliary linear program has a feasible solution. If yes, we will further ask, by the same trick, whether the objective function can be at least 75, and if not, we will check whether it can be at least 25. A reader with computer-science conditioned reflexes has probably already recognized the strategy of *binary search*, which allows us to quickly localize the maximum value of the objective function with great accuracy.

Geometrically, the set of all solutions of a system of linear inequalities is an intersection of finitely many half-spaces in $\mathbb{R}^n$. Such a set is called a *convex polyhedron*, and familiar examples of convex polyhedra in $\mathbb{R}^3$ are a cube, a rectangular box, a tetrahedron, and a regular dodecahedron. Convex polyhedra

[2] An affine subspace is a vector subspace translated by a fixed vector $\mathbf{x} \in \mathbb{R}^n$. For example, affine subspaces of $\mathbb{R}^2$ are all points, all lines, and $\mathbb{R}^2$.

are mathematically much more complex objects than vector subspaces or affine subspaces (we will return to this later). So, actually, we can be grateful for the objective function in a linear program: It is enough to compute a single point $\mathbf{x}^* \in \mathbb{R}^n$ as a solution and we need not worry about the whole polyhedron.

In linear programming, a role comparable to the one of the Gauss elimination in linear algebra is played by the *simplex method*. It is an algorithm for solving linear programs, usually quite efficient, and it also allows one to prove theoretical results.

Let us summarize the analogies between linear algebra and linear programming in a tabular form:

	Basic problem	Algorithm	Solution set
Linear algebra	system of linear equations	Gauss elimination	affine subspace
Linear programming	system of linear inequalities	simplex method	convex polyhedron

1.4 Significance and History of Linear Programming

In a special issue of the journal *Computing in Science & Engineering*, the simplex method was included among "the ten algorithms with the greatest influence on the development and practice of science and engineering in the 20th century."[3] Although some may argue that the simplex method is only number fourteen, say, and although each such evaluation is necessarily subjective, the importance of linear programming can hardly be cast in doubt.

The simplex method was invented and developed by George Dantzig in 1947, based on his work for the U. S. Air Force. Even sooner, in 1939, Leonid Vitalyevich Kantorovich was charged with reorganization of timber industry in the U. S. S. R. and as a part of his task he formulated a restricted class of linear programs and a method for their solution. As it happens under such regimes, his discoveries went almost unnoticed and nobody continued his work. Kantorovich together with Tjalling Koopmans received the Nobel Prize in Economy in 1975, for pioneering work in resource allocation. Somewhat ironically Dantzig, whose contribution to linear programming is no doubt much more significant, has never been awarded a Nobel Prize; his work was probably considered too mathematical for a prize in economy.

The discovery of the simplex method had a great impact on both theory and practice in economy. Linear programming was used to allocate resources, plan production, schedule workers, plan investment portfolios and formulate marketing and military strategies. Even entrepreneurs and managers accustomed to relying on their experience and intuition were impressed when the costs were

[3]The remaining nine algorithms on this list are: the Metropolis algorithm for Monte Carlo simulations, the Krylov subspace iteration methods, the decompositional approach to matrix computations, the Fortran optimizing compiler, the QR algorithm for computing eigenvalues, the Quicksort algorithm for sorting, the fast Fourier transform, the detection of integer relations, and the fast multipole method.

cut by 20%, say, by a mere reorganization according to some mysterious calculation. Especially when such feat was accomplished by someone who was not really familiar with the company, just on the basis of some numerical data. Suddenly, mathematical methods could no longer be ignored with impunity in a competitive environment.

Linear programming has evolved a great deal since the 1940s, and new types of applications have been found, by far not restricted to mathematical economy.

In theoretical computer science it has become one of the fundamental tools in algorithm design. For a number of computational problems the existence of an efficient (polynomial-time) algorithm was first established by general techniques based on linear programming.

For other problems, known to be computationally difficult (NP-hard, if this term tells the reader anything), finding an exact solution is often hopeless. One looks for approximate algorithms, and linear programming is a key component of the most powerful known methods.

Another surprising application of linear programming is theoretical: *the Duality Theorem*, which will be explained in Chapter 6, appears in proofs of numerous mathematical statements, most notably in combinatorics, and it provides an unifying abstract view of many seemingly unrelated results. The Duality Theorem is also significant algorithmically.

We will show examples of methods for constructing algorithms and proofs based on linear programming, but many other results of this kind are too advanced for a short introductory text like ours.

The theory of algorithms for linear programming itself has also grown considerably. As everybody knows, today's computers are many orders of magnitude faster than those fifty years ago, and so it doesn't sound surprising that much larger linear programs can be solved today. But it may be surprising that this enlargement of manageable problems probably owes more to theoretical progress in algorithms than to by faster computers. Some of the new computational methods will be sketched in Chapter 7.

2

Examples

Linear programming is a wonderful tool. But in order to use it, one first has to start suspecting that the considered computational problem might be expressible by a linear program, and then one has to really express it that way. Neither of these steps needs to be straightforward, while the solution of the resulting linear program is usually routine. In this chapter we present at least a small part of a wide spectrum of problems amenable to linear programming methods, and we demonstrate a few tricks for reformulating problems that do not look like linear programs at first sight. More examples will be mentioned in Chapter 3.

2.1 Optimized Diet: Wholesome and Cheap?

> *... and when Rabbit said, "Honey or condensed milk with your bread?" he was so excited that he said, "Both," and then, so as not to seem greedy, he added, "But don't bother about the bread, please."*
>
> *A. A. Milne, Winnie the Pooh*

The Nutrition Inspection of the EU revealed that dishes served at the dining and beverage facility "Bullneck's", such as herrings, hot dogs, and house-style hamburgers, do not comport with the new nutritional regulations, and its report mentioned explicitly the lack of vitamins A and C and dietary fiber. The owner and operator of the aforementioned facility tries to rectify these shortcomings by augmenting the dishes by vegetable side-dishes, which he intends to combine from white cabbage, carrot, and a stockpile of soused cucumber discovered in the cellar. The following table summarizes the numerical data: the prescribed amount of the vitamins and fiber per dish, their contents in the foods, and unit prices of the foods.[1]

[1] For those interested in healthy diet: the vitamin contents and other data are more or less realistic.

Food	Carrot, Raw	White Cabbage, Raw	Cucumber, Soused	Required per dish
Vitamin A [mg/kg]	35	0.5	0.5	0.5 mg
Vitamin C [mg/kg]	60	300	10	15 mg
Dietary Fiber [g/kg]	30	20	10	4 g
price [€/kg]	0.75	0.5	0.15*	—

*Residual accounting price of the inventory, most likely unsaleable.

At what minimum additional price per dish can the requirements of the Nutrition Inspection be satisfied? This question can be expressed by the following linear program:

$$\begin{array}{ll} \text{minimize} & 0.75x_1 + 0.5x_2 + 0.15x_3 \\ \text{subject to} & x_1 \geq 0 \\ & x_2 \geq 0 \\ & x_3 \geq 0 \\ & 35x_1 + 0.5x_2 + 0.5x_3 \geq 0.5 \\ & 60x_1 + 300x_2 + 10x_3 \geq 15 \\ & 30x_1 + 20x_2 + 10x_3 \geq 4. \end{array}$$

The variable x_1 specifies the amount of carrot to be added to each dish, and similarly for x_2 (cabbage) and x_3 (cucumber). The objective function expresses the price of the combination. The amounts of carrot, cabbage, and cucumber are always nonnegative, which is captured by the conditions $x_1 \geq 0$, $x_2 \geq 0$, $x_3 \geq 0$ (if we didn't include them, an optimal solution might perhaps have the amount of carrot, say, negative, by which one would seemingly save money). Finally the inequalities in the last three lines force the requirements on vitamins A and C and of dietary fiber.

The linear program can be solved by standard methods. The optimal solution yields the price of €0.07 with the following doses: carrot 9.5g, cabbage 3.8g, and soused cucumber 290g per dish (all rounded to two significant digits). This probably wouldn't pass another round of inspection. In reality one would have to add further constraints, for example, one on the maximum amount of soused cucumber.

We have included this example so that our treatment doesn't look too revolutionary. As it seems, all introductions to linear programming begin with various dietary problems, most likely because the first large-scale problem on which the simplex method was tested in 1947 was the determination of an adequate diet of least cost. Which foods should be combined and in what amounts so that the required amounts of all essential nutrients are satisfied and the daily ration is the cheapest possible. The linear program had 77 variables and 9 constraints, and its solution by the simplex method using hand-operated desk calculators took approximately 120 man-days.

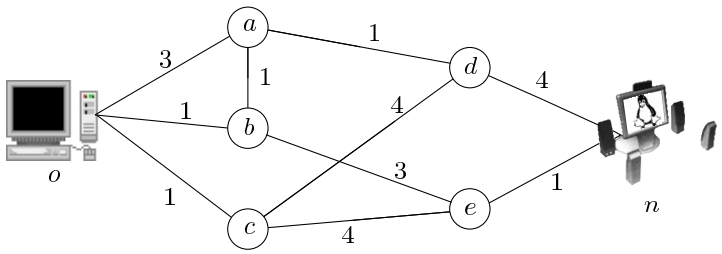
Later on, when George Dantzig had already gained access to an electronic computer, he tried to optimize his own diet as well. The optimal solution of the first linear program that he constructed recommended daily consumption of several liters of vinegar. When he removed vinegar from the next input, he obtained approximately 200 bouillon cubes as the basis of the daily diet.

This story, whose truth is not entirely out of question, doesn't diminish the power of linear programming in any way, but it illustrates how difficult it is to capture mathematically all important aspects of real-life problems. In the realm of nutrition, for example, it is not clear up to the present, what exactly is the influence of various components of food on the human body. (Although, of course, many things *are* clear, and hopes that science of the future will recommend hamburgers as the main ingredient of healthy diet will almost surely be disappointed.) Even if it were known perfectly, few people want and can formulate exactly what they expect from their diet—apparently, it is much easier to formulate such requirements for the diet of someone else. Moreover, there are nonlinear dependencies among the effects of various nutrients, and so the dietary problem can never be captured perfectly by linear programming.

There are many applications of linear programming in industry, agriculture, services, etc. that, from an abstract point of view, are variations of the diet problem and do not bring substantial new mathematical tricks. It may be still challenging to design good models for real-life problems of this kind, but the challenges are not mathematical. We will not dwell on such problems here (many examples can be found in Chvátal's book cited in Chapter 9), and we will present problems where the use of linear programming has different flavors.

2.2 Network Flow

An administrator of a computer network convinced his employer to purchase a new computer with an improved sound system. He wants to transfer his music collection from an old computer to the new one, using a local network. The network looks like this:



What is the maximum transfer rate from computer o (old) to computer n (new)? The numbers near each data link specify the maximum transfer rate of that link (in Mb/s, say). The nodes $a, b, \ldots, e$ are not suitable for storing substantial amounts of data, and hence all data entering them has to be sent further immediately. From this we can already see that the maximum transfer rate cannot be used on all links simultaneously (consider the node a, for example). Thus we have to find an appropriate value of the data flow for each link so that the total transfer rate from o to n is maximum.

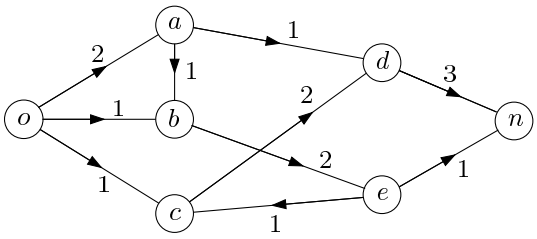
For every link in the network we introduce one variable. For example, x_{be} specifies the rate by which data is transfered from b to e. Here x_{be} can also be negative, which means that data flow in the opposite direction, from e to b.

(And we thus do *not* introduce another variable x_{eb}, which would correspond to transfer rate from e to b.) There are 10 variables x_{oa}, x_{ob}, x_{oc}, x_{ab}, x_{ad}, x_{be}, x_{cd}, x_{ce}, x_{dn}, and x_{en}. We set up the following linear program:

$$\begin{array}{lll}
\text{maximize} & x_{oa} + x_{ob} + x_{oc} & \\
\text{subject to} & -3 \le x_{oa} \le 3, \quad -1 \le x_{ob} \le 1, & -1 \le x_{oc} \le 1 \\
& -1 \le x_{ab} \le 1, \quad -1 \le x_{ad} \le 1, & -3 \le x_{be} \le 3 \\
& -4 \le x_{cd} \le 4, \quad -4 \le x_{ce} \le 4, & -4 \le x_{dn} \le 4 \\
& -1 \le x_{en} \le 1 & \\
& x_{oa} = x_{ab} + x_{ad} & \\
& x_{ob} + x_{ab} = x_{be} & \\
& x_{oc} = x_{cd} + x_{ce} & \\
& x_{ad} + x_{cd} = x_{dn} & \\
& x_{be} + x_{ce} = x_{en}. &
\end{array}$$

The objective function $x_{oa} + x_{ob} + x_{oc}$ expresses the total rate by which data is sent out from computer o. Since it is neither stored nor lost (hopefully) anywhere, it has to be received at n at the same rate. The next 10 constraints, $-3 \le x_{oa} \le 3$ through $-1 \le x_{en} \le 1$, restrict the transfer rates along the individual links. The remaining constraints say that whatever enters each of the nodes a through e has to leave immediately.

The optimal solution of this linear program is depicted below:



The number near each link is the transfer rate on that link, and the arrow determines the direction of the data flow. Note that between c and e data has to be sent in the direction from e to c, and hence $x_{ce} = -1$. The optimum value of the objective function is 4, and this is the desired maximum transfer rate.

In this example it is easy to see that the transfer rate cannot be larger, since the total capacity of all links connecting the computers o and a to the rest of the network equals 4. This is a special case of a remarkable theorem on maximum flow and minimum cut, which is usually discussed in courses on graph algorithms (see also Section 8.2).

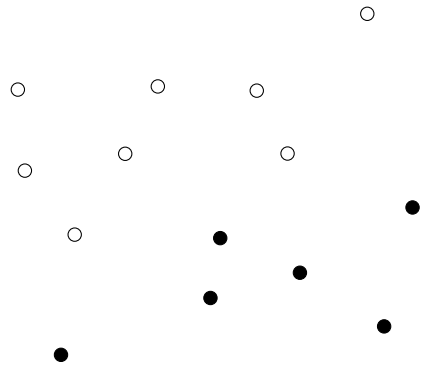
Our example of data flow in a network is small and simple. In practice, however, flows are considered in intricate networks, sometimes even with many source nodes and sink nodes. These can be electrical networks (current flows), road or railroad networks (cars or trains flow), telephone networks (voice or data signals flow), financial (money flows), and so on.

Historically, the network flow problem was first formulated by American military experts in search of efficient ways of disrupting the railway system of the Soviet block; see

A. Schrijver: On the history of the transportation and maximum flow problems, *Math. Programming Ser. B* 91(2002) 437–445.

2.3 Separation of Points

We are given m white points $p_1, p_2, \ldots, p_m$ and n black points $q_1, q_2, \ldots, q_n$ in the plane.



We would like to find out whether there exists a line having all white points on one side and all black points on the other side (none of the points should lie on the line).

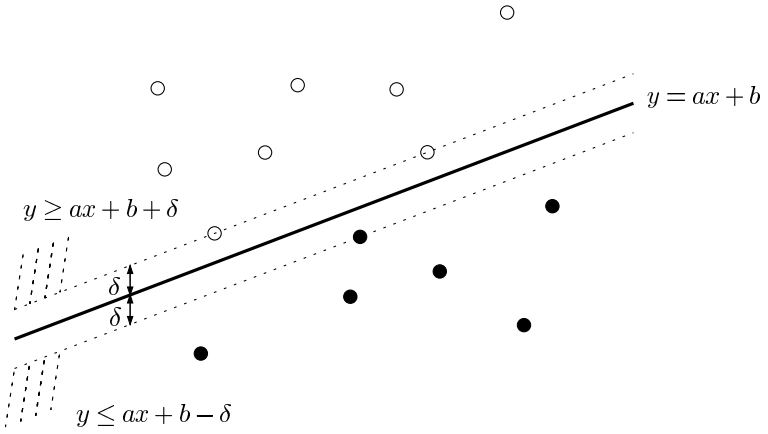
In a solution of this question by linear programming we distinguish three cases. First we test whether there exists a *vertical* line with the required property. This case needs neither linear programming nor particular cleverness.

The next case is the existence of a line that is not vertical and that has all black points below it and all white points above it. Let us write the equation of such a line as $y = ax + b$, where a and b are some yet unknown real numbers. A point r with coordinates $x(r)$ and $y(r)$ lies above this line if $y(r) > a \cdot x(r) + b$, and it lies below it if $y(r) < a \cdot x(r) + b$. So a suitable line exists if and only if the following system of inequalities with variables a and b has a solution:

$$\begin{aligned} y(p_i) &> a \cdot x(p_i) + b \qquad \text{for } i = 1, 2, \ldots, m \\ y(q_j) &< a \cdot x(q_j) + b \qquad \text{for } j = 1, 2, \ldots, n. \end{aligned}$$

We haven't yet mentioned strict inequalities in connection with linear programming, and actually, they are not allowed in linear programs. But here we can get around this issue by a small trick: We introduce a new variable δ, which stands for the "gap" between the left and right sides of each strict inequality. Then we will try to make the gap as large as possible:

$$\begin{array}{lll} \text{maximize} & \delta & \\ \text{subject to} & y(p_i) \geq a \cdot x(p_i) + b + \delta & \text{for } i = 1, 2, \ldots, m \\ & y(q_j) \leq a \cdot x(q_j) + b - \delta & \text{for } j = 1, 2, \ldots, n. \end{array}$$



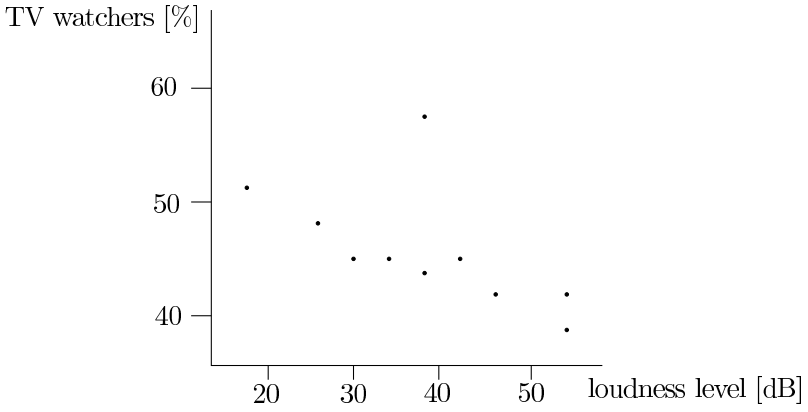
This linear program has three variables a, b, and δ. The optimal δ is positive exactly if the preceding system of strict inequalities has a solution, and the latter happens exactly if a nonvertical line exists with all black points below and all white points above.

Similarly we can deal with the third case, namely the existence of a nonvertical line having all black points above it and all white points below it. This completes the description of an algorithm for the line separation problem.

A plane separating two point sets in $\mathbb{R}^3$ can be computed by the same approach, and we can also solve the analogous problem in higher dimensions.

2.4 Fitting a Line

The loudness level of nightingale singing was measured every evening for a number of days in a row, and the percentage of people watching main TV news was surveyed by questionnaires. The following diagram plots the measured values by points in the plane:



The simplest dependencies are linear, and many dependencies can be well approximated by a linear function. We thus want to find a line that best fits the measured points. (Who feels that this example is not sufficiently realistic can recall some measurements in physics labs, where the measured quantities should actually obey an exact linear dependence.)

How can one formulate mathematically that a given line "best fits" the points? There is no unique way, and several different criteria are commonly used for line fitting in practice. The most popular one is the method of *least squares*, which for given points $(x_1, y_1), \ldots, (x_n, y_n)$ seeks a line with equation $y = ax + b$ minimizing the expression

$$\sum_{i=1}^{n} (ax_i + b - y_i)^2. \tag{2.1}$$

In words, for every point we take its vertical distance from the line, square it, and sum these "squares of errors."

This method need not always be the most suitable. For instance, if a few exceptional points are measured with very large error, they can influence the resulting line a great deal. An alternative method, less sensitive to a small number of "outliers", is to minimize the sum of absolute values of all errors:

$$\sum_{i=1}^{n} |ax_i + b - y_i|. \tag{2.2}$$

Surprisingly, this can be captured by a linear program:

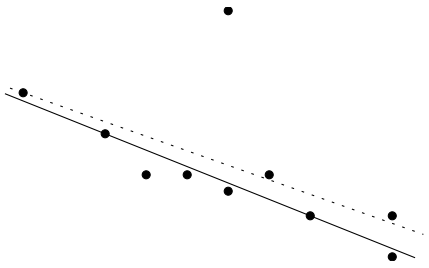
$$\begin{array}{lll} \text{Minimize} & e_1 + e_2 + \cdots + e_n & \\ \text{subject to} & e_i \geq ax_i + b - y_i & \text{for } i = 1, 2, \ldots, n \\ & e_i \geq -(ax_i + b - y_i) & \text{for } i = 1, 2, \ldots, n. \end{array}$$

The variables are a, b, and $e_1, e_2, \ldots, e_n$ (while $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ are given numbers). Each e_i is an auxiliary variable standing for the error at the ith point. The constraints guarantee that

$$e_i \geq \max\Big(ax_i + b - y_i, -(ax_i + b - y_i)\Big) = |ax_i + b - y_i|.$$

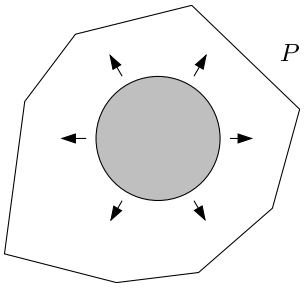
In an optimal solution each of these inequalities have to be satisfied with equality, for otherwise, we could decrease the corresponding e_i. Thus, an optimal solution yields a line minimizing the expression (2.2).

A picture shows a line fitted by this method (solid) and a line fitted using least squares (dotted):

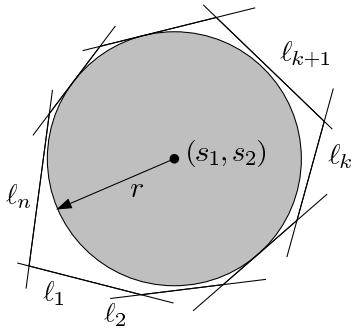


2.5 Disk in a Convex Polygon

Given a convex n-gon P in the plane, we want to find the largest circular disk contained in P.



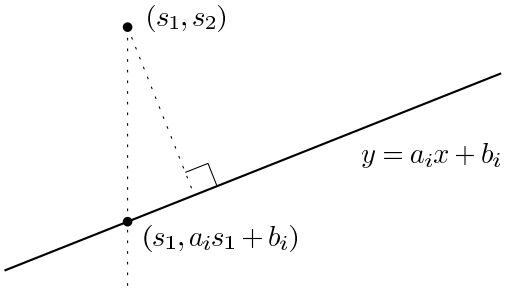
For simplicity let us assume that none of the sides of P is vertical. Let the ith side of P lie on a line ℓ_i with equation $y = a_i x + b_i$, $i = 1, 2, \ldots, n$, and let us choose the numbering of the sides in such a way that the first, second, up until kth side bound P from below, while the $(k+1)$st through nth side bound it from above.



Let us now ask, under what conditions does a circle with center $\mathbf{s} = (s_1, s_2)$ and radius r lie completely inside P? This is if and only if the point $\mathbf{s}$ has distance at least r from each of the lines $\ell_1, \ldots, \ell_n$, lies above the lines $\ell_1, \ldots, \ell_k$, and lies below the lines $\ell_{k+1}, \ldots, \ell_n$. We compute the distance of $\mathbf{s}$ from ℓ_i. A simple calculation using similarity of triangles and the Pythagoras theorem shows that this distance equals the absolute value of the expression

$$\frac{s_2 - a_i s_1 - b_i}{\sqrt{a_i^2 + 1}}.$$

Moreover, the expression is positive if $\mathbf{s}$ lies above ℓ_i, and it is negative if $\mathbf{s}$ lies below ℓ_i.



The disk of radius r centered at $\mathbf{s}$ thus lies inside P exactly if the following system of inequalities is satisfied:

$$\begin{array}{rcll} \frac{s_2 - a_i s_1 - b_i}{\sqrt{a_i^2+1}} & \geq & r, & i = 1, 2, \ldots, k, \text{ and} \\ \frac{s_2 - a_i s_1 - b_i}{\sqrt{a_i^2+1}} & \leq & -r, & i = k+1, k+2, \ldots, n. \end{array}$$

We thus want to find the largest r such that there exist s_1 and s_2 so that all the constraints are satisfied. This yields a linear program! (Some might be frightened by the square roots, but these can be computed in advance, since all the a_i are concrete numbers.)

$$\begin{array}{lrcll} \text{Maximize} & r & & & \\ \text{subject to} & \frac{s_2 - a_i s_1 - b_i}{\sqrt{a_i^2+1}} & \geq & r & \text{for } i = 1, 2, \ldots, k \\ & \frac{s_2 - a_i s_1 - b_i}{\sqrt{a_i^2+1}} & \leq & -r & \text{for } i = k+1, k+2, \ldots, n. \end{array}$$

There are three variables s_1, s_2, and r. An optimal solution yields the desired largest disk contained in P.

A similar problem in higher dimension can be solved analogously. For example, in the three-dimensional space we can ask for the largest ball that can be placed into the intersection of given n half-spaces.

Interestingly, another similar-looking problem, namely, finding the smallest disk containing a given convex n-gon in the plane, cannot be expressed by a linear program and has to be solved differently; see Section 8.7.

2.6 Example From Paper Industry

Here we again have an industrial problem, but the application of linear programming is quite non-obvious here. Moreover, we will naturally encounter an integrality constraint, which will bring us to the topic of the next chapter.

Paper mills usually manufacture rolls of paper of a standard width, say 3 meters. But customers want to buy paper rolls as wide as they like, and the paper mill has to cut such rolls from the 3m rolls. One 3m roll can be cut, for instance, into two rolls 93cm wide, one roll of width 108cm, and a rest of 6cm (which goes to waste).

Let us consider an order of

- 97 rolls of width 135cm,

- 610 rolls of width 108cm,
- 395 rolls of width 93cm, and
- 211 rolls of width 42cm.

What is the smallest number of 3m rolls that have to be cut in order to satisfy this order, and how should they be cut?

In order to engage linear programming one has to be generous in introducing new variables. We write down all widths occurring in the order, i.e. 42cm, 93cm, 108cm, and 135cm. Then we list all possibilities of cutting a 3m paper roll into rolls of some of these lengths (we consider only possibilities where the wasted piece is shorter than 42cm):

P1: 2×135,

P2: $135 + 108 + 42$,

P3: $135 + 93 + 42$,

P4: $135 + 3 \times 42$,

P5: $2 \times 108 + 2 \times 42$,

P6: $108 + 2 \times 93$,

P7: $108 + 93 + 2 \times 42$,

P8: $108 + 4 \times 42$,

P9: $2 \times 93 + 2 \times 42$,

P10: $93 + 4 \times 42$,

P11: 7×42.

For each possibility Pj on the list we introduce a variable x_j representing the number of rolls cut according to that possibility. We want to minimize the total number of rolls cut, i.e. $\sum_{j=1}^{11} x_j$, in such a way that the customers are satisfied. For example, to satisfy the demand for 395 rolls of width 93cm we require

$$x_3 + 2x_6 + x_7 + 2x_9 + x_{10} \geq 395.$$

For each of the widths we obtain one constraint.[2]

The optimal solution of the resulting linear program has $x_1 = 48.5$, $x_5 = 206.25$, $x_6 = 197.5$, and all other components 0. In order to cut 48.5 rolls according to the possibility P1, one has to unwind half of a roll. Here we need

[2]For a more complicated order, the list of possibilities would most likely be produced by computer. We would be in a quite typical situation where a linear program is not entered "by hand," but rather it is generated by some computer program. More advanced techniques even generate the possibilities "on the fly," during the solution of the linear program, which may save time and memory considerably. One can read about this in Chvátal's book cited in Chapter 9, from which this example is taken.

more information about the technical possibilities of the paper mill: Is cutting a fraction of a roll technically and economically feasible? If yes, we have solved the problem optimally. If not, we have to work further and somehow take into account the restriction that only feasible solutions of the linear program with *integral* x_i are of interest. This is not at all easy in general, and it is the subject of Chapter 3.

2.7 The General and the Particular

All of the examples above, as well as hundreds of others, can be expressed as linear programs and thus solved by general algorithms of linear programming. From a programmer's point of view this is very convenient, since it suffices to input the appropriate objective function and constraints into a general-purpose software.

But by far this need not be the computationally most efficient method. For all examples in this chapter with the exception of the diet problem (Section 2.1), specialized algorithms are known that solve them substantially faster. For example, the study of network flows (Section 2.2) constitutes an extensive subfield of theoretical computer science, and fairly efficient algorithms have been developed. Computing a maximum flow via linear programming is thus a wasteful approach.

Linear programming is a general method, which in many cases achieves the goal in a straightforward way. But if we seek a really fast algorithm for some problem, looking for more efficient algorithms is usually worth the effort.

3

Integer Programming and LP Relaxation

3.1 Integer Programming

In Section 2.6 we have encountered a situation where among all feasible solutions of a linear program, only those with all components integral are of interest in the practical application. A similar situation occurs quite often in attempts to apply linear programming, because objects that can be split into arbitrary fractions are more an exception than a rule. When hiring workers, scheduling buses, or cutting paper rolls one somehow has to deal with the fact that workers, buses, and paper rolls occur only in integral quantities.

Sometimes it is enough to simply round the components of an optimal solution to integers, either up, or down, or to the nearest integer. If an optimal solution prescribes cutting 197.5 paper rolls in a certain way, then cutting 198 rolls that way won't change the costs by much. But if an optimal solution specifies that for most of 197 bus lines among villages it is best to schedule something between 0.1 and 0.3 buses, then, clearly, rounding to integers exerts a truly radical influence.

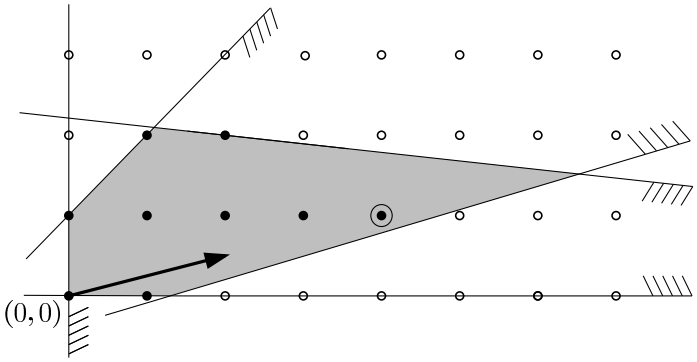
Integer programming. The problem of cutting paper rolls actually leads to a problem with a linear objective function and linear constraints (equations and inequalities), but the variables are allowed to attain only integer values. Such an optimization problem is called an integer program, and after a small adjustment we can write it in a way similar to the one used for a linear program in Chapter 1:

An integer program:

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}^n. \end{array}$$

Here $\mathbb{Z}$ denotes the set of integers and $\mathbb{Z}^n$ is the set of all n-component integer vectors.

The set of all feasible solutions is no longer a convex polyhedron, as was the case for linear programming, but it consists of separate integer points. A picture illustrates a two-dimensional integer program with five constraints:



Feasible solutions are marked with full dots and the optimal solution is designated by a circle. Note that it lies quite far from the optimum of the *linear* program with the same five constraints and the same objective function.

It is known that *solving a general integer program is computationally difficult* (more exactly, it is an *NP-hard problem*), in contrast to solving a linear program. Linear programs with many thousands of variables and constraints can be handled in practice, but an integer program with 50 variables and 100 constraints may be insurmountable even for the most modern computers and software.

> Adding the integrality constraints can thus change the difficulty of a problem in a drastic way indeed. This may not look so surprising anymore if we realize that integer programs can model yes/no decisions, since an integer variable x_j satisfying the linear constraints $0 \leq x_j \leq 1$ has possible values only 0 (no) and 1 (yes). For those familiar with the foundations of NP-completeness it is thus not hard to model the problem of satisfiability of logical formulas by an integer program. In Section 3.4 we will see how an integer program can express the maximum size of an independent set in a given graph, which is also one of the basic NP-complete problems.

Several tools have been developed for solving integer programs. In the literature they can be found under the headings *cutting planes*, *Gröbner bases*, and *branch and bound method* (this is a more general strategy, not restricted to integer programming). The most successful strategies usually employ linear programming as a subroutine for solving certain auxiliary problems. How to do this efficiently is investigated in a branch of mathematics called *polyhedral combinatorics*.

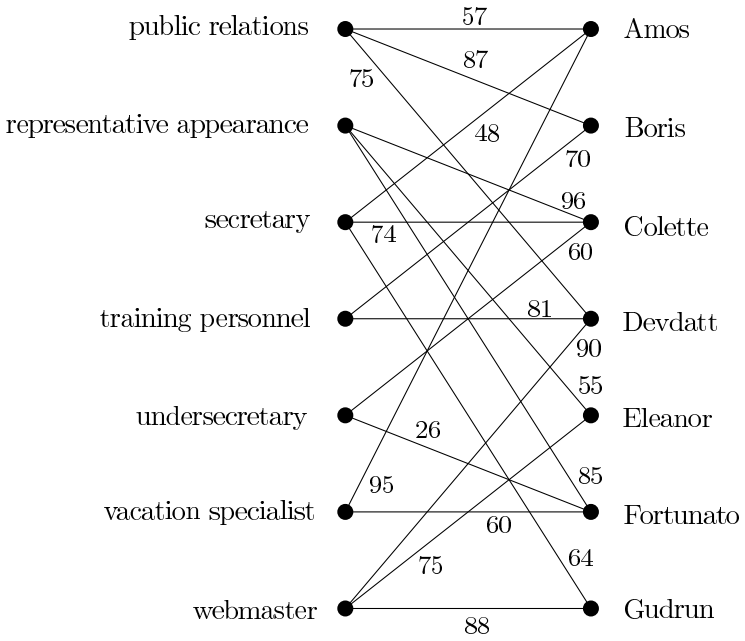
The most widespread use of linear programming today, and one that consumes the largest share of computer time, is most likely in auxiliary computations for integer programs. At least this is claimed by people who should know such things.

Let us remark that there are many optimization problems in which some of the variables are integral, while others may attain arbitrary real values. Then one speaks of *mixed integer programming*. This is in all likelihood the most frequent type of optimization problems in practice.

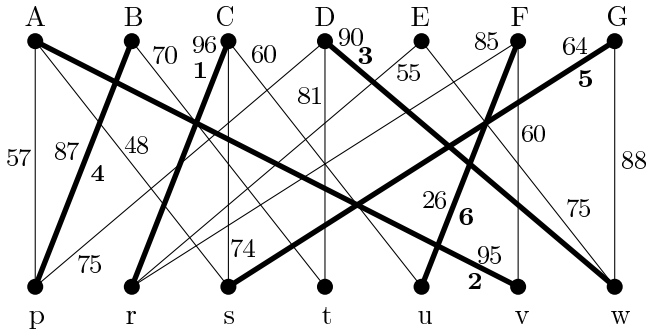
We will demonstrate several important optimization problems that can easily be formulated as integer programs, and we will show how linear programming can or cannot be used in their solution. But it will be only a small sample from this area, which has recently developed extensively and which uses many complicated techniques and clever tricks.

3.2 Maximum-Weight Matching

A company dissolved a department with seven employees in a reorganization, but flexibly enough it created seven new positions. The human resources manager, in order to assign the new positions to the seven employees, conducted interviews with them and gave them extensive questionnaires to fill in. Then he summarized the results in scores: Each employee got a score between 0 and 100 for each of the positions she or he was willing to accept. The manager depicted this information in a scheme, in which an expert can immediately recognize a bipartite graph:



For example, this scheme tells us that Boris is willing to accept the job in public relations, for which he achieved score of 87, or the job in training, for which he has score 70. Now the manager wants to select a position for everyone so that the sum of scores is maximized. The first idea naturally coming to mind is to give everyone the position for which he/she has the largest score. But this cannot be done since, for example, three people are best suited for the profession of webmaster: Eleanor, Gudrun, and Devdatt. If we try to assign the positions by a "greedy" algorithm, meaning that in each step we make an assignment with the largest possible score among those with yet unoccupied positions and still unassigned employees, we end up with filling only 6 positions:



(The bold digits 1–6 indicate the order of making assignments by the greedy algorithm.)

In the language of graph theory we have a bipartite graph with vertex set $V = A \cup B$ and edge set E. Each edge connects a vertex of A to a vertex of B. Moreover, we have $|A| = |B|$. For each edge $e \in E$ we are given a nonnegative weight w_e. We want to find a subset $M \subseteq E$ of edges such that each vertex of both A and B is incident to exactly one edge of M (such an M is called a **perfect matching**), and the sum $\sum_{e \in M} w_e$ is the largest possible.

In order to formulate this problem as an integer program we introduce variables x_e, one for each edge $e \in E$, that can attain values 0 or 1. They will encode the sought after set M: $x_e = 1$ means $e \in M$ and $x_e = 0$ means $e \notin M$. Then $\sum_{e \in M} w_e$ can be written as

$$\sum_{e \in E} w_e x_e,$$

and this is the objective function. The requirement that each vertex $v \in V$ have exactly one incoming edge of M is expressed by the constraints $\sum_{e \in E : v \in e} x_e = 1$. The resulting integer program is

$$\begin{array}{ll} \text{maximize} & \sum_{e \in E} w_e x_e \\ \text{subject to} & \sum_{e \in E : v \in e} x_e = 1 \text{ for each vertex } v \in V, \text{ and} \\ & x_e \in \{0, 1\} \text{ for each edge } e \in E. \end{array} \tag{3.1}$$

If we leave out the integrality conditions, i.e., if we allow each x_e to attain all values in the interval $[0, 1]$, we obtain the following linear program:

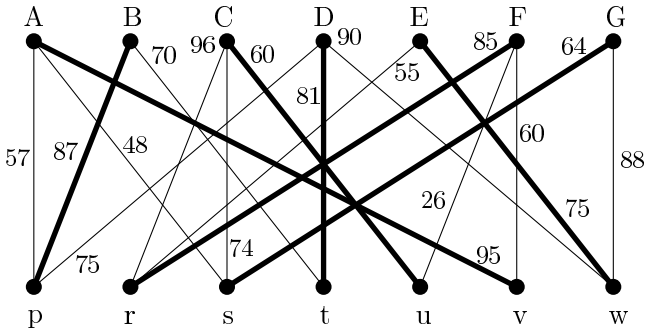
$$\begin{array}{ll} \text{maximize} & \sum_{e \in E} w_e x_e \\ \text{subject to} & \sum_{e \in E : v \in e} x_e = 1 \text{ for each vertex } v \in V, \text{ and} \\ & 0 \le x_e \le 1 \text{ for each edge } e \in E. \end{array}$$

It is called an **LP relaxation** of the integer program (3.1)—we have relaxed the constraints $x_e \in \{0, 1\}$ to the weaker constraints $0 \le x_e \le 1$. We can solve the LP relaxation, say by the simplex method, and we obtain an optimal solution $\mathbf{x}^*$.

What can such an $\mathbf{x}^*$ be good for? Certainly it provides an *upper bound* on the best possible solution of the original integer program (3.1). More precisely, the optimum of the objective function in the integer program (3.1) is bounded above by the value of the objective function at $\mathbf{x}^*$. This is because every feasible

solution of the integer program is also a feasible solution of the LP relaxation, and so we are maximizing over a larger set of vectors in the LP relaxation. An upper bound can be very valuable: For example, if we manage to find a feasible solution of the integer program for which the value of the objective function is 98% of the upper bound, we can usually stop striving, since we know that we cannot improve by more than 2% no matter how hard we try. (Of course, it depends on what we are dealing with; if it is a state budget, even 2% are still worth some effort.)

A pleasant surprise awaits us in the particular problem we are considering here: The LP relaxation not only yields an upper bound, but it provides an optimal solution of the original problem! Namely, if we solve the LP relaxation by the simplex method, we obtain an optimal $\mathbf{x}^*$ that has all components equal to 0 or 1, and thus it determines an optimum perfect matching. (If a better perfect matching existed, it would determine a better solution of the LP relaxation.) The optimal solution thus discovered is drawn below:



The following (nontrivial) theorem shows that things work this nicely for every problem of the considered type.

3.2.1 Theorem. *Let $G = (V, E)$ be an arbitrary bipartite graph with real edge weights w_e. If the LP relaxation of the integer program (3.1) has at least one feasible solution, then it has at least one integral optimal solution. This is an optimal solution for the integer program (3.1) as well.*

An interested reader can find a proof at the end of this section.

The theorem doesn't say that *every* optimal solution of the LP relaxation is necessarily integral. However, the proof gives an easy recipe for producing an integral optimal solution from an arbitrary optimal solution. Moreover, it can be shown that the simplex method always returns an integral optimal solution (using the terminology of Chapter 4, each basic feasible solution is integral).

An LP relaxation can be considered for an arbitrary integer program: The condition $\mathbf{x} \in \mathbb{Z}^n$ is simply replaced by $\mathbf{x} \in \mathbb{R}^n$. Cases where we always get an optimal solution of the integer program from the LP relaxation, such as in Theorem 3.2.1, are rather rare, but LP relaxation can also be useful in other ways.

The maximum of the objective function for the LP relaxation always provides an upper bound for the maximum of the integer program. Sometimes this

bound is quite tight, but at other times it can be very bad; see Section 3.4. The quality of the upper bound from an LP relaxation has been studied for many types of problems. Sometimes one adds new linear constraints to the LP relaxation, which are satisfied by all integral solutions (that is, all feasible solutions of the integer program satisfy these new constraints), but which exclude some of the nonintegral solutions. In this way the upper bound on the optimum of the integer program can often be greatly improved. If we continue adding suitable new constraints for long enough, then we even arrive at an optimal solution of the integer program. This is the main idea of the method of cutting planes.

A nonintegral optimal solution of the LP relaxation can sometimes be converted to an approximately optimal solution of the integer program by rounding. We will see a simple example in Section 3.3.

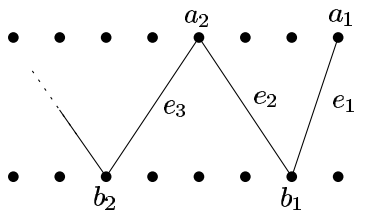
Proof of Theorem 3.2.1. Let $\mathbf{x}^*$ be an optimal solution of the LP relaxation, and let $w(\mathbf{x}^*) = \sum_{e \in E} w_e x_e^*$ be the value of the objective function at $\mathbf{x}^*$. Let us denote the number of nonintegral components of the vector $\mathbf{x}^*$ by $k(\mathbf{x}^*)$.

If $k(\mathbf{x}^*) = 0$, then we are done. For $k(\mathbf{x}^*) > 0$ we describe a procedure that yields another optimal solution $\tilde{\mathbf{x}}$ with $k(\tilde{\mathbf{x}}) < k(\mathbf{x}^*)$. We reach an integral optimal solution by finitely many repetitions of this procedure.

Let $x_{e_1}^*$ be a nonintegral component of the vector $\mathbf{x}^*$, corresponding to some edge $e_1 = \{a_1, b_1\}$. Since $0 < x_{e_1}^* < 1$ and

$$\sum_{e \in E : b_1 \in e} x_e^* = 1,$$

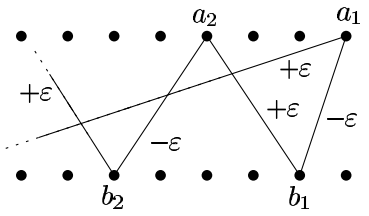
there exists another edge $e_2 = \{a_2, b_1\}$, $a_2 \neq a_1$, with $x_{e_2}^*$ nonintegral. For a similar reason we can also find a third edge $e_3 = \{a_2, b_2\}$ with $0 < x_{e_3}^* < 1$. We continue in this manner and look for nonintegral components along a longer and longer path $(a_1, b_1, a_2, b_2, \ldots)$:



Since the graph G has finitely many vertices, eventually we reach a vertex that we have already visited before. This means that we have found a *cycle* C in which $0 < x_e^* < 1$ for all edges. Since the graph is bipartite, the length t of the cycle C is even. For simplicity of notation let us assume that the edges of C are $e_1, e_2, \ldots, e_t$, although in reality the cycle found as above need not begin with the edge e_1.

Now for a small real number ε we define a vector $\tilde{\mathbf{x}}$ by

$$\tilde{x}_e = \begin{cases} x_e^* - \varepsilon & \text{for } e \in \{e_1, e_3, \ldots, e_{t-1}\} \\ x_e^* + \varepsilon & \text{for } e \in \{e_2, e_4, \ldots, e_t\} \\ x_e^* & \text{otherwise.} \end{cases}$$



It is easy to see that this $\tilde{\mathbf{x}}$ satisfies all the conditions

$$\sum_{e \in E: v \in e} \tilde{x}_e = 1,$$

since at the vertices of the cycle C we have added ε once and subtracted it once, while for all other vertices the weights of the incident edges haven't changed at all. For ε sufficiently small the conditions $0 \leq \tilde{x}_e \leq 1$ are satisfied too, since all components $x^*_{e_i}$ are strictly between 0 and 1. Hence $\tilde{\mathbf{x}}$ is again a feasible solution of the LP relaxation for all sufficiently small ε (positive or negative).

What happens to the value of the objective function? We have

$$w(\tilde{\mathbf{x}}) = \sum_{e \in E} w_e \tilde{x}_e = w(\mathbf{x}^*) + \varepsilon \sum_{i=1}^{t} (-1)^i w_{e_i} = w(\mathbf{x}^*) + \varepsilon \Delta,$$

where we have set $\Delta = \sum_{i=1}^{t} (-1)^i w_{e_i}$. Since $\mathbf{x}^*$ is optimal, necessarily $\Delta = 0$, for otherwise, we could achieve $w(\tilde{\mathbf{x}}) > w(\mathbf{x}^*)$ either by choosing $\varepsilon > 0$ (for $\Delta > 0$) or by choosing $\varepsilon < 0$ (for $\Delta < 0$). This means that $\tilde{\mathbf{x}}$ is an optimal solution whenever it is feasible, i.e., for all ε with a sufficiently small absolute value.

Let us now choose the largest $\varepsilon > 0$ such that $\tilde{\mathbf{x}}$ is still feasible. Then there has to exist $e \in \{e_1, e_2, \ldots, e_t\}$ with $\tilde{x}_e \in \{0, 1\}$, and $\tilde{\mathbf{x}}$ has fewer nonintegral components than $\mathbf{x}^*$. □

Let us now consider another situation, where we have more employees than positions and we want to fill all positions optimally, i.e., so that the sum of scores is maximized. Then we have $|A| < |B|$ in the considered bipartite graph, and for vertices $v \in B$ the condition $\sum_{e \in E: v \in e} x_e = 1$ (every vertex of B is incident to *exactly one* edge of M) is replaced by $\sum_{e \in E: v \in e} x_e \leq 1$ (every vertex of B is incident to *at most one* edge of M). The claim of Theorem 3.2.1 remains valid: If the LP relaxation has a feasible solution, then it also has an integral optimal solution. The proof presented above fails, though. In Section 8.2, we give a proof that works both in this new situation and in the original setting of Theorem 3.2.1. In this context, we will also briefly discuss the nonbipartite case.

3.3 Minimum Vertex Cover

The Internet had been expanding rapidly in the Free Republic of West Mordor, and the government issued a regulation, purely in the interest of improved security of the citizens, that every data link connecting two computers must be equipped by a special device for gathering statistical data about the traffic.

An operator of a part of the network has to attach government's monitoring boxes to some of his computers so that each link has a monitored computer on at least one end. Which computers should get boxes so that the total price is minimum? Let us assume that there is a flat rate per box.

It is again convenient to use graph-theoretic terminology. The computers in the network are vertices and the links are edges. So we have a graph $G = (V, E)$ and we want to find a subset $S \subseteq V$ of the vertices such that each edge has at least one end-vertex in S (such an S is called a **vertex cover**), and S is as small as possible.

This problem can be written as an integer program:

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V} x_v & \\ \text{subject to} & x_u + x_v \geq 1 & \text{for every edge } \{u, v\} \in E, \text{ and} \\ & x_v \in \{0, 1\} & \text{for all } v \in V. \end{array} \tag{3.2}$$

For every vertex v we have a variable x_v, which can attain values 0 or 1. The meaning of $x_v = 1$ is $v \in S$, and $x_v = 0$ means $v \notin S$. The constraint $x_u + x_v \geq 1$ guarantees that the edge $\{u, v\}$ has at least one vertex in S. The objective function is the size of S.

It is known that finding a minimum vertex cover is a computationally difficult (NP-complete) problem. We will describe an approximation algorithm based on linear programming that always finds a vertex cover with at most twice as many vertices as in the smallest possible vertex cover.

An LP relaxation of the above integer program is:

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V} x_v & \\ \text{subject to} & x_u + x_v \geq 1 & \text{for every edge} \\ & & \{u, v\} \in E, \text{ and} \\ & 0 \leq x_v \leq 1 & \text{for all } v \in V. \end{array} \tag{3.3}$$

The first step of the approximation algorithm for vertex cover consists of computing an optimal solution $\mathbf{x}^*$ of this LP relaxation (by some standard algorithm for linear programming). The components of $\mathbf{x}^*$ are real numbers in the interval $[0, 1]$. In the second step we define the set

$$S_{\text{LP}} = \{v \in V : x^*_v \geq \tfrac{1}{2}\}.$$

This is a vertex cover, since for every edge $\{u, v\}$ we have $x^*_u + x^*_v \geq 1$, and so $x^*_u \geq \frac{1}{2}$ or $x^*_v \geq \frac{1}{2}$.

Let S_{OPT} be some vertex cover of the minimum possible size (we don't have it but we can theorize about it). We claim that

$$|S_{\text{LP}}| \leq 2 \cdot |S_{\text{OPT}}|.$$

To see this, let $\bar{\mathbf{x}}$ be an optimal solution of the integer program (3.2), which corresponds to the set S_{OPT}, i.e., $\bar{x}_v = 1$ for $v \in S_{\text{OPT}}$ and $\bar{x}_v = 0$ otherwise. This $\bar{\mathbf{x}}$ is definitely a feasible solution of the LP relaxation (3.3), and so it cannot have a better value of the objective function than an optimal solution $\mathbf{x}^*$:

$$\sum_{v \in V} x^*_v \leq \sum_{v \in V} \bar{x}_v.$$

On the other hand, $|S_{\mathrm{LP}}| \le 2 \cdot \sum_{v \in V} x_v^*$, since $x_v^* \ge \frac{1}{2}$ for each $v \in S_{\mathrm{LP}}$. Therefore

$$|S_{\mathrm{LP}}| \le 2 \cdot \sum_{v \in V} x_v^* \le 2 \cdot \sum_{v \in V} \bar{x}_v = 2 \cdot |S_{\mathrm{OPT}}|.$$

This proof illustrates an important aspect of approximation algorithms: In order to asses the quality of the computed solution, we always need a bound on the quality of the optimal solution, although we don't know it. The LP relaxation provides such a bound, which can sometimes be useful, as in the example of this section. In other problems it may be useless, though, as we will see in the next section.

Remarks. A natural attempt at an approximate solution of the considered problem is again a *greedy algorithm*: Select vertices one by one and always take a vertex that covers the maximum possible number of yet uncovered edges. Although this algorithm may not be bad in most cases, examples can be constructed where it yields a solution at least ten times worse, say, than an optimal solution (and 10 can be replaced by any other constant). Discovering such a construction is a lovely exercise.

There is another, combinatorial approximation algorithm for the minimum vertex cover: Find a maximal matching (that is, an inclusion-maximal set of vertex-disjoint edges) and use its vertices as the vertex cover. It always gives a vertex cover at most twice as big as the optimum, similar to the algorithm explained above.

The algorithm based on linear programming has the advantage of being easy to generalize for a **weighted vertex cover** (the government boxes may have different prices for different computers). In the same way as we did for unit prices one can show that the cost of the computed solution is never larger than twice the optimum cost.

3.4 Maximum Independent Set

Here the authors got tired of inventing imitations of real-life problems, and so we formulate the next problem in the graph theory language right away. For a graph $G = (V, E)$, a set $I \subseteq V$ of vertices is called **independent** (or *stable*) if no two vertices of I are connected by an edge in G.

Computing an independent set with the maximum possible number of vertices for a given graph is one of the notoriously difficult graph-theoretic problems. It can be easily expressed by an integer program:

$$\begin{array}{lll} \text{maximize} & \sum_{v \in V} x_v & \\ \text{subject to} & x_u + x_v \le 1 & \text{for each edge } \{u, v\} \in E, \text{ and} \\ & x_v \in \{0, 1\} & \text{for all } v \in V. \end{array} \tag{3.4}$$

An optimal solution $\mathbf{x}^*$ corresponds to a maximum independent set: $v \in I$ if and only if $x_v^* = 1$. The constraints $x_u + x_v \le 1$ ensure that whenever two vertices u and v are connected by an edge, only one of them can get into I.

In an LP relaxation the condition $x_v \in \{0, 1\}$ is replaced by the inequalities $0 \le x_v \le 1$. The resulting linear program always has a feasible solution with all $x_v = \frac{1}{2}$, which yields objective function equal to $|V|/2$. Hence the optimal value of the objective function is $|V|/2$ or larger.

Let us consider a complete graph on n vertices, in which every two vertices are connected by an edge. The largest independent set consists of a single vertex and thus has size 1. However, as we have seen, the optimal value for the LP relaxation is at least $n/2$. Hence, and this is the point of this section, the LP relaxation behaves in a way completely different from the original integer program.

The complete graph is by no means an isolated case. Dense graphs typically have maximum independent set much smaller than half of the vertices, and so for such graphs, too, an optimal solution of the LP relaxation tells us almost nothing about the maximum independent set.

It is even known that the size of a maximum independent set cannot be approximated well by any reasonably efficient algorithm whatsoever (provided that some widely believed but unproved assumptions hold, such as $\mathrm{P} \neq \mathrm{NP}$). This result is from

J. Håstad: Clique is hard to approximate within $n^{1-\varepsilon}$, *Acta Mathematica* 182(1999) 105–142,

and

`http://www.nada.kth.se/~viggo/problemlist/compendium.html`

is a convenient web site for inapproximability results.

4

Theory of Linear Programming: First Steps

4.1 Equational Form

In the introductory chapter we have explained how each linear program can be converted to the form

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\mathbf{x} \leq \mathbf{b}.$$

But the simplex method requires a different form, which is usually called the *standard form* in the literature. In this text we introduce a less common, but more descriptive term *equational form*. It looks like this:

Equational form of a linear program:

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}. \end{array}$$

As usual, $\mathbf{x}$ is the vector of variables, A is a given $m \times n$ matrix, and $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$ are given vectors. The constraints are thus partly equations, and partly inequalities of a very special form $x_j \geq 0$, the so-called **nonnegativity constraints**. (Warning: although we call this form equational, it contains inequalities as well, and these must not be forgotten!)

Let us emphasize that *all* variables in the equational form have to satisfy the nonnegativity constraints.

In problems encountered in practice we often have nonnegativity constraints automatically, since many quantities, such as the amount of consumed cucumber, cannot be negative.

Transformation of a general linear program to the equational form. We illustrate such a transformation on the linear program

$$\begin{array}{ll} \text{maximize} & 3x_1 - 2x_2 \\ \text{subject to} & 2x_1 - x_2 \leq 4 \\ & x_1 + 3x_2 \geq 5 \\ & x_2 \geq 0. \end{array}$$

We proceed as follows:

1. In order to convert the inequality $2x_1 - x_2 \leq 4$ to an equation, we introduce a new variable x_3, together with the nonnegativity constraint $x_3 \geq 0$, and we replace the considered inequality by the equation $2x_1 - x_2 + x_3 = 4$. The auxiliary variable x_3, which won't appear anywhere else in the transformed linear program, represents the difference between the right-hand side and the left-hand side of the inequality.

2. For the next inequality $x_1 + 3x_2 \geq 5$ we first multiply by -1, which reverses the direction of the inequality. Then we introduce another auxiliary variable x_4, $x_4 \geq 0$, and we replace the inequality by the equation $-x_1 - 3x_2 + x_4 = -5$.

3. We are not finished yet: The variable x_1 in the original linear program is allowed to attain both positive and negative values. We introduce two new, nonnegative variables y_1 and z_1, $y_1 \geq 0$, $z_1 \geq 0$, and we substitute for x_1 the difference $y_1 - z_1$ everywhere. The variable x_1 itself disappears.

The resulting equational form of our linear program is

$$\begin{array}{ll} \text{maximize} & 3y_1 - 3z_1 - 2x_2 \\ \text{subject to} & 2y_1 - 2z_1 - x_2 + x_3 = 4 \\ & -y_1 + z_1 - 3x_2 + x_4 = -5 \\ & y_1 \geq 0,\ z_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0. \end{array}$$

So as to comply with the conventions of the equational form in full, we should still rename the variables to $x_1, x_2, \ldots, x_5$.

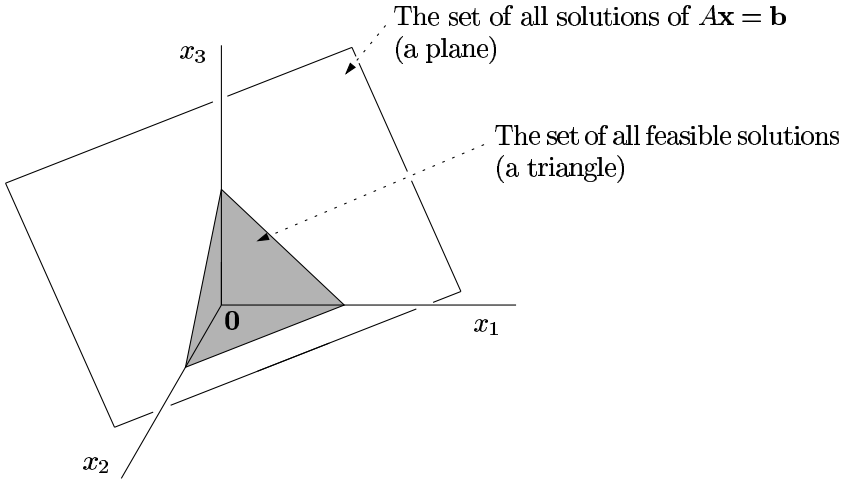
The presented procedure converts an arbitrary linear program with n variables and m constraints into a linear program in equational form with at most $m+2n$ variables and m equations (and, of course, nonnegativity constraints for all variables).

Geometry of a linear program in equational form. Let us consider a linear program in equational form:

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0}.$$

As is derived in linear algebra, the set of all solutions of the system $A\mathbf{x} = \mathbf{b}$ is an affine subspace F of the space $\mathbb{R}^n$ (i.e., a vector subspace translated by some vector $\mathbf{x}_0$). Hence the set of all feasible solutions of the linear program is the intersection of F with the *nonnegative orthant*, i.e., the set of points in $\mathbb{R}^n$ with all coordinates nonnegative.[1] The following picture illustrates the geometry of feasible solutions for a linear program with $n = 3$ variables and $m = 1$ equation, namely the equation $x_1 + x_2 + x_3 = 1$:

[1] In the plane ($n = 2$) this set is called the nonnegative *quadrant*, in $\mathbb{R}^3$ it is the nonnegative *octant*, and the name *orthant* is used for an arbitrary dimension.



(In interesting cases we usually have more than 3 variables and no picture can be drawn.)

A preliminary cleanup. If we change the system $A\mathbf{x} = \mathbf{b}$ by some transformation that preserves the set of solutions, such as a row operation in the Gaussian elimination, it influences neither feasible solutions nor optimal solutions of the linear program. This will be amply used in the simplex method.

From now on we will consider only linear programs in equational form such that

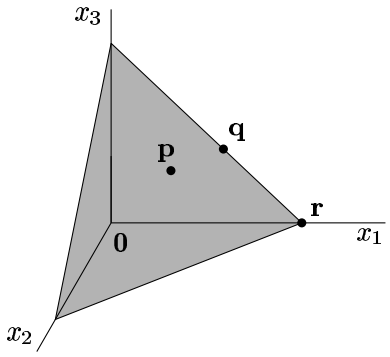
- the system of equations $A\mathbf{x} = \mathbf{b}$ has at least one solution, and
- the rows of the matrix A are linearly independent.

As an explanation of this assumption we need to recall a few facts from linear algebra. Checking whether the system $A\mathbf{x} = \mathbf{b}$ has a solution is easy by Gaussian elimination, and if there is no solution, the considered linear program has no feasible solution either, and we can thus disregard it. (Warning: $A\mathbf{x} = \mathbf{b}$ having a solution doesn't necessarily mean that the linear program has a solution, since a feasible solution has to satisfy $A\mathbf{x} = \mathbf{b}$, and *moreover*, it has to have all components nonnegative.)

If the system $A\mathbf{x} = \mathbf{b}$ has a solution and if some row of A is a linear combination of the other rows, then the corresponding equation is redundant and it can be deleted from the system without changing the set of solutions. We may thus assume that the matrix A has m linearly independent rows and (therefore) rank m.

4.2 Basic Feasible Solutions

Among all feasible solutions of a linear program, a privileged status is granted to so-called basic feasible solutions. In this section we will consider them only for linear programs in equational form. Let us look again at the picture of the set of feasible solutions for a linear program with $n = 3$, $m = 1$:



Among the feasible solutions **p**, **q**, and **r** only **r** is basic. Expressed geometrically and very informally, a basic feasible solution is a tip (corner, spike) of the set of feasible solutions. We will formulate this kind of geometric description of a basic feasible solution later (see Theorem 4.4.1).

The definition that we present next turns out to be equivalent, but it looks rather different. It requires that, very roughly speaking, a basic feasible solution has sufficiently many zero components. Before stating it we introduce a new piece of notation.

In this section A is always a matrix with m rows, $n \geq m$ columns, and rank m. For a subset $B \subseteq \{1, 2, \ldots, n\}$ we let A_B denote the matrix consisting of the columns of A whose indices belong to B. For instance, for

$$A = \begin{pmatrix} 1 & 5 & 3 & 4 & 6 \\ 0 & 1 & 3 & 5 & 6 \end{pmatrix} \text{ and } B = \{2, 4\} \text{ we have } A_B = \begin{pmatrix} 5 & 4 \\ 1 & 5 \end{pmatrix}.$$

We will use a similar notation for vectors; e.g., for $\mathbf{x} = (3, 5, 7, 9, 11)$ and[2] $B = \{2, 4\}$ we have $\mathbf{x}_B = (5, 9)$.

Now we are ready to state a formal definition.

A vector $\mathbf{x} \in \mathbb{R}^n$ is a **basic feasible solution** of the linear program

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0},$$

if $\mathbf{x}$ is a feasible solution and there exists an m-element set $B \subseteq \{1, 2, \ldots, n\}$ such that

- the (square) matrix A_B is nonsingular; i.e., the columns indexed by B are linearly independent, and
- $x_j = 0$ for all $j \notin B$.

For example, $\mathbf{x} = (0, 2, 0, 1, 0)$ is a basic feasible solution for

$$A = \begin{pmatrix} 1 & 5 & 3 & 4 & 6 \\ 0 & 1 & 3 & 5 & 6 \end{pmatrix}, \ \mathbf{b} = \begin{pmatrix} 14 \\ 7 \end{pmatrix}$$

[2]Here, by our conventions, $\mathbf{x}$ is a column vector, so we should really write $\mathbf{x} = (3, 5, 7, 9, 11)^T$. But in order to make the notation simpler, we omit the transposition sign in such cases, since this can hardly lead to misunderstanding.

with $B = \{2, 4\}$.

If such a B is fixed, we call the variables x_j with $j \in B$ the **basic variables**, while the remaining variables are called **nonbasic**. We can thus briefly say that all nonbasic variables are zero in a basic feasible solution.

Let us note that the definition doesn't consider the vector $\mathbf{c}$ at all, and that basic feasible solutions depend solely on A and $\mathbf{b}$.

For some considerations it is convenient to reformulate the definition of a basic feasible solution a little.

4.2.1 Lemma. *A feasible solution $\mathbf{x}$ of a linear program in equational form is basic if and only if the columns of the matrix A_K are linearly independent, where $K = \{j \in \{1, 2, \ldots, n\} : x_j > 0\}$.*

Proof. One of the implications is obvious: If $\mathbf{x}$ is a basic feasible solution and B is the corresponding m-element set as in the definition, then $K \subseteq B$ and thus the columns of the matrix A_K are linearly independent.

Conversely, let $\mathbf{x}$ be feasible and such that the columns of A_K are linearly independent. If $|K| = m$, then we can simply take $B = K$. Otherwise, for $|K| < m$, we extend K to an m-element set B by adding $m - |K|$ more indices so that the columns of A_B are linearly independent. This is possible by the Steinitz Exchange Theorem of linear algebra (in our case we need to complete the linearly independent set of columns of A_K by further columns of A to a basis of the column space of A). □

4.2.2 Proposition. *A basic feasible solution is uniquely determined by the set B. That is, for every m-element set $B \subseteq \{1, 2, \ldots, n\}$ with A_B nonsingular there exists at most one feasible solution $\mathbf{x} \in \mathbb{R}^n$ with $x_j = 0$ for all $j \notin B$.*

Let us stress right away that a single basic feasible solution may be obtained from many different sets B.

Proof of Proposition 4.2.2. For $\mathbf{x}$ to be feasible we must have $A\mathbf{x} = \mathbf{b}$. The left-hand side can be rewritten to $A\mathbf{x} = A_B\mathbf{x}_B + A_N\mathbf{x}_N$, where $N = \{1, 2, \ldots, n\} \setminus B$. By the definition of a basic feasible solution, the vector $\mathbf{x}_N$ of nonbasic variables equals $\mathbf{0}$, and thus the vector $\mathbf{x}_B$ of basic variables satisfies $A_B\mathbf{x}_B = \mathbf{b}$. And here we use the fact that A_B is a nonsingular square matrix: The system $A_B\mathbf{x}_B = \mathbf{b}$ has exactly one solution $\tilde{\mathbf{x}}_B$. If all components of $\tilde{\mathbf{x}}_B$ are nonnegative, then we have exactly one basic feasible solution for the considered B (we amend $\tilde{\mathbf{x}}_B$ by zeros), and otherwise, we have none. □

We introduce the following terminology: We call an m-element set $B \subseteq \{1, 2, \ldots, n\}$ with A_B nonsingular a **basis.**[3] If, moreover, B determines a basic feasible solution, or in other words, if the unique solution of the system $A_B\mathbf{x}_B = \mathbf{b}$ is nonnegative, then we call B a **feasible basis.**

[3] This is a shortcut. The index set B itself is not a basis in the sense of linear algebra, of course. Rather the set of columns of the matrix A_B constitutes a basis of the column space of A.

The following theorem deals with the existence of optimal solutions, and moreover, it shows that it suffices to look for them solely among basic feasible solutions.

4.2.3 Theorem. *Let us consider a linear program in equational form*

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}.$$

(i) *("Optimal solutions may fail to exist only for obvious reasons") If there is at least one feasible solution and the objective function is bounded from above on the set of all feasible solutions, then there exists an optimal solution.*

(ii) *If an optimal solution exists, then there is a basic feasible solutions that is optimal.*

A proof is not necessary for further reading and we defer it to the end of this section. The theorem also follows from the correctness of the simplex method, which will be discussed in the next chapter.

The theorem just stated implies a finite, although entirely impractical, algorithm for solving linear programs in equational form. We consider all m-element subsets $B \subseteq \{1, 2, \ldots, n\}$ one by one, and for each of them we check whether it is a feasible basis, by solving a system of linear equations (we obtain at most one basic feasible solution for each B by Proposition 4.2.2). Then we calculate the maximum of the objective function over all basic feasible solutions found in this way.

Strictly speaking, this algorithm doesn't work if the objective function is unbounded. Formulating a variant of the algorithm that functions properly even in this case, i.e. it reports that the linear program is unbounded, we leave as an exercise. Soon we will discuss the considerably more efficient simplex method, and there we show in detail how to deal with unboundedness.

We have to consider $\binom{n}{m}$ sets B in the above algorithm. For example, for $n = 2m$, the function $\binom{2m}{m}$ grows roughly like 4^m, i.e., exponentially, and this is too much even for moderately large m.

As we will see in Chapter 5, the simplex method also goes through basic feasible solutions, but in a more clever way. It walks from one to another while improving the value of the objective function all the time, until it finds an optimal solution.

Proof of Theorem 4.2.3. We will use some steps that will reappear in the simplex method in a more elaborate form, and so the present proof is a kind of preparation.

We prove the following statement:

If the objective function of a linear program in equational form is bounded above, then for every feasible solution $\mathbf{x}_0$ there exists a basic feasible solution $\tilde{\mathbf{x}}$ with the same or larger value of the objective function; i.e., $\mathbf{c}^T\tilde{\mathbf{x}} \geq \mathbf{c}^T\mathbf{x}_0$.

How does this imply the theorem? If the linear program is feasible and bounded, then according to the statement, for every feasible solution there is a

basic feasible solution with the same or larger objective function. Since there are only finitely many basic feasible solutions, some of them have to give the maximum value of the objective function, which means that they are optimal. We thus get both (i) and (ii) at once.

In order to prove the statement, let us consider an arbitrary feasible solution $\mathbf{x}_0$. Among all feasible solutions $\mathbf{x}$ with $\mathbf{c}^T\mathbf{x} \geq \mathbf{c}^T\mathbf{x}_0$ we choose one that has the largest possible number of zero components, and we call it $\tilde{\mathbf{x}}$ (it need not be determined uniquely). We define an index set

$$K = \{j \in \{1, 2, \ldots, n\} : \tilde{x}_j > 0\}.$$

If the columns of the matrix A_K are linearly independent, then $\tilde{\mathbf{x}}$ is a basic feasible solution as in the statement, by Lemma 4.2.1.

So let us suppose that the columns of A_K are linearly dependent, which means that there is a nonzero $|K|$-component vector $\mathbf{v}$ such that $A_K\mathbf{v} = \mathbf{0}$. We extend $\mathbf{v}$ by zeros in positions outside K to an n-component vector $\mathbf{w}$ (that is, $\mathbf{w}_K = \mathbf{v}$ and $A\mathbf{w} = A_K\mathbf{v} = \mathbf{0}$).

Let us assume for a moment that $\mathbf{w}$ satisfies the following two conditions (we will show later why we can assume this):

(i) $\mathbf{c}^T\mathbf{w} \geq 0$.

(ii) There exists $j \in K$ with $w_j < 0$.

For a real number $t \geq 0$ let us consider the vector $\mathbf{x}(t) = \tilde{\mathbf{x}} + t\mathbf{w}$. We show that for some suitable $t_1 > 0$ the vector $\mathbf{x}(t_1)$ is a feasible solution with more zero components than $\tilde{\mathbf{x}}$. At the same time, $\mathbf{c}^T\mathbf{x}(t_1) = \mathbf{c}^T\tilde{\mathbf{x}} + t_1\mathbf{c}^T\mathbf{w} \geq \mathbf{c}^T\mathbf{x}_0 + t_1\mathbf{c}^T\mathbf{w} \geq \mathbf{c}^T\mathbf{x}_0$, and so we get a contradiction to the assumption that $\tilde{\mathbf{x}}$ has the largest possible number of zero components.

We have $A\mathbf{x}(t) = \mathbf{b}$ for all t since $A\mathbf{x}(t) = A\tilde{\mathbf{x}} + tA\mathbf{w} = A\tilde{\mathbf{x}} = \mathbf{b}$, because $\tilde{\mathbf{x}}$ is feasible. Moreover, for $t = 0$ the vector $\mathbf{x}(0) = \tilde{\mathbf{x}}$ has all components from K strictly positive and all other components zero. For the jth component of $\mathbf{x}(t)$ we have $x(t)_j = \tilde{x}_j + tw_j$, and if $w_j < 0$ as in condition (ii), we get $x(t)_j < 0$ for all sufficiently large $t > 0$. If we begin with $t = 0$ and let t grow, then those $x(t)_j$ with $w_j < 0$ are decreasing, and at a certain moment t_1 the first of these decreasing components reaches 0. At this moment, obviously, $\mathbf{x}(t_1)$ still has all components nonnegative, and thus it is feasible, but it has at least one extra zero component compared to $\tilde{\mathbf{x}}$. This, as we have already noted, is a contradiction.

Now what do we do if the vector $\mathbf{w}$ fails to satisfy condition (i) or (ii)? If $\mathbf{c}^T\mathbf{w} = 0$, then (i) holds and (ii) can be recovered by changing the sign of $\mathbf{w}$ (since $\mathbf{w} \neq \mathbf{0}$). So we assume $\mathbf{c}^T\mathbf{w} \neq 0$, and again after a possible sign change we can achieve $\mathbf{c}^T\mathbf{w} > 0$ and thus (i). Now if (ii) fails, we must have $\mathbf{w} \geq 0$. But this means that $\mathbf{x}(t) = \tilde{\mathbf{x}} + t\mathbf{w} \geq 0$ for all $t \geq 0$, and hence all such $\mathbf{x}(t)$ are feasible. The value of the objective function for $\mathbf{x}(t)$ is $\mathbf{c}^T\mathbf{x}(t) = \mathbf{c}^T\tilde{\mathbf{x}} + t\mathbf{c}^T\mathbf{w}$, and it tends to infinity as $t \to \infty$. Hence the linear program is unbounded. This concludes the proof. □

4.3 ABC of Convexity and Convex Polyhedra

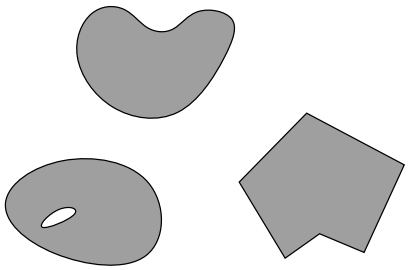
Convexity is one of the fundamental notions of all mathematics, and in the theory of linear programming it is encountered very naturally. Here we recall the definition and present some of the most basic notions and results, which, at the very least, help in gaining a better intuition about linear programming.

On the other hand, linear programming can be presented without these notions, and in concise courses there is usually no time for such material. Accordingly, this section and the next are meant as extending material, and the rest of the book should mostly be accessible without them.

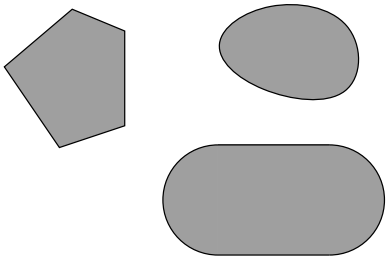
A set $X \subseteq \mathbb{R}^n$ is **convex** if with every two points $\mathbf{x}, \mathbf{y} \in X$ it also contains the segment $\mathbf{xy}$. Expressed differently, for every $\mathbf{x}, \mathbf{y} \in X$ and every $t \in [0,1]$ we have $t\mathbf{x} + (1-t)\mathbf{y} \in X$.

A word of explanation might be in order: $t\mathbf{x} + (1-t)\mathbf{y}$ is the point on the segment $\mathbf{xy}$ at distance t from $\mathbf{y}$ and distance $1-t$ from $\mathbf{x}$, if we take the length of the segment as unit distance.

Here are a few examples of convex and nonconvex sets in the plane:



nonconvex



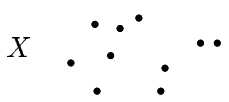
convex

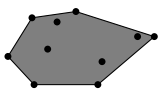
The convex set at the bottom in this picture, a stadium, is worth remembering, since often it is a counterexample to statements about convex sets that may look obvious at first sight but are false.

In calculus one works mainly with convex *functions*. Both notions, convex sets and convex functions, are closely related: For instance, a real function $f\colon \mathbb{R} \to \mathbb{R}$ is convex if and only if its epigraph, i.e., the set $\{(x,y) \in \mathbb{R}^2 : y \geq f(x)\}$, is a convex set in the plane.

Convex hull and convex combinations. It is easily seen that the intersection of an arbitrary collection of convex sets is again a convex set. This allows us to define the convex hull.

Let $X \subset \mathbb{R}^n$ be a set. The **convex hull** of X is the intersection of all convex sets that contain X. Thus it is the smallest convex set containing X, in the sense that any convex set containing X also contains its convex hull.





the convex hull of X

This is not a very constructive definition. The convex hull can also be described using convex combinations, in a way similar to the description of the linear span of a set of vectors using linear combinations. Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ be points in $\mathbb{R}^n$. Every point of the form

$$t_1\mathbf{x}_1 + t_2\mathbf{x}_2 + \cdots + t_m\mathbf{x}_m, \text{ where } t_1, t_2, \ldots, t_m \geq 0 \text{ and } \sum_{i=1}^{m} t_i = 1,$$

is called a **convex combination** of $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$. A convex combination is thus a particular kind of a linear combination, in which the coefficients are nonnegative and sum to 1.

Convex combinations of two points $\mathbf{x}$ and $\mathbf{y}$ are of the form $t\mathbf{x} + (1-t)\mathbf{y}$, $t \in [0,1]$, and as we said after the definition of a convex set, they fill exactly the segment $\mathbf{xy}$. It is easy but instructive to show that all convex combinations of three points $\mathbf{x}, \mathbf{y}, \mathbf{z}$ fill exactly the triangle $\mathbf{xyz}$ (unless the points are collinear, that is).

4.3.1 Lemma. *The convex hull C of a set $X \subseteq \mathbb{R}^n$ equals the set*

$$\tilde{C} = \left\{ \sum_{i=1}^{m} t_i\mathbf{x}_i : m \geq 1, \mathbf{x}_1, \ldots, \mathbf{x}_m \in X, t_1, \ldots, t_m \geq 0, \sum_{i=1}^{m} t_i = 1 \right\}$$

of all convex combinations of finitely many points of X.

Proof. First we prove by induction on m that each convex combination has to lie in the convex hull C. For $m = 1$ it is obvious and for $m = 2$ it follows directly from the convexity of C.

Let $m \geq 3$ and let $\mathbf{x} = t_1\mathbf{x}_1 + \cdots + t_m\mathbf{x}_m$ be a convex combination of points of X. If $t_m = 1$, then we have $\mathbf{x} = \mathbf{x}_m \in C$. For $t_m < 1$ let us put $t'_i = t_i/(1-t_m)$, $i = 1, 2, \ldots, m-1$. Then $\mathbf{x}' = t'_1\mathbf{x}_1 + \cdots + t'_{m-1}\mathbf{x}_{m-1}$ is a convex combination of the points $\mathbf{x}_1, \ldots, \mathbf{x}_{m-1}$ (the t'_i sum to 1), and by the inductive hypothesis $\mathbf{x}' \in C$. So $\mathbf{x} = (1-t_m)\mathbf{x}' + t_m\mathbf{x}_m$ is a convex combination of two points of the (convex) set C and as such it also lies in C.

We have thus proved $\tilde{C} \subseteq C$. For the reverse inclusion it suffices to prove that $\tilde{C}$ is convex, that is, to verify that whenever $\mathbf{x}, \mathbf{y} \in \tilde{C}$ are two convex combinations and $t \in (0,1)$, then $t\mathbf{x} + (1-t)\mathbf{y}$ is again a convex combination. This is straightforward and we take the liberty of omitting further details. □

Convex sets encountered in the theory of linear programming are of a special type and they are called convex polyhedra.

Hyperplanes, half-spaces, polyhedra. We recall that a **hyperplane** in $\mathbb{R}^n$ is an affine subspace of dimension $n-1$. In other words, it is the set of all solutions of a single linear equation of the form

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b,$$

where $a_1, a_2, \ldots, a_n$ are not all 0. Hyperplanes in $\mathbb{R}^2$ are lines and hyperplanes in $\mathbb{R}^3$ are ordinary planes.

A hyperplane divides $\mathbb{R}^n$ into two half-spaces and it constitutes their common boundary. For the hyperplane with equation $a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$, the two half-spaces have the following analytic expression:

$$\Big\{\mathbf{x} \in \mathbb{R}^n : a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b\Big\}$$

and

$$\Big\{\mathbf{x} \in \mathbb{R}^n : a_1x_1 + a_2x_2 + \cdots + a_nx_n \geq b\Big\}.$$

More exactly, these are **closed half-spaces** that contain their boundary.

The intersection of finitely many closed half-spaces in $\mathbb{R}^n$ is a **convex polyhedron**.

A half-space is obviously convex, and hence an intersection of half-spaces is convex as well. Thus convex polyhedra bear the attribute convex by right.

A disk in the plane is a convex set, but it is not a convex polyhedron (because, roughly speaking, a convex polyhedron has to be "edgy"... but try proving this formally).

A half-space is the set of all solutions of a single linear inequality (with at least one nonzero coefficient of some variable x_j). The set of all solutions of a system of finitely many linear inequalities, a.k.a. the set of all feasible solutions of a linear program, is geometrically the intersection of finitely many half-spaces, alias a convex polyhedron. (We should perhaps also mention that a hyperplane is the intersection of two half-spaces, and so the constraints can be both inequalities and equations.)

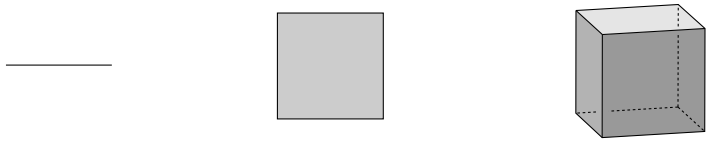
Let us note that a convex polyhedron may also be unbounded since, for example, a single half-space is also a convex polyhedron. A bounded convex polyhedron, i.e. one that can be placed inside some large enough ball, is called a *convex polytope*.

The **dimension** of a convex polyhedron $P \subseteq \mathbb{R}^n$ is the smallest dimension of an affine subspace containing P. Equivalently, it is the largest d for which P contains points $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_d$ such that the d-tuple of vectors $(\mathbf{x}_1 - \mathbf{x}_0, \mathbf{x}_2 - \mathbf{x}_0, \ldots, \mathbf{x}_d - \mathbf{x}_0)$ is linearly independent.

The empty set is also a convex polyhedron, and its dimension is usually defined as -1.

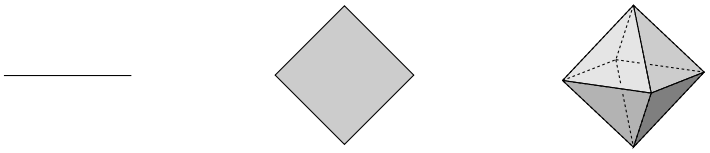
All convex polygons in the plane are two-dimensional convex polyhedra. Several types of three-dimensional convex polyhedra are taught at high schools and decorate mathematical cabinets, such as cubes, boxes, pyramids, or even regular dodecahedra, which can also be met as less usual desktop calendars. Simple examples of convex polyhedra of an arbitrary dimension n are:

- The n-dimensional **cube** $[-1, 1]^n$, which can be written as the intersection of $2n$ half-spaces (which ones?):



$n = 1$ $n = 2$ $n = 3$

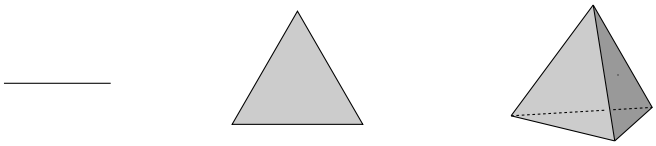
- the n-dimensional **crosspolytope** $\{\mathbf{x} \in \mathbb{R}^n : |x_1|+|x_2|+\cdots+|x_n| \leq 1\}$:



$n = 1$ $n = 2$ $n = 3$

For $n = 3$ we get the regular octahedron. For expressing the n-dimensional crosspolytope as an intersection of half-spaces we need 2^n half-spaces (can you find them?).

- The regular n-dimensional **simplex**



$n = 1$ $n = 2$ $n = 3$

can be defined in a quite simple and nice way as a subset of $\mathbb{R}^{n+1}$:

$$\{\mathbf{x} \in \mathbb{R}^{n+1} : x_1, x_2, \ldots, x_{n+1} \geq 0,\ x_1 + x_2 + \cdots + x_{n+1} = 1\}.$$

We note that this is exactly the set of all feasible solutions of the linear program with the single equation $x_1+x_2+\cdots+x_{n+1} = 1$ and nonnegativity constraints;[4] see the picture in Section 4.1. In general, any n-dimensional convex polytope bounded by $n+1$ hyperplanes is called a simplex.

Many interesting examples of convex polyhedra are obtained as sets of feasible solutions of natural linear programs. For example, the LP relaxation of the problem of maximum-weight matching (Section 3.2) for a complete bipartite graph leads to the *Birkhoff polytope*. Geometric properties of such polyhedra are often related to properties of combinatorial objects and to solutions of combinatorial optimization problems in an interesting way. A very nice book about convex polyhedra is

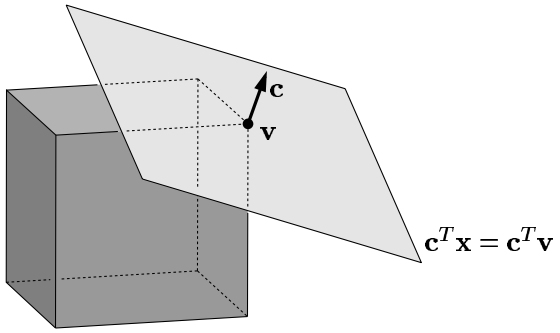
G. M. Ziegler: *Lectures on Polytopes*, Springer-Verlag, Heidelberg, 1994 (corrected 2nd edition 1998).

[4] On the other hand, the set of feasible solutions of a linear program in equational form certainly isn't always a simplex! The simplex method is called that way for a rather complicated reason, related to an alternative geometric view of a linear program in equational form, different from the one discussed in this text. According to this view, the m-tuple of numbers in the jth *column* of the matrix A together with the number c_j is interpreted as a point in $\mathbb{R}^{m+1}$. Then the simplex method can be interpreted as a walk through certain simplices with vertices at these points. It was this view that gave Dantzig faith in the simplex method and convinced him that it makes sense to study it.

4.4 Vertices and Basic Feasible Solutions

A vertex of a convex polyhedron can be thought of as a "tip" or "spike". For instance, a three-dimensional cube has 8 vertices, and a regular octahedron has 6 vertices.

Mathematically, a vertex is defined as a point where some linear function attains a unique maximum. Thus a point $\mathbf{v}$ is called a **vertex** of a convex polyhedron $P \subset \mathbb{R}^n$ if $\mathbf{v} \in P$ and there exists a nonzero vector $\mathbf{c} \in \mathbb{R}^n$ such that $\mathbf{c}^T\mathbf{v} > \mathbf{c}^T\mathbf{y}$ for all $\mathbf{y} \in P \setminus \{\mathbf{v}\}$. Geometrically it means that the hyperplane $\{\mathbf{x} \in \mathbb{R}^n : \mathbf{c}^T\mathbf{x} = \mathbf{c}^T\mathbf{v}\}$ touches the polyhedron P exactly at $\mathbf{v}$.



Three-dimensional polyhedra have not only vertices, but also edges and faces. A general polyhedron $P \subseteq \mathbb{R}^n$ of dimension n can have vertices, edges, 2-dimensional faces, 3-dimensional faces, up until $(n-1)$-dimensional faces. They are defined as follows: A subset $F \subseteq P$ is a ***k*-dimensional face** of a convex polyhedron P if F has dimension k and there exist a nonzero vector $\mathbf{c} \in \mathbb{R}^n$ and a number $z \in \mathbb{R}$ such that $\mathbf{c}^T\mathbf{x} = z$ for all $\mathbf{x} \in F$ and $\mathbf{c}^T\mathbf{x} < z$ for all $\mathbf{x} \in P \setminus F$. In other words, there exists a hyperplane that touches P exactly at F. Since such an F is the intersection of a hyperplane with a convex polyhedron, it is a convex polyhedron itself, and its dimension is thus well defined. Now an **edge** is a 1-dimensional face and a vertex is a 0-dimensional face.

Now we prove that vertices of a convex polyhedron and basic feasible solutions of a linear program are the same concept.

4.4.1 Theorem. *Let P be the set of all feasible solutions of a linear program in equational form (so P is a convex polyhedron). Then the following two conditions for a point $\mathbf{v} \in P$ are equivalent:*

(i) $\mathbf{v}$ *is a vertex of the polyhedron P.*

(ii) $\mathbf{v}$ *is a basic feasible solution of the linear program.*

Proof. The implication (i)⇒(ii) follows immediately from Theorem 4.2.3 (with $\mathbf{c}$ being the vector defining $\mathbf{v}$). It remains to prove (ii)⇒(i).

Let us consider a basic feasible solution $\mathbf{v}$ with a feasible basis B, and let us define a vector $\tilde{\mathbf{c}} \in \mathbb{R}^n$ by $\tilde{c}_j = 0$ for $j \in B$ and $\tilde{c}_j = -1$ otherwise. We have $\tilde{\mathbf{c}}^T\mathbf{v} = 0$, and $\tilde{\mathbf{c}}^T\mathbf{x} \leq 0$ for any $\mathbf{x} \geq \mathbf{0}$, and hence $\mathbf{v}$ maximizes the objective function $\tilde{\mathbf{c}}^T\mathbf{x}$. Moreover, $\tilde{\mathbf{c}}^T\mathbf{x} < 0$ whenever $\mathbf{x}$ has a nonzero component outside

B. But by Proposition 4.2.2, $\mathbf{v}$ is the *only* feasible solution with all nonzero components in B, and therefore, $\mathbf{v}$ is the only point of P maximizing $\tilde{\mathbf{c}}^T\mathbf{x}$. □

A similar theorem is valid for an arbitrary linear program, not only for one in equational form. We will not prove it here, but we at least say what a basic feasible solution is for a general linear program:

A **basic feasible solution** of a linear program with n variables is a feasible solution for which some n linearly independent constraints hold with equality.

A constraint that is an equation always has to be satisfied with equality, while an inequality constraint may be satisfied either with equality or with a strict inequality. The nonnegativity constraints satisfied with equality are also counted. The linear independence of constraints means that the vectors of the coefficients of the variables are linearly independent. For example, for $n = 4$, the constraint $3x_1 + 5x_3 - 7x_4 \leq 10$ has the corresponding vector $(3, 0, 5, -7)$.

As is known from linear algebra, a system of n linearly independent linear equations in n variables has exactly one solution. Hence, if $\mathbf{x}$ is a basic feasible solution and it satisfies some n linearly independent constraints with equality, then it is the only point in $\mathbb{R}^n$ that satisfies these n constraints with equality. Geometrically speaking, the constraints satisfied with equality determine hyperplanes, $\mathbf{x}$ lies on some n of them, and these n hyperplanes meet in a single point.

The definition of a basic feasible solution for the equational form looks quite different, but in fact, it is a special case of the new definition, as we now indicate. For a linear program in equational form we have m linearly independent equations always satisfied with equality, and so it remains to satisfy with equality some $n-m$ of the nonnegativity constraints, and these must be linearly independent with the equations. The coefficient vector of the nonnegativity constraint $x_j \geq 0$ is $\mathbf{e}_j$, with 1 at position j and with zeros elsewhere. If $\mathbf{x}$ is a basic feasible solution according to the new definition, then there exists a set $N \subseteq \{1, 2, \ldots, n\}$ of size $n - m$ such that $x_j = 0$ for all $j \in N$ and the rows of the matrix A together with the vectors $(\mathbf{e}_j : j \in N)$ constitute a linearly independent collection. This happens exactly if the matrix A_B has linearly independent rows, where $B = \{1, 2, \ldots, n\} \setminus N$, and we are back at the definition of a basic feasible solution for the equational form.

For a general linear program none of the optimal solutions have to be basic, as is illustrated by the linear program

$$\text{maximize } x_1 + x_2 \text{ subject to } x_1 + x_2 \leq 1.$$

This contrast with the situation for the equational form (cf. Theorem 4.2.3) and it is one of the advantages of the equational form.

The intuitive notion of a "tip" of a convex set can be viewed mathematically in at least two ways. One of them is captured by the above definition of a vertex of a convex polyhedron. The other one leads to a definition talking about points that cannot be "generated by segments." These are called **extremal points**; thus a point $\mathbf{x}$ is an extremal point of convex set $C \subseteq \mathbb{R}^n$ if $\mathbf{x} \in C$ and there are no two points $\mathbf{y}, \mathbf{z} \in C$ different from $\mathbf{x}$ and such that $\mathbf{x}$ lies on the segment $\mathbf{yz}$.

For a convex polyhedron it is not difficult to show that the extremal points are exactly the vertices. Hence we have yet another equivalent description of a basic feasible solution.

5

The Simplex Method

In this chapter we explain the simplex method for solving linear programs. We will make use of the terms *equational form* and *basic feasible solution* from the previous chapter.

The Gaussian elimination in linear algebra has a fundamental theoretical and didactic significance, as a starting point for further developments. But in practice it has mostly been replaced by more complicated and more efficient algorithms. Similarly, the basic version of the simplex method that we discuss here is not commonly used for solving linear programs in practice. Here we do not put emphasis on the most efficient possible organization of the computations, but rather we concentrate on the main ideas.

5.1 An Introductory Example

We will first show the simplex method in action on a small concrete example, namely, on the following linear program:

$$\begin{array}{lrcrcl} \text{maximize} & x_1 & + & x_2 & & \\ \text{subject to} & -x_1 & + & x_2 & \leq & 1 \\ & x_1 & & & \leq & 3 \\ & & & x_2 & \leq & 2 \\ & x_1, x_2 \geq 0. & & & & \end{array} \tag{5.1}$$

We intentionally do not take a linear program in equational form: The variables are nonnegative, but the inequalities have to be replaced by equations, by introducing auxiliary variables. The equational form is

$$\begin{array}{lrcrcrcrcl} \text{maximize} & x_1 & + & x_2 & & & & & & \\ \text{subject to} & -x_1 & + & x_2 & + & x_3 & & & = & 1 \\ & x_1 & & & & & + & x_4 & = & 3 \\ & & & x_2 & & & & & + \; x_5 \; = & 2 \\ & x_1, x_2, \ldots, x_5 \geq 0, & & & & & & & & \end{array}$$

with the matrix

$$A = \begin{pmatrix} -1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

In the simplex method we first express each linear program in the form of a *simplex tableau*. In our case we begin with the tableau

$$\begin{array}{lcrcrcr} x_3 & = & 1 & + & x_1 & - & x_2 \\ x_4 & = & 3 & - & x_1 & & \\ x_5 & = & 2 & & & - & x_2 \\ \hline z & = & & & x_1 & + & x_2. \end{array}$$

The first three rows consist of the equations of the linear program, in which the auxiliary variables are carried over to the left-hand side and the remaining terms are on the right-hand side. The last row, separated by a line, contains a new variable z, which expresses the objective function.

Each simplex tableau is associated with a certain basic feasible solution. In our case we substitute 0 for the variables x_1 and x_2 from the right-hand side, and without calculation we see that $x_3 = 1, x_4 = 3, x_5 = 2$. This feasible solution is indeed basic with $B = \{3,4,5\}$; we note that A_B is the identity matrix. The variables x_3, x_4, x_5 from the left-hand side are basic and the variables x_1, x_2 from the right-hand side are nonbasic. The value of the objective function $z = 0$ corresponding to this basic feasible solution can be read off from the last row of the tableau.

From the initial simplex tableau we will construct a sequence of tableaus of a similar form, by gradually rewriting them according to certain rules. Each tableau will contain the *same* information about the linear program, only written differently. The procedure terminates with a tableau that represents the information so that the desired optimal solution can be read off directly.

Let us go to the first step. We try to increase the value of the objective function by increasing one of the nonbasic variables x_1 or x_2. In the above tableau we observe that increasing the value of x_1 (i.e. making x_1 positive) increases the value of z. The same is true for x_2, because both variables have positive coefficients in the z-row of the tableau. We can choose either x_1 or x_2; let us decide (arbitrarily) for x_2. We will increase it, while x_1 will stay at 0. By how much can we increase x_2? If we want to maintain feasibility, we have to be careful not to let any of the basic variables x_3, x_4, x_5 go below zero. This means that the equations determining x_3, x_4, x_5 may limit the increment of x_2. Let us consider the first equation

$$x_3 = 1 + x_1 - x_2.$$

Together with the implicit constraint $x_3 \geq 0$ it lets us increase x_2 up to the value $x_2 = 1$ (while keeping $x_1 = 0$). The second equation

$$x_4 = 3 - x_1$$

does not limit the increment of x_2 at all, and the third equation

$$x_5 = 2 - x_2$$

allows for an increase up to the value $x_2 = 2$ before x_5 gets negative. The most stringent restriction thus follows from the first equation.

We increase x_2 as much as we can, obtaining $x_2 = 1$ and $x_3 = 0$. From the remaining equations of the tableau we get the values of the other variables:

$$\begin{aligned} x_4 &= 3 - x_1 = 3, \\ x_5 &= 2 - x_2 = 1. \end{aligned}$$

In this new feasible solution x_3 became zero and x_2 nonzero. Quite naturally we thus transfer x_3 to the right-hand side, where the nonbasic variables live, and x_2 to the left-hand side, where the basic variables reside. We do it by means of the equation $x_3 = 1 + x_1 - x_2$, from which we express

$$x_2 = 1 + x_1 - x_3.$$

We substitute the right-hand side for x_2 into the remaining equations, and we arrive at a new tableau

$$\begin{array}{rcrcrcr} x_2 &=& 1 &+& x_1 &-& x_3 \\ x_4 &=& 3 &-& x_1 & & \\ x_5 &=& 1 &-& x_1 &+& x_3 \\ \hline z &=& 1 &+& 2x_1 &-& x_3. \end{array}$$

Here $B = \{2, 4, 5\}$, which corresponds to the basic feasible solution $\mathbf{x} = (0, 1, 0, 3, 1)$ with the value of the objective function $z = 1$.

This process of rewriting one simplex tableau into another is called a **pivot step**. In each pivot step some nonbasic variable, in our case x_2, *enters* the basis, while some basic variable, in our case x_3, *leaves* the basis.

In the new tableau we can further increase the value of the objective function by increasing x_1, while increasing x_3 would lead to a smaller z-value. The first equation does not restrict the increment of x_1 in any way, from the second one we get $x_1 \leq 3$, and from the third one $x_1 \leq 1$, so the strictest limitation is implied by the third equation. Similarly as in the previous step, we express x_1 from it and we substitute this expression into the remaining equations. Thereby x_1 enters the basis and moves to the left-hand side, and x_5 leaves the basis and migrates to the right-hand side. The tableau we obtain is

$$\begin{array}{rcrcrcr} x_1 &=& 1 &+& x_3 &-& x_5 \\ x_2 &=& 2 & & &-& x_5 \\ x_4 &=& 2 &-& x_3 &+& x_5 \\ \hline z &=& 3 &+& x_3 &-& 2x_5 \end{array}$$

with $B = \{1, 2, 4\}$, basic feasible solution $\mathbf{x} = (1, 2, 0, 2, 0)$, and $z = 3$. After one more pivot step, in which x_3 enters the basis and x_4 leaves it, we arrive at the tableau

$$\begin{array}{rcrcrcr} x_1 &=& 3 &-& x_4 & & \\ x_2 &=& 2 & & &-& x_5 \\ x_3 &=& 2 &-& x_4 &+& x_5 \\ \hline z &=& 5 &-& x_4 &-& x_5 \end{array}$$

with basis $\{1, 2, 3\}$, basic feasible solution $\mathbf{x} = (3, 2, 2, 0, 0)$, and $z = 5$. In this tableau, no nonbasic variable can be increased without making the objective

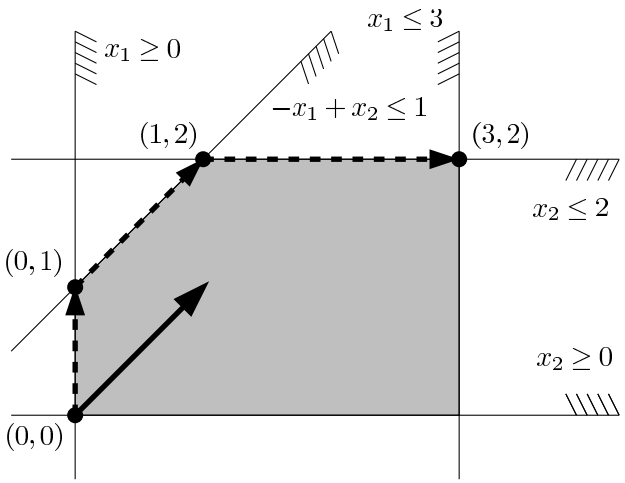
function value smaller, so we are stuck. Luckily, this also means that we have already found an optimal solution! Why?

Let us consider an *arbitrary* feasible solution $\tilde{\mathbf{x}} = (\tilde{x}_1, \ldots, \tilde{x}_5)$ of our linear program, with the value of the objective function equal to some number $\tilde{z}$. Now $\tilde{\mathbf{x}}$ and $\tilde{z}$ satisfy all equations in the final tableau, which was obtained from the original equations of the linear program by equivalent transformations. Hence we necessarily have

$$\tilde{z} \;= 5 - \tilde{x}_4 - \tilde{x}_5.$$

Together with the nonnegativity constraints $\tilde{x}_4, \tilde{x}_5 \geq 0$ this implies $\tilde{z} \leq 5$. The tableau even delivers a proof that $\mathbf{x} = (3, 2, 2, 0, 0)$ is the *only* optimal solution: If $z = 5$, then $x_4 = x_5 = 0$, and this determines the values of the remaining variables uniquely.

A geometric illustration. For each feasible solution (x_1, x_2) of the original linear program (5.1) with inequalities we have exactly one corresponding feasible solution $(x_1, x_2, \ldots, x_5)$ of the modified linear program in equational form, and conversely. The sets of feasible solutions are isomorphic in a suitable sense, and we can thus follow the progress of the simplex method narrated above in a planar picture for the original linear program (5.1):



We can see the simplex method moving along the edges from one feasible solution to another, while the value of the objective function grows until it reaches the optimum. In the example we could also take a shorter route if we decided to increase x_1 instead of x_2 in the first step.

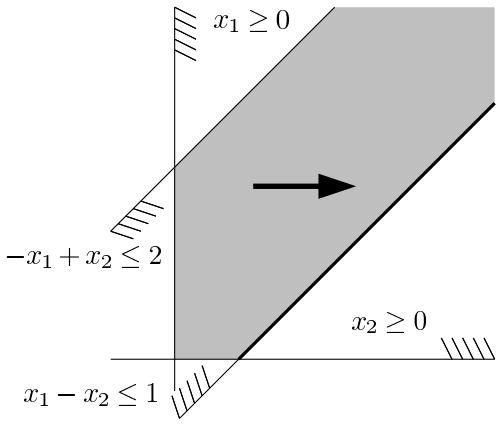
Potential troubles. In our modest example the simplex method has run smoothly without any problems. In general we must deal with several complications. We will demonstrate them on examples in the next sections.

5.2 Exception Handling: Unboundedness

What happens in the simplex method for an unbounded linear program? We will show it on the example

$$\begin{array}{lrcrcl} \text{maximize} & x_1 & & & & \\ \text{subject to} & x_1 & - & x_2 & \leq & 1 \\ & -x_1 & + & x_2 & \leq & 2 \\ & x_1, x_2 \geq 0 & & & & \end{array}$$

illustrated in the picture below:



After the usual transformation to equational form by introducing auxiliary variables x_3, x_4, we can use these variables as a feasible basis and we obtain the initial simplex tableau

$$\begin{array}{rcrcrcr} x_3 & = & 1 & - & x_1 & + & x_2 \\ x_4 & = & 2 & + & x_1 & - & x_2 \\ \hline z & = & & & x_1. & & \end{array}$$

After the first pivot step with entering variable x_1 and leaving variable x_3 the resulting tableau is

$$\begin{array}{rcrcrcr} x_1 & = & 1 & + & x_2 & - & x_3 \\ x_4 & = & 3 & & & - & x_3 \\ \hline z & = & 1 & + & x_2 & - & x_3. \end{array}$$

If we now try to introduce x_2 into the basis, we discover that none of the equations in the tableau restrict its increase in any way. We can thus take x_2 arbitrarily large, and we also get z arbitrarily large—the linear program is unbounded.

Let us analyze this situation in more detail. From the tableau one can see that for arbitrarily large number $t \geq 0$ we obtain a feasible solution by setting $x_2 = t$, $x_3 = 0$, $x_1 = 1 + t$, and $x_4 = 3$, with the value of the objective function $z = 1 + t$. In other words, the semiinfinite ray

$$\{(1,0,0,3) + t(1,1,0,0) : t \geq 0\}$$

is contained in the set of feasible solutions. It "witnesses" the unboundedness of the linear program, since the objective function attains arbitrarily large values on it. The corresponding semiinfinite ray for the original two-dimensional linear program is drawn thick in the picture above.

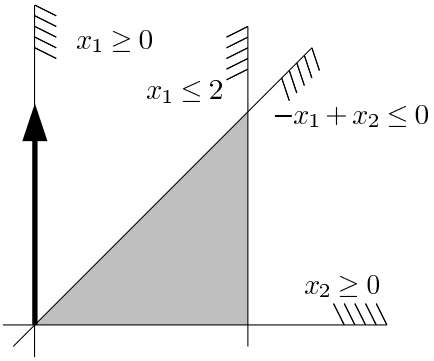
A similar ray is the output of the simplex method for all unbounded linear programs.

5.3 Exception Handling: Degeneracy

While we can make some nonbasic variable arbitrarily large in the unbounded case, the other extreme happens in a situation called a degeneracy: The equations in a tableau do not permit any increment of any of the nonbasic variables, and hence increasing the objective function z in a single pivot step is impossible, too.

Let us consider the linear program

$$\begin{array}{lrcrcl} \text{maximize} & & & x_2 & & \\ \text{subject to} & -x_1 & + & x_2 & \leq & 0 \\ & x_1 & & & \leq & 2 \\ & x_1, x_2 \geq 0. & & & & \end{array} \tag{5.2}$$



In the usual way we convert it to equational form and construct the initial tableau

$$\begin{array}{rcrcrcr} x_3 & = & & & x_1 & - & x_2 \\ x_4 & = & 2 & - & x_1 & & \\ \hline z & = & & & & & x_2. \end{array}$$

The only candidate for entering the basis is x_2, but the first row of the tableau shows that its value cannot be increased without making x_3 negative. Unfortunately, the impossibility of making progress in this case does not imply optimality, so we have to perform a degenerate pivot step, i.e., one with zero progress in the objective function. In our example bringing x_2 into the basis (with x_3 leaving) results in another tableau with the same basic feasible solution $(0, 0, 0, 2)$:

$$\begin{array}{rcrcrcr} x_2 & = & & & x_1 & - & x_3 \\ x_4 & = & 2 & - & x_1 & & \\ \hline z & = & & & x_1 & - & x_3. \end{array}$$

Nevertheless, the situation has improved. The nonbasic variable x_1 can now be increased, and by entering it into the basis (replacing x_4) we already obtain the final tableau

$$\begin{array}{rcrcrcr} x_1 & = & 2 & & & - & x_4 \\ x_2 & = & 2 & - & x_3 & - & x_4 \\ \hline z & = & 2 & - & x_3 & - & x_4 \end{array}$$

with an optimal solution $\mathbf{x} = (2, 2, 0, 0)$.

A situation that forces a degenerate pivot step may occur only for a linear program in which several feasible bases correspond to a single basic feasible solution. Such linear programs are called **degenerate.**

It is easily seen that in order that a single basic feasible solution be obtained from several bases, some of the *basic* variables have to be zero.

In this example, after one degenerate pivot step we were again able to make progress. In general, there might be longer runs of degenerate pivot steps. It may even happen that some tableau is repeated in a sequence of degenerate pivot steps, and so the algorithm might pass through an infinite sequence of tableaus without any progress. This phenomenon is called **cycling**. An example of a linear program for which the simplex method may cycle is quite elaborate (the smallest possible example has 6 variables and 3 equations), and we will not present it here.

If the simplex method doesn't cycle, then it necessarily finishes in a finite number of steps. This is because there are only finitely many possible simplex tableaus for any given linear program, namely at most $\binom{n}{m}$, which we will prove in Section 5.5.

How can cycling be prevented? This is a nontrivial issue and it will be discussed in Section 5.8.

5.4 Exception Handling: Infeasibility

In order that the simplex method can start at all, we need a feasible basis. In examples discussed up until now we got a feasible basis more or less for free. It works this way for all linear programs of the form

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\mathbf{x} \leq \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0}$$

with $\mathbf{b} \geq \mathbf{0}$. Indeed, the auxiliary variables introduced in the transformation to equational form can serve as a feasible basis.

But in general, finding any feasible solution of a linear program is equally difficult as finding an optimal solution (see the remark in Section 1.3). But computing the initial feasible basis can be done by the simplex method itself, if we apply it to a suitable auxiliary problem.

Let us consider the linear program in equational form

$$\begin{array}{lrcrcrcl} \text{maximize} & x_1 & + & 2x_2 & & & & \\ \text{subject to} & x_1 & + & 3x_2 & + & x_3 & = & 4 \\ & & & 2x_2 & + & x_3 & = & 2 \\ & x_1, x_2, x_3 \geq 0. & & & & & & \end{array}$$

Let us try to produce a feasible solution starting with $(x_1, x_2, x_3) = (0, 0, 0)$. This vector is nonnegative but, of course, it is not feasible since it does not satisfy the equations of the linear program. We introduce auxiliary variables x_4 and x_5 as "corrections" of infeasibility: $x_4 = 4 - x_1 - 3x_2 - x_3$ expresses by how much the original variables x_1, x_2, x_3 fail to satisfy the first equation, and $x_5 = 2 - 2x_2 - x_3$ plays a similar role for the second equation. If we manage to find nonnegative values of x_1, x_2, x_3 for which both of these corrections come out as zeros, we would have a feasible solution of the considered linear program.

The task of finding nonnegative x_1, x_2, x_3 with zero corrections can be captured by a linear program:

$$\begin{array}{lrcrcrcrcl} \text{maximize} & & & & & & - & x_4 & - & x_5 & \\ \text{subject to} & x_1 & + & 3x_2 & + & x_3 & + & x_4 & & & = \ 4 \\ & & & 2x_2 & + & x_3 & & & + & x_5 & = \ 2 \\ & x_1, x_2, \ldots, x_5 \geq 0. & & & & & & & & & \end{array}$$

The optimal value of the objective function $-x_4 - x_5$ is 0 exactly if there exist values of x_1, x_2, x_3 with zero corrections, i.e., a feasible solution of the original linear program.

This is the right auxiliary linear program. The variables x_4 and x_5 form a feasible basis, with the basic feasible solution $(0, 0, 0, 4, 2)$. (Here we use that the right-hand sides, 4 and 2, are nonnegative, but since we deal with *equations*, this can always be achieved by sign changes.) Once we express the objective function using the nonbasic variables, that is, in the form $z = -6 + x_1 + 5x_2 + 2x_3$, we can start the simplex method on the auxiliary linear program.

The auxiliary linear program is surely bounded, since the objective function cannot be positive. The simplex method thus computes a basic feasible solution that is optimal. As a training the reader can check that if we let x_1 enter the basis in the first pivot step and x_3 in the second, the final simplex tableau comes out as

$$\begin{array}{rcrcrcrcl} x_1 & = & 2 & - & x_2 & - & x_4 & & \\ x_3 & = & 2 & - & x_2 & & & - & x_5 \\ \hline z & = & & & & - & x_4 & - & x_5. \end{array}$$

The corresponding optimal solution $(2, 0, 2, 0, 0)$ yields the basic feasible solution of the original linear program $(x_1, x_2, x_3) = (2, 0, 2)$. The initial simplex tableau for the original linear program can even be obtained from the final tableau of the auxiliary linear program, by leaving out the columns of the auxiliary variables x_4 and x_5 and by changing the objective function:

$$\begin{array}{rcrcl} x_1 & = & 2 & - & x_2 \\ x_3 & = & 2 & - & x_2 \\ \hline z & = & 2 & + & x_2. \end{array}$$

Starting from this tableau, a single pivot step already reaches the optimum.

5.5 Simplex Tableaus in General

In this section and the next one we formulate in general, and mostly with proofs, what has previously been explained on examples.

Let us consider a general linear program in equational form

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0}.$$

The simplex method applied to it computes a sequence of simplex tableaus. Each of them corresponds to a feasible basis B and it determines a basic feasible solution, as we will soon verify. (Let us recall that a feasible basis is an m-element $B \subseteq \{1, 2, \ldots, n\}$ such that the matrix A_B is nonsingular and the (unique) solution of the system $A_B\mathbf{x}_B = \mathbf{b}$ is nonnegative.)

Formally, we will define a simplex tableau as a certain system of linear equations of a special form, in which the basic variables and the variable z, representing the value of the objective function, stand on the left-hand side and they are expressed in terms of the nonbasic variables.

A **simplex tableau** $\mathcal{T}(B)$ determined by a feasible basis B is a system of $m+1$ linear equations in variables $x_1, x_2, \ldots, x_n$, and z that has the *same set of solutions* as the system $A\mathbf{x} = \mathbf{b}$, $z = \mathbf{c}^T\mathbf{x}$, and in a matrix notation it looks as follows:

$$\begin{array}{rcl} \mathbf{x}_B &=& \mathbf{p} + Q\,\mathbf{x}_N \\ \hline z &=& z_0 + \mathbf{r}^T\mathbf{x}_N, \end{array}$$

where $\mathbf{x}_B$ is the vector of the basic variables, $N = \{1, 2, \ldots, n\} \setminus B$, $\mathbf{x}_N$ is the vector of nonbasic variables, $\mathbf{p} \in \mathbb{R}^m$, $\mathbf{r} \in \mathbb{R}^{n-m}$, Q is an $m \times (n-m)$ matrix, and $z_0 \in \mathbb{R}$.

The basic feasible solution corresponding to this tableau can be read off immediately: It is obtained by substituting $\mathbf{x}_N = \mathbf{0}$; that is, we have $\mathbf{x}_B = \mathbf{p}$. From the feasibility of the basis B we see that $\mathbf{p} \geq \mathbf{0}$. The objective function for this basic feasible solution has value $z_0 + \mathbf{r}^T\mathbf{0} = z_0$.

The values of $\mathbf{p}, Q, \mathbf{r}, z_0$ can easily be expressed using B and $A, \mathbf{b}, \mathbf{c}$:

5.5.1 Lemma. *For each feasible basis B there exists exactly one simplex tableau, and it is given by*

$$Q = -A_B^{-1}A_N,\ \mathbf{p} = A_B^{-1}\mathbf{b},\ z_0 = \mathbf{c}_B^T A_B^{-1}\mathbf{b}, \quad \textit{and} \quad \mathbf{r} = \mathbf{c}_N - (\mathbf{c}_B^T A_B^{-1} A_N)^T.$$

It is neither necessary nor very useful to remember these formulas; they are easily re-derived if needed. The proof is not very exciting and we write it more concisely than other parts of the text and we leave some details to a diligent reader. We will proceed similarly with subsequent proofs of a similar kind.

Proof. First let us see how these formulas can be discovered: We rewrite the system $A\mathbf{x} = \mathbf{b}$ to $A_B\mathbf{x}_B = \mathbf{b} - A_N\mathbf{x}_N$, and we multiply it by the inverse matrix A_B^{-1} from the left (these transformations preserve the solution set), which leads to

$$\mathbf{x}_B = A_B^{-1}\mathbf{b} - A_B^{-1}A_N\mathbf{x}_N.$$

We substitute the right-hand side for $\mathbf{x}_B$ into the equation $z = \mathbf{c}^T\mathbf{x} = \mathbf{c}_B^T\mathbf{x}_B + \mathbf{c}_N^T\mathbf{x}_N$, and we obtain

$$\begin{aligned} z &= \mathbf{c}_B^T(A_B^{-1}\mathbf{b} - A_B^{-1}A_N\mathbf{x}_N) + \mathbf{c}_N^T\mathbf{x}_N = \\ &= \mathbf{c}_B^T A_B^{-1}\mathbf{b} + (\mathbf{c}_N^T - c_B^T A_B^{-1} A_N)\mathbf{x}_N. \end{aligned}$$

Thus the formulas in the lemma do yield a simplex tableau, and it remains to verify the uniqueness.

Let $\mathbf{p}, Q, \mathbf{r}, z_0$ determine a simplex tableau for a feasible basis B, and let $\mathbf{p}', Q', \mathbf{r}', z_0'$ do as well. Since each choice of $\mathbf{x}_N$ determines $\mathbf{x}_B$ uniquely, the equality $\mathbf{p} + Q\mathbf{x}_N = \mathbf{p}' + Q'\mathbf{x}_N$ has to hold for all $\mathbf{x}_N \in \mathbb{R}^{n-m}$. The choice $\mathbf{x}_N = \mathbf{0}$ gives $\mathbf{p} = \mathbf{p}'$, and if we substitute the unit vectors $\mathbf{e}_i$ of the standard basis for $\mathbf{x}_N$ one by one, we also get $Q = Q'$. The equalities $z_0 = z_0'$ and $\mathbf{r} = \mathbf{r}'$ are proved similarly. □

5.6 The Simplex Method in General

Optimality. Exactly as in the concrete example in Section 5.1, we observe:

> If $\mathcal{T}(B)$ is a simplex tableau such that the coefficients of the nonbasic variables are nonpositive in the last row, i.e., if
>
> $$\mathbf{r} \leq \mathbf{0},$$
>
> then the corresponding basic feasible solution is *optimal*.

Indeed, the basic feasible solution corresponding to such a tableau has the objective function equal to z_0, while for any other feasible solution $\tilde{\mathbf{x}}$ we have $\tilde{\mathbf{x}}_N \geq 0$ and $\mathbf{c}^T\tilde{\mathbf{x}} = z_0 + \mathbf{r}^T\tilde{\mathbf{x}}_N \leq z_0$.

A pivot step: who enters and who leaves. In each step of the simplex method we go from an "old" basis B and simplex tableau $\mathcal{T}(B)$ to a "new" basis B' and the corresponding simplex tableau $\mathcal{T}(B')$. A nonbasic variable x_v enters the basis and a basic variable x_u leaves the basis,[1] and hence $B' = (B\backslash\{u\})\cup\{v\}$.

We always select the entering variable x_v first.

> A nonbasic variable may enter the basis if and only if its coefficient in the last row of the simplex tableau is *positive*.

Only incrementing such nonbasic variables increases the value of the objective function.

Usually there are several positive coefficients in the last row, and hence several possible choices of the entering variable. For the time being the reader may think of this choice as arbitrary. We will discuss ways of selecting one of these possibilities in Section 5.7.

Once we decide that the entering variable is some x_v, it remains to pick the leaving variable.

> The leaving variable x_u has to be such that its nonnegativity, together with the corresponding equation in the simplex tableau having x_u on the left-hand side, limit the increment of the entering variable x_v most strictly.

[1] The letters u and v do not denote vectors here (the alphabet is not that long, after all).

Expressed by a formula, this condition might look complicated because of some double indices, but the idea is simple and we have already seen it in examples. Let us write $B = \{j_1, j_2, \ldots, j_m\}$, $j_1 < j_2 < \cdots < j_m$, and $N = \{\ell_1, \ell_2, \ldots, \ell_{n-m}\}$, $\ell_1 < \ell_2 < \cdots < \ell_{n-m}$. Then the ith equation of the simplex tableau has the form

$$x_{j_i} = p_i + \sum_{k=1}^{n-m} q_{ik} x_{\ell_k}.$$

We now want to write the index v of the chosen entering variable as $v = \ell_t$. In more detail, we define $t \in \{1, 2, \ldots, n-m\}$ as the index for which $v = \ell_t$. Similarly, the index u of the leaving variable (which hasn't been selected yet) will be written in the form $u = j_s$.

Since all nonbasic variables x_{ℓ_k}, $k \neq t$, should remain zero, the nonnegativity condition $x_{j_i} \geq 0$ limits the possible values of the entering variable x_{ℓ_t} by the inequality $-q_{it} x_{\ell_t} \leq p_i$. If $q_{it} \geq 0$, then this inequality doesn't restrict the increase of x_{ℓ_t} in any way, while for $q_{it} < 0$ it yields the restriction $x_{\ell_t} \leq -p_i/q_{it}$.

The leaving variable x_{j_s} is thus always such that

$$q_{st} < 0 \quad \text{and} \quad -\frac{p_s}{q_{st}} = \min\left\{-\frac{p_i}{q_{it}} : q_{it} < 0,\ i = 1, 2, \ldots, m\right\}. \tag{5.3}$$

That is, in the simplex tableau we consider only the rows in which the coefficient of x_v is negative. In such rows we divide by this coefficient the component of the vector $\mathbf{p}$, we change sign, and we seek a minimum among these ratios.

If there is no row with a negative coefficient of x_v, i.e., the minimum of the right-hand side of equation (5.3) is over an empty set, then the linear program is unbounded and the computation finishes.

For a proof that the simplex method really goes through a sequence of feasible bases we need the following lemma.

5.6.1 Lemma. *If B is a feasible basis and $\mathcal{T}(B)$ is the corresponding simplex tableau, and if the entering variable x_v and the leaving variable x_u have been selected according to the criteria described above (and otherwise arbitrarily), then $B' = (B \setminus \{u\}) \cup \{v\}$ is again a feasible basis.*

If no x_u satisfies the criterion for a leaving variable, then the linear program is unbounded. For all $t \geq 0$ we obtain a feasible solution by substituting t for x_v and 0 for all other nonbasic variables, and the value of the objective function for these feasible solutions tends to infinity as $t \to \infty$.

The proof is one of those not essential for a basic understanding of the material.

Proof (sketch). We need to verify that the matrix $A_{B'}$ is nonsingular. This holds exactly if $A_B^{-1} A_{B'}$ is nonsingular, since we assume nonsingularity of A_B. The matrix $A_{B'}$ agrees with A_B in $m-1$ columns, and for such columns we get (distinct) unit vectors in the corresponding column of $A_B^{-1} A_{B'}$. The remaining column of the matrix $A_B^{-1} A_{B'}$ occurs in the simplex tableau $\mathcal{T}(B)$ as the column of the entering variable x_v, since $Q = -A_B^{-1} A_N$ by Lemma 5.5.1. There is a nonzero number q_{st} in the row corresponding to the leaving variable x_u, since we have selected x_u that

way, and the other columns of $A_B^{-1}A_{B'}$ have 0 in that row. Hence the matrix is nonsingular as claimed.

Next, we need to check feasibility of the basis B'. Here we use the fact that the new basic feasible solution, that for B', can be written in terms of the old one, and the nonnegativity of its basic variables are exactly those conditions that are used for choosing the leaving variable.

In practically the same way one can show the part of the lemma dealing with unbounded linear programs. We omit further details. □

A geometric view. As we saw in Section 4.4, basic feasible solutions are vertices of the polyhedron of feasible solutions. It is not hard to verify that a pivot step of the simplex method corresponds to a move from one vertex to another along an edge of the polyhedron (where an edge is a 1-dimensional face, i.e., a segment connecting the considered vertices; see Section 4.4). Degenerate pivot steps are an exception, where we stay at the same vertex and only the feasible basis changes.

Organization of the computations. Whenever we find a new feasible basis as above, we could compute the new simplex tableau according to the formulas from Lemma 5.5.1. But this way is never used since it is inefficient.

For *hand calculation* the new simplex tableau is computed from the old one. We have already illustrated one possible approach in the examples. We take the equation of the old tableau with the leaving variable x_u on the left, and in this equation we move the entering variable x_v to the left and x_u to the right. Thus modified equation becomes the equation for x_v in the new tableau. The right-hand side is then substituted for x_v into all the other equations, including the one for z in the last row. This finishes the construction of the new tableau.

Traditionally, the update of the simplex tableau is presented differently, in way similar to the Gauss elimination. Since these days the simplex method is done by hand or by calculators solely for didactic purposes, we stay with the formulation described above.

In computer implementations of the simplex method the simplex tableau is typically not computed in full. Rather, only the basic components of the basic feasible solution, i.e., the vector $\mathbf{p} = A_B^{-1}\mathbf{b}$, and the inverse matrix A_B^{-1} are maintained, and the other entries of the simplex tableau are computed only when needed. (Let us note that for the optimality test and for selecting the entering variable we need only the last row, and for selecting the leaving variable we need only $\mathbf{p}$ and the column of the entering variable.) Since an efficient implementation of the simplex method is not among our main concerns, we will not describe how these things are actually done. This computational approach is called the *revised simplex method*. For m considerably smaller than n it is usually much more efficient than maintaining all of the simplex tableau. In particular, $O(m^2)$ arithmetic operations per pivot step are sufficient for maintaining A_B^{-1}, as opposed to about mn operations required for maintaining the simplex tableau.

Computing an initial feasible basis. If the given linear program has no "obvious" feasible basis, we look for an initial feasible basis by the procedure indicated in Section 5.4. For a linear program in the usual equational form

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0}$$

we first arrange for $\mathbf{b} \geq \mathbf{0}$: We multiply the equations with $b_i < 0$ by -1. Then we introduce m new variables x_{n+1} through x_{n+m}, and we first solve the

auxiliary linear program

$$\begin{array}{ll} \text{maximize} & -(x_{n+1} + x_{n+2} + \cdots + x_{n+m}) \\ \text{subject to} & \bar{A}\,\bar{\mathbf{x}} = \mathbf{b} \\ & \bar{\mathbf{x}} \geq \mathbf{0}, \end{array}$$

where $\bar{\mathbf{x}} = (x_1, \ldots, x_{n+m})$ is the vector of all variables including the new ones, and $\bar{A} = (A \,|\, I_m)$ is obtained from A by appending the $m \times m$ identity matrix to the right. Any feasible solution of the original linear program provides a feasible solution of the auxiliary linear program by setting $x_{n+1} = x_{n+2} = \cdots = x_{n+m} = 0$. Such a feasible solution of the auxiliary linear program has the objective function equal to 0 and thus is optimal. Hence the original linear program is feasible if and only if an optimal solution of the auxiliary linear program satisfies $x_{n+1} = x_{n+2} = \cdots = x_{n+m} = 0$.

The new variables x_{n+1} through x_{n+m} constitute a feasible basis of the auxiliary linear program, and so we can engage the simplex method. It returns some optimal solution. If it doesn't satisfy $x_{n+1} = x_{n+2} = \cdots = x_{n+m} = 0$, we are done—the original linear program is infeasible.

Thus, let us assume that the optimal solution of the auxiliary linear program has $x_{n+1} = x_{n+2} = \cdots = x_{n+m} = 0$. The simplex method always returns a basic feasible solution. If none of the new variables x_{n+1} through x_{n+m} are in the basis for the returned optimal solution, then such a basis is then a feasible basis for the original linear program, too, and it allows us to start the simplex method.

In some degenerate cases it may happen that the basis returned by the simplex method for the auxiliary linear program contains some of the variables $x_{n+1}, \ldots, x_{n+m}$, and such a basis cannot directly be used for the original linear program. But this is a cosmetic problem only: From the returned optimal solution one can get a feasible basis for the original linear program by simple linear algebra. Namely, the optimal solution has at most m nonzero components, and their columns in the matrix A are linearly independent. If these columns are fewer than m, we can add more linearly independent columns and thus get a basis; see Lemma 4.2.1. In the simplex method this part can elegantly be implemented by a pivot step of a special kind, in which the possible leftovers of the variables x_{n+1} through x_{n+m} are forced to leave the basis, but we will not consider this in any detail here.

5.7 Pivot Rules

A *pivot rule* is any rule for selecting the entering variable if there are several possibilities, which is usually the case. Sometimes there may also be more than one possibility for choosing the leaving variable, and some pivot rules specify this choice as well, but this part is typically not so important.

The number of pivot steps needed for solving a linear program depends substantially on the pivot rule. (See the example in Section 5.1.) The problem is, of course, that we do not know in advance which choices will be good in the long run.

Here we list some of the common pivot rules. By an "improving variable" we mean any nonbasic variable with a positive coefficient in the z-row of the simplex tableau, in other words, a candidate for the entering variable.

LARGEST COEFFICIENT. Choose an improving variable with the largest coefficient in the row of the objective function z. This is the original rule, suggested by Dantzig, that maximizes the improvement of z *per unit increase* of the entering variable.

LARGEST INCREASE. Choose an improving variable that leads to the largest *absolute* improvement in z. This rule is computationally more expensive than the LARGEST COEFFICIENT rule, but it locally maximizes the progress.

STEEPEST EDGE. Choose an improving variable whose entering into the basis moves the current basic feasible solution in a direction closest to the direction of the vector $\mathbf{c}$. Written by a formula, the ratio

$$\frac{\mathbf{c}^T(\mathbf{x}_{\text{new}} - \mathbf{x}_{\text{old}})}{\|\mathbf{x}_{\text{new}} - \mathbf{x}_{\text{old}}\|}$$

should be maximized, where $\mathbf{x}_{\text{old}}$ is the basic feasible solution for the current simplex tableau and $\mathbf{x}_{\text{new}}$ is the basic feasible solution for the tableau that would be obtained by entering the considered improving variable into the basis. (We recall that $\|\mathbf{v}\| = (v_1^2 + v_2^2 + \cdots + v_n^2)^{1/2} = \sqrt{\mathbf{v}^T\mathbf{v}}$ denotes the Euclidean length of the vector $\mathbf{v}$, and the expression $\mathbf{u}^T\mathbf{v}/(\|\mathbf{u}\| \cdot \|\mathbf{v}\|)$ is the cosine of the angle of the vectors $\mathbf{u}$ and $\mathbf{v}$.)

The STEEPEST EDGE is a champion among pivot rules in practice. According to extensive computational studies it is usually faster than all other pivot rules described here and many others.

BLAND'S RULE. Choose the improving variable with the smallest index, and if there are several possibilities for the leaving variable, also take the one with the smallest index. BLAND'S RULE is theoretically very significant since it prevents cycling, as we will discuss in Section 5.8.

RANDOM EDGE. Select the entering variable uniformly at random among all improving variables. This is the simplest example of a *randomized pivot rule*, where the choice of the entering variable uses random numbers in some way. Randomized rules are also very important theoretically, since they lead to the current best provable bounds for the number of pivot steps of the simplex method.

5.8 The Struggle Against Cycling

As we have already mentioned, it may happen that for some linear programs the simplex method cycles (and theoretically this is the only possibility of how it may fail). Such a situation is encountered very rarely in practice, if at all, and thus many implementations simply ignore the possibility of cycling.

There are at least three ways that provably avoid cycling. One of them is the already mentioned BLAND'S RULE: We prove below that the simplex method never cycles if Bland's rule is applied consistently. Unfortunately, regarding

efficiency, Bland's rule is one of the slowest pivot rules and it is almost never used in practice.

Another possibility can be found in the literature under the heading *symbolic perturbations*, and here we only sketch it. Cycling can occur only for degenerate linear programs. For equational form degeneracy means that the set F of solutions of the system $A\mathbf{x} = \mathbf{b}$ contains a point with more than $n - m$ zero components, and thus it is not in general position with respect to coordinate axes. The basic idea of symbolic perturbations is to translate the set F by an infinitesimal amount, by changing the vector $\mathbf{b}$ a little. This brings F in "general position," while the optimal solutions changes only by very little. We do not modify the vector $\mathbf{b}$ by concrete numbers, but we replace it by the vector $\mathbf{b}(\varepsilon) = \mathbf{b} + (\varepsilon, \varepsilon^2, \ldots, \varepsilon^m)$, where $\varepsilon > 0$ is a variable that we think of as an extremely small ("infinitesimal") number. The simplex method does not compute with the entries of $\mathbf{b}(\varepsilon)$ as with numbers, but rather as with polynomials in the variable ε; this is another nice trick. When we need to compare two such polynomials, we do it according to the principle "ε is smaller than any specific positive number." So for example, $0.001 > 100\varepsilon$, $1 + 3\varepsilon + 20\varepsilon^3 < 1 + 3\varepsilon + \varepsilon^2$, and so on. The optimal solution computed in this way contains epsilons, but an optimal solution of the original linear program is obtained from it simply by putting $\varepsilon = 0$. If one analyzes this approach in detail and implements it judiciously, it turns out that this variant of the simplex method can never cycle, and at the same time, it doesn't make many more arithmetic operations than the basic variant without epsilons.

Finally, for the pivot rule RANDOM EDGE it can be shown that for every linear program the simplex method is finite with probability 1 (a reader familiar with probability theory can supply an exact formulation of this statement by himself, and others need not worry about it). It can run in a cycle for a while, but sooner or later it leaves it because of the randomness of the pivoting.

5.8.1 Theorem. *The simplex method with Bland's pivot rule (the entering variable is the one with the* smallest index *among the eligible variables, and similarly for the leaving variable) is always finite; i.e., cycling is impossible.*

This is a basic result in the theory of linear programming (the Duality Theorem is an easy consequence, for example). Unfortunately, the proof is not very intuitive and somewhat demanding.

Proof. Assume there is a cycle, and let the set F consist of the indices of all variables that enter (and therefore also leave) the basis at least once during the cycle. We call these the *fickle* variables. First we verify a general claim about cycling of the simplex method, valid for any pivot rule.

Claim I. All bases encountered in the cycle yield the same basic feasible solution, and all the fickle variables are 0 *in it.*

Proof of Claim I. Since the objective function never decreases, it has to stay constant along the cycle.

Let B be a feasible basis encountered along the cycle, let $N = \{1, 2, \ldots, n\} \setminus B$ as usual, and let $B' = (B \setminus \{u\}) \cup \{v\}$ be the next basis. The only one among the *nonbasic* variables that may possibly change value in the pivot step from B to B' is the entering variable x_v; all others remain nonbasic and thus 0. By the rule for selecting the entering variable, the coefficient of x_v in the z-row of the tableau $\mathcal{T}(B)$ (i.e., in the vector $\mathbf{r}$) is strictly positive. Since the objective function

is given by $z = z_0 + \mathbf{r}^T\mathbf{x}_N$, we see that if x_v became strictly positive, the objective function would increase. Hence the basic feasible solutions corresponding to B and B', respectively, agree in all components in N. Since these components determine the remaining ones uniquely (Proposition 4.2.2), the basic feasible solution does not change at all.

Finally, since every fickle variable is nonbasic at least once during the cycle, it has to be 0 all the time. The claim is proved.

The first trick in the proof of Theorem 5.8.1 is to consider the *largest* index v in the set F. Let B be a basis in the cycle just before x_v enters, and B' another basis just before x_v leaves (and x_u enters, say). Let $\mathbf{p}, Q, \mathbf{r}, z_0$ be the parameters of the simplex tableau $\mathcal{T}(B)$, and let $\mathbf{p}', Q', \mathbf{r}', z_0'$ be the parameters of $\mathcal{T}(B')$. (We remark that neither B nor B' have to be determined uniquely.)

Next, we use Bland's rule to infer some properties of the tableaus $\mathcal{T}(B)$ and $\mathcal{T}(B')$. First we focus on the situation at B. As in Section 5.6, we write B and $N = \{1,2,\ldots,n\} \setminus B$ as ordered sets: $B = \{j_1, j_2, \ldots, j_m\}$, $j_1 < j_2 < \cdots < j_m$, and $N = \{\ell_1, \ell_2, \ldots, \ell_{n-m}\}$, $\ell_1 < \ell_2 < \cdots < \ell_{n-m}$. Since we have chosen v as the largest index in F, and Bland's rule requires v to be the smallest index of a candidate for entering the basis, no other fickle variable is a candidate at this point. Thus all fickle variables except for x_v have nonpositive coefficients in the z-row of $\mathcal{T}(B)$. Expressed formally, if t is the index such that $v = \ell_t$, we have

$$r_t > 0 \text{ and } r_k \le 0 \text{ for all } k \text{ such that } \ell_k \in F \setminus \{v\}. \tag{5.4}$$

Second, we consider the tableau $\mathcal{T}(B')$. This time we write $B' = \{j_1', j_2', \ldots, j_m'\}$, $N' = \{1,2,\ldots,n\} \setminus B' = \{\ell_1', \ell_2', \ldots, \ell_{n-m}'\}$, we let s' be the index of the leaving variable x_v in B', i.e., the one with $j_{s'}' = v$, and we let t' be the index of the entering variable x_u in N', i.e., the one with $\ell_{t'}' = u$. By the same logic as above, x_v is the *only* candidate for leaving the basis among all the basic fickle variables in $\mathcal{T}(B')$. Recalling the criterion (5.3) for leaving the basis, we get that $i = s'$ is the only i with $j_i' \in F$ and $q_{it'}' < 0$ that minimizes the ratio $-p_i'/q_{it'}'$. Since $\mathbf{p}'$ specifies the values of the basic variables and all fickle variables remain 0 during the cycle, we have $p_i' = 0$ for all i with $j_i \in F$. Consequently,

$$q_{s't'}' < 0 \text{ and } q_{it'}' \ge 0 \text{ for all } i \text{ such that } j_i' \in F \setminus \{v\}. \tag{5.5}$$

The idea is now to modify, just for the purpose of this proof, the input linear program in such a way that the two strict inequalities in (5.4) and (5.5) get reversed. In this modified problem, and restricted to the fickle variables, the tableau for B has a nonpositive last row, while the tableau for B' has a nonnegative column corresponding to the entering variable. This hopefully rings two bells: A nonpositive last row witnesses optimality, while a nonnegative column corresponding to the entering variable witnesses unboundedness. The rest of the proof is concerned with the formal derivation of a linear program that has an optimal solution and is unbounded at the same time, a clear contradiction.

Here is how it works. We are actually going to define *two* auxiliary linear programs, one of them now and the second later. The first one is

$$\text{maximize } \hat{\mathbf{c}}^T\mathbf{x} \text{ subject to } \hat{A}\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \ge \mathbf{0}, \tag{5.6}$$

where

$$\hat{c}_j = \begin{cases} -c_j & \text{for } j = v \\ c_j & \text{for } j \ne v \end{cases} \quad \text{and} \quad \hat{a}_{ij} = \begin{cases} -a_{ij} & \text{for } j = v \\ a_{ij} & \text{for } j \ne v. \end{cases}$$

That is, we flip the sign of c_v and of the vth column of A.

Claim II. (i) *Both B and B' are feasible bases for the linear program (5.6).*

(ii) *If $\hat{\mathbf{p}}, \hat{Q}, \hat{\mathbf{r}}, \hat{z}_0$ are the parameters of the tableau $\mathcal{T}(B)$ for (5.6), and $\hat{\mathbf{p}}', \hat{Q}', \hat{\mathbf{r}}', \hat{z}_0'$ the ones for $\mathcal{T}(B')$, then:*

$$\hat{\mathbf{p}}' = \mathbf{p}', \tag{5.7}$$

$$\hat{\mathbf{r}}' = \mathbf{r}', \tag{5.8}$$

$$\hat{r}_k \leq 0 \text{ for all } k \text{ with } \ell_k \in F, \text{ and} \tag{5.9}$$

$$\hat{q}'_{it'} \geq 0 \text{ for all } i \text{ with } j'_i \in F. \tag{5.10}$$

Proof of Claim II. This is a routine exercise. Since $v \notin B$, we have $\hat{A}_B = A_B$ nonsingular and $\hat{\mathbf{p}} = \hat{A}_B^{-1}\mathbf{b} = \mathbf{p} \geq \mathbf{0}$ (Lemma 5.5.1), hence B is a feasible basis.

The matrix $\hat{A}_{B'}$ differs from $A_{B'}$ exactly in the signs of the s'th column (corresponding to x_v). It is a small exercise to deduce from this that $\hat{A}_{B'}^{-1}$ exists and differs from $A_{B'}^{-1}$ exactly in the signs of the s'th *row*. Thus, $\hat{\mathbf{p}}'$ agrees with $\mathbf{p}'$ in all entries except possibly the s'th one, where $\hat{p}'_{s'} = -p'_{s'}$. But x_v is 0 throughout the cycle, so $\hat{p}'_{s'} = 0 = p'_{s'}$, and $\hat{\mathbf{p}}' = \mathbf{p}' \geq 0$. We have verified part (i) and (5.7).

By Lemma 5.5.1 again we have $\mathbf{r} = \mathbf{c}_N - (\mathbf{c}_B^T A_B^{-1} A_N)^T$ and $\hat{\mathbf{r}} = \hat{\mathbf{c}}_N - (\hat{\mathbf{c}}_B^T \hat{A}_B^{-1} \hat{A}_N)^T$. Using $\hat{\mathbf{c}}_B = \mathbf{c}_B$, $\hat{A}_B = A_B$ and a little matrix algebra, we see that $\hat{r}_t = -r_t$ and $\hat{r}_i = r_i$ for all $i \neq t$. Together with (5.4) we obtain (5.9).

The equality (5.8), $\hat{\mathbf{r}}' = \mathbf{r}'$, follows easily from the formula from Lemma 5.5.1 employed a few lines above and from our knowledge about $\hat{A}_{B'}^{-1}$.

Finally, using the formula $Q' = A_{B'}^{-1} A_{N'}$ from Lemma 5.5.1, we deduce that $\hat{Q}'$ agrees with Q' everywhere except for the s'th row, in which all signs are reversed. This and (5.5) yield (5.10). Claim II is proved.

Now we are ready for the punchline (which, like any good punchline, needed a long preparation). We recall that the equation $z = \hat{z}_0 + \hat{\mathbf{r}}^T \mathbf{x}_N$ expresses the value of the objective function of the auxiliary linear program in terms of the nonbasic variables corresponding to the feasible basis B. With (5.9), we get $z \leq \hat{z}_0$ whenever $\hat{A}\mathbf{x} = \mathbf{b}$, $\mathbf{x}_F \geq \mathbf{0}$ and $\mathbf{x}_{N \setminus F} = \mathbf{0}$. In other words, the basic feasible solution associated with B is an *optimal* solution to the following *second auxiliary linear program:*

$$\text{maximize } \hat{\mathbf{c}}^T \mathbf{x} \text{ subject to } \hat{A}\mathbf{x} = \mathbf{b}, \mathbf{x}_{N \setminus F} = \mathbf{0}, \text{ and } \mathbf{x}_F \geq \mathbf{0} \tag{5.11}$$

(note that we do *not* require the nonnegativity of the variables with indices in $B \setminus F$!).

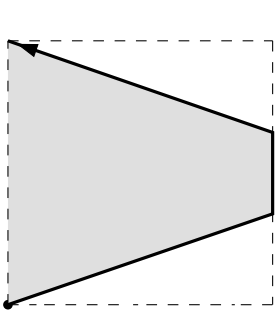
On the other hand, since x_u was a candidate for entering the basis B' in the original linear program, and since $\hat{\mathbf{r}}' = \mathbf{r}'$, x_u is still a candidate in the first auxiliary linear program. But (5.10) shows that when we increase the value of x_u, no basic fickle variable limits the increment of x_u. More precisely, if we set x_u to an arbitrarily large real number, let $\mathbf{x}_{N' \setminus \{u\}} = \mathbf{0}$, and compute $\mathbf{x}_{B'}$ from $\hat{A}\mathbf{x} = \mathbf{b}$, all fickle variables are nonnegative; moreover, $\mathbf{x}_{N \setminus F} = \mathbf{0}$, because $N \setminus F = N' \setminus F \subseteq N' \setminus \{u\}$, so we obtain a feasible solution to the second auxiliary linear program (5.11), while $\hat{\mathbf{c}}^T \mathbf{x}$ becomes arbitrarily large. Therefore, the linear program (5.11), to which we have previously found an optimal solution, is unbounded at the same time. This is the desired contradiction to the existence of a cycle under Bland's rule. □

5.9 Efficiency of the simplex method

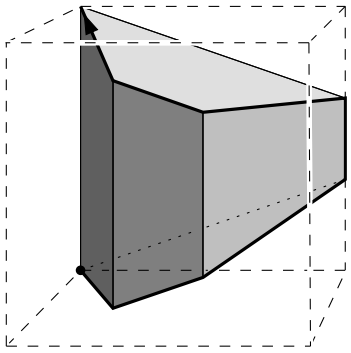
In practice the simplex method performs very satisfactorily even for large linear programs. Computational experiments indicate that for linear programs in equational form with m equations it typically reaches an optimal solution in something between $2m$ and $3m$ pivot steps.

It was thus a great surprise when Klee and Minty constructed a linear program in equational form with $2n$ variables and n equations for which the simplex method with Dantzig's original pivot rule (LARGEST COEFFICIENT) and with some initial feasible basis needs exponentially many pivot steps, namely $2^n - 1$!

The set of feasible solutions is an ingeniously deformed n-dimensional cube, called the *Klee–Minty cube*, constructed in such a way that the simplex method passes through all of its vertices. It is not hard to see that there is a deformed n-dimensional cube with an x_n-increasing path, say, through all vertices. Instead of a formal description we illustrate such a construction by pictures for dimensions 2 and 3:

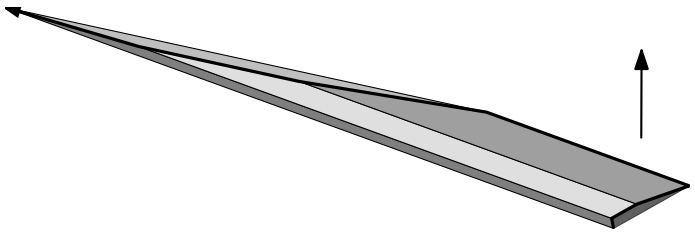


$n = 2$



$n = 3$

The deformed cube is inscribed in an ordinary cube in order to better convey the shape. With some pivot rules, the simplex method may traverse the path marked thick. The particular deformed cube shown in the picture won't fool Dantzig's rule, for which the original example of Klee and Minty was constructed, though. A deformed cube that does fool Dantzig's rule looks more bizarre:



The direction of the objective function is drawn vertical. The corresponding linear program with $n = 3$ variables is simple:

$$
\begin{array}{lrcrcrcl}
\text{maximize} & 9x_1 & + & 3x_2 & + & x_3 & & \\
\text{subject to} & x_1 & & & & & \leq & 1 \\
 & 6x_1 & + & x_2 & & & \leq & 9 \\
 & 18x_1 & + & 6x_2 & + & x_3 & \leq & 81 \\
 & x_i \geq 0, & & i = 1, \ldots, 3. & & & &
\end{array}
$$

It is instructive to see how, after the standard conversion to equational form, this linear program forces Dantzig's rule to go through all feasible bases.

Later on, very slow examples of a similar type have been discovered for many other pivot rules, among them all the rules mentioned above. Many people have tried to design a pivot rule and prove that the number of pivot steps is always bounded by some polynomial function of m and n, but nobody has succeeded so far. The best known bound is known for the following simple randomized pivot rule: Choose a random ordering of the variables at the beginning of the computation (in other words, randomly permute the indices of the variables in the input linear program), and then follow Bland's rule. For every linear program with at most n variables and at most n constraints, the expected number of pivot steps is bounded by $e^{C\sqrt{n \ln n}}$, where C is a (not too large) constant. (Here the expectation means the arithmetic average over all possible orderings of the variables.) This bound is considerably better than, say, 2^n, but much worse than a polynomial bound.

This algorithm was found independently and almost at the same time by Kalai and by Matoušek, Sharir, and Welzl. For a recent treatment in a somewhat broader context see

> B. Gärtner and E. Welzl: Explicit and implicit enforcing—randomized optimization, in *Lectures of the Graduate Program Computational Discrete Mathematics*, *Lecture Notes in Computer Science 2122*, Springer, Berlin etc., pages 26–49.

A very good bound is not known even for the most clever possible pivot rule, let us call it the clairvoyant's rule, that would always select the shortest possible sequence of pivot steps leading to an optimal solution. The *Hirsch conjecture*, one of the famous open problems in mathematics, claims that the clairvoyant's rule always reaches optimum in $O(n)$ pivot steps. But the best result proved so far gives only the bound of $n^{1+\ln n}$, due to

> G. Kalai and D. Kleitman: Quasi-polynomial bounds for the diameter of graphs of polyhedra, *Bull. Amer Math. Soc.* 26(1992), 315–316.

This is better than the $e^{C\sqrt{n \ln n}}$, but still worse than any polynomial function of n, and it doesn't provide a real pivot rule since nobody knows how to simulate clairvoyant's decisions by an algorithm.

In spite of the Klee–Minty cube and similar artificial examples, the simplex method is being used successfully. Remarkable theoretical results indicate that these willful examples are rare indeed. For instance, it is known that if a linear program in equational form is generated in a suitable (precisely defined) way at random, then the number of pivot steps is of order at most m^2 with high probability. Another, more recent result, by

> D. A. Spielman and S.-H. Teng: Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time, *Journal of the ACM* 51,3(2004) 385–463,

claims that if we take an arbitrary linear program and then we change its coefficients by small random amounts, then the simplex method with a certain pivot rule reaches the optimum of the resulting linear program by polynomially many steps with high probability (a concrete bound on the polynomial depends on a precise specification of the "small random amounts" of change). Exact formulation of these

two results requires a number of rather technical notions which we do not want to introduce here, and so we omit it.

5.10 Summary

Let us review the simplex method once again.

Algorithm SIMPLEX METHOD

1. Convert the input linear program to equational form

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0}$$

with n variables and m equations, where A has rank m.

2. If no feasible basis is available, arrange for $\mathbf{b} \geq \mathbf{0}$, and solve the following auxiliary linear program by the simplex method:

$$\begin{array}{ll} \text{maximize} & -(x_{n+1} + x_{n+2} + \cdots + x_{n+m}) \\ \text{subject to} & \bar{A}\bar{\mathbf{x}} = \mathbf{b} \\ & \bar{\mathbf{x}} \geq \mathbf{0}, \end{array}$$

where $x_{n+1}, \ldots, x_{n+m}$ are new variables, $\bar{\mathbf{x}} = (x_1, \ldots, x_{n+m})$, and $\bar{A} = (A \mid I_m)$. If the optimal value of the objective function comes out negative, the original linear program is infeasible; **stop.** Otherwise, the first n components of the optimal solution form a basic feasible solution of the original linear program (see Section 5.6).

3. For a feasible basis $B \subseteq \{1, 2, \ldots, n\}$ compute the simplex tableau $\mathcal{T}(B)$, of the form

$$\begin{array}{rcccc} \mathbf{x}_B & = & \mathbf{p} & + & Q\mathbf{x}_N \\ \hline z & = & z_0 & + & \mathbf{r}^T\mathbf{x}_N. \end{array}$$

4. If $\mathbf{r} \leq \mathbf{0}$ in the current simplex tableau, return an optimal solution ($\mathbf{p}$ specifies the basic components, while the nonbasic components are 0); **stop.**

5. Otherwise, select an *entering variable* x_v, whose coefficient in the vector $\mathbf{r}$ is positive. If there are several possibilities, use some pivot rule; see Section 5.7.

6. If the column of the entering variable x_v in the simplex tableau is non-negative, the linear program is unbounded; **stop.**

7. Otherwise, select a *leaving variable* x_u. Consider all rows of the simplex tableau where the coefficient of x_v is negative, and in each such row divide the component of the vector $\mathbf{p}$ by that coefficient and change sign. The row of the leaving variable is one in which this ratio is minimum (if there are several possibilities, decide by a pivot rule, or arbitrarily if the pivot rule doesn't specify how to break ties in this case).

8. Replace the current feasible basis B by the new feasible basis $(B \setminus \{u\}) \cup \{v\}$. Update the simplex tableau so that it corresponds to this new basis (see Section 5.6). Go to Step 4.

6

Duality of Linear Programming

6.1 The Duality Theorem

Here we formulate arguably the most important theoretical result about linear programs.

Let us consider the linear program

$$\begin{array}{lrcrcr} \text{maximize} & 2x_1 & + & 3x_2 & & \\ \text{subject to} & 4x_1 & + & 8x_2 & \leq & 12 \\ & 2x_1 & + & x_2 & \leq & 3 \\ & 3x_1 & + & 2x_2 & \leq & 4 \\ & x_1, x_2 \geq 0. & & & & \end{array} \tag{6.1}$$

Without computing the optimum, we can immediately infer from the first inequality and from the nonnegativity constraints that the maximum of the objective function is not larger than 12, because for nonnegative x_1 and x_2 we have

$$2x_1 + 3x_2 \leq 4x_1 + 8x_2 \leq 12.$$

We obtain a better upper bound if we first divide the first inequality by two:

$$2x_1 + 3x_2 \leq 2x_1 + 4x_2 \leq 6.$$

An even better bound results if we add the first two equations together and divide by three, which leads to the inequality

$$2x_1 + 3x_2 = \frac{1}{3}(4x_1 + 8x_2 + 2x_1 + x_2) \leq \frac{1}{3}(12 + 3) = 5,$$

and hence the objective function cannot be larger than 5.

How good an upper bound can we get in this way? And, what does it mean "in this way"? Let us begin with the latter question: From the constraints we are trying to derive an inequality of the form

$$d_1 x_1 + d_2 x_2 \leq h,$$

where $d_1 \geq 2$, $d_2 \geq 3$, and h is as small as possible. Then we can claim that for all $x_1, x_2 \geq 0$ we have

$$2x_1 + 3x_2 \leq d_1 x_1 + d_2 x_2 \leq h,$$

and therefore, h is an upper bound on the maximum of the objective function. How can we derive such inequalities? We combine the three inequalities in the linear program with some nonnegative coefficients y_1, y_2, y_3 (nonnegativity is needed so that the direction of inequality is not reversed). We obtain

$$(4y_1 + 2y_2 + 3y_3)x_1 + (8y_1 + y_2 + 2y_3)x_2 \leq 12y_1 + 3y_2 + 4y_3,$$

and thus $d_1 = 4y_1 + 2y_2 + 3y_3$, $d_2 = 8y_1 + y_2 + 2y_3$, and $h = 12y_1 + 3y_2 + 4y_3$.

How do we choose the best coefficients y_1, y_2, y_3? We must assure $d_1 \geq 2$ and $d_2 \geq 3$, and we want h the smallest possible under these constraints. This is again a linear program:

$$\begin{array}{lrcrcrcl} \text{minimize} & 12y_1 & + & 3y_2 & + & 4y_3 & & \\ \text{subject to} & 4y_1 & + & 2y_2 & + & 3y_3 & \geq & 2 \\ & 8y_1 & + & y_2 & + & 2y_3 & \geq & 3 \\ & y_1, y_2, y_3 \geq 0. & & & & & & \end{array}$$

It is called the linear program *dual* to the linear program (6.1) we started with. The dual linear program "guards" the original linear program from above, in the sense that every feasible solution (y_1, y_2, y_3) of the dual linear program provides an upper bound on the maximum of the objective function in (6.1).

How well it guards? Perfectly! The optimal solution of the dual linear program is $\mathbf{y} = (\frac{5}{16}, 0, \frac{1}{4})$ with objective function equal to 4.75, and this is also the optimal value of the linear program (6.1), which is attained for $\mathbf{x} = (\frac{1}{2}, \frac{5}{4})$.

The Duality Theorem asserts that the dual linear program *always* guards perfectly. Let us repeat the above considerations in a more general setting, for a linear program of the form

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\,\mathbf{x} \leq \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0}, \qquad \text{(P)}$$

where A is a matrix with m rows and n columns. We are trying to combine the m inequalities of the system $A\mathbf{x} \leq \mathbf{b}$ with some nonnegative coefficients $y_1, y_2, \ldots, y_m$ so that

- the resulting inequality has the jth coefficients at least c_j, $j = 1, 2, \ldots, n$, and
- the right-hand side is as small as possible.

This leads to the **dual linear program**

$$\text{minimize } \mathbf{b}^T\mathbf{y} \text{ subject to } A^T\mathbf{y} \geq \mathbf{c} \text{ and } \mathbf{y} \geq \mathbf{0}; \qquad \text{(D)}$$

who doesn't believe may write it in components. In this connection the linear program (P) is referred to as the **primal linear program.**

From the way we have produced the dual linear program (D), we obtain:

6.1.1 Proposition. *For each feasible solution* $\mathbf{y}$ *of the dual linear program* (D) *the value* $\mathbf{b}^T\mathbf{y}$ *provides an upper bound on the maximum of the objective*

function of the linear program (P). *In other words, for each feasible solution* $\mathbf{x}$ *of* (P) *and each feasible solution* $\mathbf{y}$ *of* (D) *we have*

$$\mathbf{c}^T\mathbf{x} \leq \mathbf{b}^T\mathbf{y}.$$

In particular, if (P) *is unbounded,* (D) *has to be infeasible, and if* (D) *is unbounded (from below!), then* (P) *is infeasible.*

This proposition is usually called the *weak duality theorem*, weak because it expresses only the guarding of the primal linear program (P) by the dual linear program (D), and it doesn't say that the guarding is perfect. This is expressed only by the Duality Theorem (sometimes also called the *Strong Duality Theorem*).

Duality Theorem of linear programming

For the linear programs

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\mathbf{x} \leq \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0} \qquad \text{(P)}$$

and

$$\text{minimize } \mathbf{b}^T\mathbf{x} \text{ subject to } A^T\mathbf{y} \geq \mathbf{c} \text{ and } \mathbf{y} \geq \mathbf{0} \qquad \text{(D)}$$

exactly one of the following possibilities occurs:

1. Neither (P) nor (D) has a feasible solution.
2. (P) is unbounded and (D) has no feasible solution.
3. (P) has no feasible solution and (D) is unbounded.
4. Both (P) and (D) have a feasible solution. Then both have an optimal solution, and if $\mathbf{x}^*$ is an optimal solution of (P) and $\mathbf{y}^*$ is an optimal solution of (D), then
$$\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*.$$
That is, the *maximum of (P) equals the minimum of (D).*

The Duality Theorem might look complicated at first encounter. For understanding it better it may be useful to consider a simpler version, called the Farkas lemma and discussed in Section 6.4. This simpler statement has several intuitive interpretations and it contains the essence of the Duality Theorem.

Proving the Duality Theorem, which we will consider in Sections 6.3 and 6.4, does take some work, unlike a proof of the Weak Duality Theorem, which is quite easy with the right approach.

The following important corollary of the Duality Theorem is known as the theorem of *complementary slackness*. We leave the easy proof to the reader, along with the question, what the term complementary slackness refers to.

6.1.2 Theorem (Complementary slackness). *Let* $\mathbf{x}^* = (x_1^*, \ldots, x_n^*)$ *be a feasible solution to (P), and let* $\mathbf{y}^* = (y_1^*, \ldots, y_m^*)$ *be a feasible solution to (D). Then the following two statements are equivalent:*

(i) $\mathbf{x}^*$ *is optimal for (P), and* $\mathbf{y}^*$ *is optimal for (D).*

(ii) For all $i \in \{1,\ldots,m\}$, $\mathbf{x}^*$ *satisfies the* i*th constraint of (P) with equality, or* $y_i^* = 0$*; similarly, for all* $j \in \{1,\ldots,n\}$, $\mathbf{y}^*$ *satisfies the* j*th constraint of (D) with equality, or* $x_j^* = 0$.

Once again: Feasibility versus optimality. In Chapter 1 we remarked that finding a feasible solution of a linear program is in general computationally as difficult as finding an optimal solution. There we have briefly substantiated this claim using binary search. The Duality Theorem provides a considerably more elegant argument: The linear program (P) has an optimal solution if and only if the following linear program, obtained by combining the constraints of (P), the constraints of (D), and an inequality between the objective functions, has a feasible solution:

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} \leq \mathbf{b} \\ & A^T\mathbf{y} \geq \mathbf{c} \\ & \mathbf{c}^T\mathbf{x} \geq \mathbf{b}^T\mathbf{y} \\ & \mathbf{x} \geq \mathbf{0}, \mathbf{y} \geq \mathbf{0} \end{array}$$

(the objective function is immaterial here, and the variables are $x_1,\ldots,x_n$ and $y_1,\ldots,y_m$). Moreover, for each feasible solution $(\tilde{\mathbf{x}},\tilde{\mathbf{y}})$ of the last linear program, $\tilde{\mathbf{x}}$ is an optimal solution of the linear program (P). All of this is a simple consequence of the Duality Theorem.

6.2 Dualization for Everyone

The Duality Theorem is valid for each linear program, not only for one of the form (P); we only have to construct the dual linear program properly. To this end, we can convert the given linear program to the form (P) using the tricks from Sections 1.1 and 4.1, and then the dual linear program has the form (D). The result can often be simplified; for example, the difference of two nonnegative variables can be replaced by a single unbounded variable (one that may attain all real values).

Simpler than doing this again and again is to adhere to the recipe below (whose validity can be proved by the just mentioned procedure). Let us assume that the primary linear program has $x_1, x_2, \ldots, x_n$, among which some may be nonnegative, some nonpositive, and some unbounded. Let the constraints be $C_1, C_2, \ldots, C_m$, where C_i has the form

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \left\{ \begin{array}{c} \leq \\ \geq \\ = \end{array} \right\} b_i.$$

(The nonnegativity or nonpositivity constraints for the variables are not counted among the C_i.) The objective function $\mathbf{c}^T\mathbf{x}$ should be *maximized*.

Then the dual linear program has variables $y_1, y_2, \ldots, y_m$, where y_i corresponds to the constraint C_i and satisfies

$$\left\{\begin{array}{l} y_i \geq 0 \\ y_i \leq 0 \\ y_i \in \mathbb{R} \end{array}\right\} \text{ if we have } \left\{\begin{array}{l} \leq \\ \geq \\ = \end{array}\right\} \text{ in } C_i.$$

The constraints of the dual linear program are $Q_1, Q_2, \ldots, Q_n$, where Q_j corresponds to the variable x_j and reads

$$a_{1j}y_1 + a_{2j}y_2 + \cdots + a_{mj}y_m \left\{\begin{array}{l} \geq \\ \leq \\ = \end{array}\right\} c_j \text{ if } x_j \text{ satisfies } \left\{\begin{array}{l} x_j \geq 0 \\ x_j \leq 0 \\ x_j \in \mathbb{R} \end{array}\right\}.$$

The objective function is $\mathbf{b}^T\mathbf{y}$, and it is to be *minimized.*

Note that in the first part of the recipe (from primal constraints to dual variables) the direction of inequalities is reversed, while in the second part (from primal variables to dual constraints) the direction is preserved.

Dualization Recipe

	Primal linear program	Dual linear program
Variables	$x_1, x_2, \ldots, x_n$	$y_1, y_2, \ldots, y_m$
Matrix	A	A^T
Right-hand side	$\mathbf{b}$	$\mathbf{c}$
Objective function	$\max \mathbf{c}^T\mathbf{x}$	$\min \mathbf{b}^T\mathbf{y}$
Inequalities	ith constraint has $\geq$	$y_i \leq 0$
	$\leq$	$y_i \geq 0$
	$=$	$y_i \in \mathbb{R}$
	$x_j \geq 0$	jth constraint has $\geq$
	$x_j \leq 0$	$\leq$
	$x_j \in \mathbb{R}$	$=$

If we want to dualize a *minimization* linear program, we can first transform it to a maximization linear program by changing the sign of the objective function, and then follow the recipe.

In this way one can also find out that the rules work symmetrically "there" and "back." By this we mean that if we start with some linear program, construct the dual linear program, and then again the dual linear program, we get back to the original (primal) linear program; two consecutive dualizations cancel out. In particular, the linear programs (P) and (D) in the Duality Theorem are *dual to each other.*

A physical interpretation of duality. Let us consider a linear program

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\mathbf{x} \le \mathbf{b}.$$

According to the dualization recipe the dual linear program is

$$\text{minimize } \mathbf{b}^T\mathbf{y} \text{ subject to } A^T\mathbf{y} = \mathbf{c} \text{ and } \mathbf{y} \ge \mathbf{0}.$$

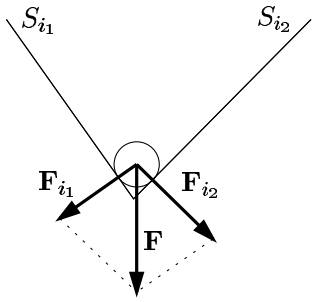
Let us assume that the primal linear program is feasible and bounded, and let $n = 3$. We regard $\mathbf{x}$ as a point in the three-dimensional space, and we interpret $\mathbf{c}$ as the gravitation vector; it thus points downward.

Each of the inequalities of the system $A\mathbf{x} \le \mathbf{b}$ determines a half-space. The intersection of these half-spaces is a nonempty convex polyhedron bounded from below. Each of its two-dimensional faces is given by one of the equations $\mathbf{a}_i^T\mathbf{x} = b_i$, where the vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m$ are the rows of the matrix A, but interpreted as column vectors. Let us denote the face given by $\mathbf{a}_i^T\mathbf{x} = b_i$ by S_i (not every inequality of the system $A\mathbf{x} \le \mathbf{b}$ has to correspond to a face, and so S_i is not necessarily defined for every i).

Let us imagine that the boundary of the polyhedron is made of cardboard and that we drop a tiny steel ball somewhere inside the polyhedron. The ball falls and rolls down to the lowest vertex (or possibly it stays on a horizontal edge or face). Let us denote the resulting position of the ball by $\mathbf{x}^*$; thus, $\mathbf{x}^*$ is an optimal solution of the linear program. In this stable position the ball touches several two-dimensional faces, typically 3. Let D be the set of i such that the ball touches the face S_i. For $i \in D$ we thus have

$$\mathbf{a}_i^T\mathbf{x}^* = b_i. \tag{6.6}$$

The gravity exerts a force $\mathbf{F}$ on the ball that is proportional to the vector $\mathbf{c}$. This force is decomposed into forces of pressure on the faces touched by the ball. The force $\mathbf{F}_i$ by which the ball acts on face S_i is perpendicular to S_i and it is directed outward from the polyhedron (if we neglect friction); see a schematic two-dimensional picture:



The forces acting on the ball are in equilibrium, and thus $\mathbf{F} = \sum_{i\in D} \mathbf{F}_i$. The outward normal of the face S_i is $\mathbf{a}_i$, hence $\mathbf{F}_i$ is proportional to $\mathbf{a}_i$, and for some nonnegative numbers y_i^* we have

$$\sum_{i\in D} y_i^*\mathbf{a}_i = \mathbf{c}.$$

If we set $y_i^* = 0$ for $i \notin D$, we can write $\sum_{i=1}^m y_i^*\mathbf{a}_i = \mathbf{c}$, or $A^T\mathbf{y}^* = \mathbf{c}$ in matrix form. Therefore, $\mathbf{y}^*$ is a is a feasible solution of the dual linear program.

Let us consider the product $(\mathbf{y}^*)^T(A\mathbf{x}^* - \mathbf{b})$. For $i \notin D$ the ith component of $\mathbf{y}^*$ equals 0, while for $i \in D$ the ith component of $A\mathbf{x}^* - \mathbf{b}$ is 0 according to (6.6). So the product is 0, and hence $(\mathbf{y}^*)^T\mathbf{b} = (\mathbf{y}^*)^TA\mathbf{x}^* = \mathbf{c}^T\mathbf{x}^*$.

We see that $\mathbf{x}^*$ is an optimal solution of the primal linear program, $\mathbf{y}^*$ is a feasible solution of the dual linear program, and $\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*$. By the Weak

Duality Theorem $\mathbf{y}^*$ is an optimal solution of the dual linear program, and we have a situation exactly as in the Duality Theorem. We have just "physically verified" a special three-dimensional case of the Duality Theorem.

We remark that the dual linear program also has an economic interpretation. The dual variables are called *shadow prices* in this context. An interested reader finds this nicely explained in Chvátal's textbook quoted in Chapter 9.

6.3 A Proof From the Simplex Method

The Duality Theorem of linear programming can be quickly derived from the correctness of the simplex method. The key observation is that we can extract an optimal solution of the dual linear program from the final tableau. We should remark, though, that proving the correctness of the simplex method, and in particular, the fact that one can always avoid cycling, requires considerable work.

Let us consider a primal linear program

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\mathbf{x} \leq \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0}. \tag{P}$$

After a conversion to equational form by introducing auxiliary variables $x_{n+1}, \ldots, x_{n+m}$ we arrive at the linear program

$$\text{maximize } \bar{\mathbf{c}}^T\bar{\mathbf{x}} \text{ subject to } \bar{A}\bar{\mathbf{x}} = \mathbf{b} \text{ and } \bar{\mathbf{x}} \geq \mathbf{0},$$

where $\bar{\mathbf{x}} = (x_1, \ldots, x_{n+m})$, $\bar{\mathbf{c}} = (c_1, \ldots, c_n, 0, \ldots, 0)$, and $\bar{A} = (A \mid I_m)$. If this last linear program is feasible and bounded, then according to Theorem 5.8.1, the simplex method with Bland's rule always finds some optimal solution $\bar{\mathbf{x}}^*$ with a feasible basis B. The first n components of the vector $\bar{\mathbf{x}}^*$ constitute an optimal solution $\mathbf{x}^*$ of the linear program (P). By the optimality criterion we have $\mathbf{r} \leq \mathbf{0}$ in the final simplex tableau, where $\mathbf{r}$ is the vector in the z-row of the tableau as in Section 5.5. The following lemma and the Weak Duality Theorem (Proposition 6.1.1) then easily imply the Duality Theorem.

6.3.1 Lemma. *In the described situation the vector* $\mathbf{y}^* = (\bar{\mathbf{c}}_B^T\bar{A}_B^{-1})^T$ *is a feasible solution of the dual linear program (D) and the equality* $\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*$ *holds.*

Proof. By Lemma 5.5.1, $\bar{\mathbf{x}}^*$ is given by $\bar{\mathbf{x}}_B^* = \bar{A}_B^{-1}\mathbf{b}$ and $\bar{\mathbf{x}}_N^* = \mathbf{0}$, and so

$$\mathbf{c}^T\mathbf{x}^* = \bar{\mathbf{c}}^T\bar{\mathbf{x}}^* = \bar{\mathbf{c}}_B^T\bar{\mathbf{x}}_B^* = \bar{\mathbf{c}}_B^T(\bar{A}_B^{-1}\mathbf{b}) = (\bar{\mathbf{c}}_B^T\bar{A}_B^{-1})\mathbf{b} = (\mathbf{y}^*)^T\mathbf{b} = \mathbf{b}^T\mathbf{y}^*.$$

The equality $\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*$ thus holds, and it remains to check the feasibility of $\mathbf{y}^*$, that is, $A^T\mathbf{y}^* \geq \mathbf{c}$ and $\mathbf{y}^* \geq \mathbf{0}$.

The condition $\mathbf{y}^* \geq \mathbf{0}$ can be rewritten to $I_m\mathbf{y}^* \geq \mathbf{0}$, and hence both of the conditions together are equivalent to

$$\bar{A}^T\mathbf{y}^* \geq \bar{\mathbf{c}}. \tag{6.8}$$

After substituting $\mathbf{y}^* = (\bar{\mathbf{c}}_B^T \bar{A}_B^{-1})^T$ the left-hand side becomes $\bar{A}^T(\bar{\mathbf{c}}_B^T \bar{A}_B^{-1})^T = (\bar{\mathbf{c}}_B^T \bar{A}_B^{-1} \bar{A})^T$. Let us denote this $(n+m)$-component vector by $\mathbf{w}$. For the basic components of $\mathbf{w}$ we have

$$\mathbf{w}_B = (\bar{\mathbf{c}}_B^T \bar{A}_B^{-1} \bar{A}_B)^T = (\bar{\mathbf{c}}_B^T I_m)^T = \bar{\mathbf{c}}_B,$$

and thus we even have equality in (6.8) for the basic components. For nonbasic components we have

$$\mathbf{w}_N = (\bar{\mathbf{c}}_B^T \bar{A}_B^{-1} \bar{A}_N)^T = \bar{\mathbf{c}}_N - \mathbf{r} \geq \bar{\mathbf{c}}_N$$

since $\mathbf{r} = \bar{\mathbf{c}}^N - (\bar{\mathbf{c}}_B^T \bar{A}_B^{-1} \bar{A}_N)^T$ by Lemma 5.5.1, and $\mathbf{r} \leq \mathbf{0}$ by the optimality criterion. The lemma is proved. □

6.4 The Farkas Lemma

Another approach to the Duality Theorem of linear programming consists in first proving a simplified version, called the *Farkas lemma*, and then substituting a skillfully composed matrix into it and thus deriving the theorem. A nice feature is that the Farkas lemma has very intuitive interpretations.

6.4.1 Proposition (Farkas lemma). *Let A be a real matrix with m rows and n columns, and let $\mathbf{b} \in \mathbb{R}^m$ be a vector. Then exactly one of the following two possibilities occurs:*

(F1) *There exists a vector $\mathbf{x} \in \mathbb{R}^n$ satisfying $A\mathbf{x} = \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$.*

(F2) *There exists a vector $\mathbf{y} \in \mathbb{R}^m$ such that $\mathbf{y}^T A \geq \mathbf{0}^T$ and $\mathbf{y}^T \mathbf{b} < 0$.*

It is easily seen that both possibilities cannot occur at the same time. Indeed, the vector $\mathbf{y}$ in (F2) determines a linear combination of the equations witnessing that $A\mathbf{x} = \mathbf{b}$ cannot have any nonnegative solution: All coefficients on the left-hand side of the resulting equation $(\mathbf{y}^T A)\mathbf{x} = \mathbf{y}^T \mathbf{b}$ are nonnegative, but the right-hand side is negative.

A geometric view. In order to view the Farkas lemma geometrically, we need the notion of convex hull; see Section 4.3. Further we define, for vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n \in \mathbb{R}^m$, the **convex cone** generated by $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$ as the set of all linear combinations of the $\mathbf{a}_i$ with nonnegative coefficients; that is, as

$$\Big\{ t_1\mathbf{a}_1 + t_2\mathbf{a}_2 + \cdots + t_n\mathbf{a}_n : t_1, t_2, \ldots, t_n \geq 0 \Big\}.$$

In other words, this convex cone is the convex hull of the rays $p_1, p_2, \ldots, p_n$, where $p_i = \{t\mathbf{a}_i : t \geq 0\}$ emanates from the origin and passes through the point $\mathbf{a}_i$.

6.4.2 Proposition (Farkas lemma geometrically). *Let $\mathbf{a}_1$, $\mathbf{a}_2$,..., $\mathbf{a}_n$ and $\mathbf{b}$ be vectors in $\mathbb{R}^m$. Then exactly one of the following possibilities occurs:*

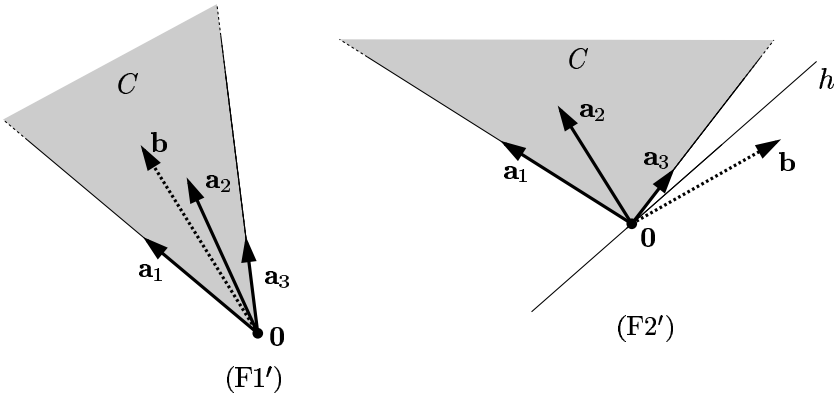
(F1′) *The point $\mathbf{b}$ lies in the convex cone C generated by $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$.*

(F2′) *There exists a hyperplane h passing through the point $\mathbf{0}$, of the form*

$$h = \{\mathbf{x} \in \mathbb{R}^m : \mathbf{y}^T\mathbf{x} = 0\}$$

for a suitable $\mathbf{y} \in \mathbb{R}^m$, such that all the vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$ (and thus the whole cone C) lie on one side and $\mathbf{b}$ lies (strictly) on the other side. That is, $\mathbf{y}^T\mathbf{a}_i \geq 0$ for all $i = 1, 2, \ldots, n$ and $\mathbf{y}^T\mathbf{b} < 0$.

A drawing illustrates both possibilities for $m = 2$ and $n = 3$:



To see that both versions mean the same, it suffices to take the columns of the matrix A for $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$. The existence of a nonnegative solution to $A\mathbf{x} = \mathbf{b}$ can be re-expressed as $\mathbf{b} = x_1\mathbf{a}_1 + x_2\mathbf{a}_2 + \cdots + x_n\mathbf{a}_n$, $x_1, x_2, \ldots, x_n \geq 0$, and this says exactly that $\mathbf{b} \in C$. The equivalence of (F2) and (F2′) hopefully doesn't need any further explanation.

A logical view. Now we explain yet another way of understanding the Farkas Lemma. We begin with something seemingly different, namely, deriving new linear inequalities from old ones. From two given inequalities, say

$$4x_1 + x_2 \leq 4 \quad \text{and} \quad -x_1 + x_2 \leq 1,$$

we can derive new inequalities by multiplying the first inequality by a *positive* real number α, the second one by a *positive* real number β, and adding the resulting inequalities together (we must be careful so that both inequality signs have the same direction!); we have already used this many times. For instance, for $\alpha = 3$ and $\beta = 2$ we derive the inequality $10x_1 + 5x_2 \leq 14$. More generally, if we start with a system of several linear inequalities, of the form $A\mathbf{x} \leq \mathbf{b}$, we can derive new inequalities by repeating this operation for various pairs, which may involve both the original inequalities and new ones derived earlier. So if we start with the system

$$4x_1 + x_2 \leq 4, \quad , -x_1 + x_2 \leq 1, \quad \text{and} \quad -2x_1 - x_2 \leq -3,$$

we can first derive $10x_1 + 5x_2 \leq 14$ from the first two as before, and then we can add to this new inequality the third inequality multiplied by 5. In this case both of the coefficients on the left-hand side cancel out, and we get the inequality

$0 \leq -1$. This last inequality *obviously* never holds, and so the original triple of inequalities cannot be satisfied by any $(x_1, x_2) \in \mathbb{R}$ either (as is easy to check using a picture).

The Farkas Lemma turns out to be equivalent to the following statement:

> Whenever a system $A\mathbf{x} \leq \mathbf{b}$ of finitely many linear inequalities is **inconsistent**, that is, there is no $\mathbf{x} \in \mathbb{R}^n$ satisfying it, we can derive from it the (obviously inconsistent) inequality $0 \leq -1$ by the above procedure.

A little thought reveals that each inequality derived by the procedure (repeated combinations of pairs) has the form $(\mathbf{y}^T A)\mathbf{x} \leq \mathbf{y}^T \mathbf{b}$ for some nonnegative vector $\mathbf{y} \in \mathbb{R}^m$, and thus, equivalently, we claim that whenever $A\mathbf{x} \leq \mathbf{b}$ is inconsistent, there exists a vector $\mathbf{y} \geq \mathbf{0}$ with $\mathbf{y}^T A = \mathbf{0}$ and $\mathbf{y}^T \mathbf{b} = -1$. This is one of many variants of the Farkas Lemma, which we will state later as Lemma 6.6.1.

The reader may wonder why we have bothered to consider repeated pairwise combinations of inequalities, instead of using a single vector $\mathbf{y}$ specifying a combination of all of the inequalities right away. The reason is that the "pairwise" formulation makes the statement more similar to a number of important and famous statements in various branches of mathematics. In logic, for example, theorems are derived (proved) from axioms by repeated application of certain simple derivation rules. In the first-order propositional calculus, there is a *completeness theorem*: Any true statement (that is, a statement valid in every model) can be derived from the axioms by a finite sequence of steps, using the appropriate derivation rules, such as *modus ponens*. In contrast, the celebrated Gödel's First Incompleteness Theorem asserts that in the Peano Arithmetic, as well as in any theory containing it, there are statements that are true but *cannot* be derived.

In analogy to this, we can view the inequalities of the original system $A\mathbf{x} \leq \mathbf{b}$ as "axioms," and we have a single derivation rule (derive a new inequality from two existing ones by a positive linear combination as above). Then the Farkas Lemma tells us that any inconsistent system of "axioms" can be refuted by a suitable derivation. (This is a "weak" completeness theorem; we could also consider a more general "completeness theorem", stating that whenever a linear inequality is valid for all $\mathbf{x} \in \mathbb{R}^n$ satisfying $A\mathbf{x} \leq \mathbf{b}$, then it can be derived from $A\mathbf{x} \leq \mathbf{b}$, but we will not go into this here.) Such a completeness result means that the theory of linear inequalities is, in a sense, "easy." Moreover, the simplex method, or also the Fourier–Motzkin elimination considered in Section 6.6, provide ways to construct such a derivation.

This view makes the Farkas Lemma a (small) cousin of various completeness theorems of logic and of other famous result, such as Hilbert's Nullstellensatz in algebraic geometry. Computer science also frequently investigates the possibility of deriving some object from given initial objects by certain derivation rules, say in the theory of formal languages.

Towards the Duality Theorem. For the proof of the Duality Theorem we first derive a more handy version of the Farkas lemma, dealing with systems of inequalities.

6.4.3 Lemma (Farkas Lemma, a variant). *The system of inequalities* $A\mathbf{x} \leq \mathbf{b}$ *and* $\mathbf{x} \geq \mathbf{0}$ *has a solution if and only if each* $\mathbf{y}$ *with* $\mathbf{y} \geq \mathbf{0}$ *and* $\mathbf{y}^T A \geq \mathbf{0}^T$ *also satisfies* $\mathbf{y}^T \mathbf{b} \geq \mathbf{0}$.

Proof. We use a trick familiar from conversions to equational form. Given an $m \times n$ matrix A, we form the matrix $\bar{A} = (A \,|\, I_m)$. Then $A\mathbf{x} \leq \mathbf{b}$ has a nonnegative solution if and only if $\bar{A}\bar{\mathbf{x}} = \mathbf{b}$ has a nonnegative solution. The latter is equivalent, by the Farkas lemma (Proposition 6.4.1), to the condition that all $\mathbf{y}$ with $\mathbf{y}^T\bar{A} \geq \mathbf{0}^T$ satisfy $\mathbf{y}^T\mathbf{b} \geq 0$. And finally, $\mathbf{y}^T\bar{A} \geq \mathbf{0}^T$ says exactly the same as $\mathbf{y}^T A \geq \mathbf{0}^T$ and $\mathbf{y} \geq \mathbf{0}$, and hence we have the desired equivalence. □

Several types of proofs of the Farkas lemma are known, and the next two sections present two of them. The lemma can be regarded as a special case of a theorem on separation of disjoint convex sets by a hyperplane, and in a way, the most natural proof is to establish the general separation theorem by tools of functional analysis and then derive the lemma from it. But this requires introducing several more notions and it takes some work.

Proof of the Duality Theorem. The idea is very simple. We express the existence of a feasible solutions $\mathbf{x}^*$ of the linear program (P) and a feasible solution $\mathbf{y}^*$ of (D) satisfying $\mathbf{c}^T\mathbf{x}^* \geq \mathbf{b}^T\mathbf{y}^*$ by a system of inequalities (which we have already done once). For this system we apply the variant of the Farkas lemma mentioned last. The execution of this plan may look complicated, but most steps are mechanical and enforced, so to speak.

Let us assume that both of the linear programs (P) and (D) have feasible solutions. It suffices to prove that the system

$$A\mathbf{x} \leq \mathbf{b}, \quad A^T\mathbf{y} \geq \mathbf{c}, \quad \mathbf{c}^T\mathbf{x} \geq \mathbf{b}^T\mathbf{y}$$

is satisfied by some nonnegative $\mathbf{x}$ and nonnegative $\mathbf{y}$. We summarize all of these inequalities using a single matrix: We rewrite the system in the form $\hat{A}\hat{\mathbf{x}} \leq \hat{\mathbf{b}}$, where

$$\hat{A} = \left(\begin{array}{c|c} A & 0 \\ \hline 0 & -A^T \\ \hline -\mathbf{c}^T & \mathbf{b}^T \end{array}\right), \quad \hat{\mathbf{x}} = \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}, \quad \hat{\mathbf{b}} = \begin{pmatrix} \mathbf{b} \\ -\mathbf{c} \\ 0 \end{pmatrix}$$

(where the zeros in $\hat{A}$ stand for zero *matrices* of appropriate dimensions). By Lemma 6.4.3, for showing solvability of the system $\hat{A}\hat{\mathbf{x}} \leq \hat{\mathbf{b}}$ it suffices to check that each nonnegative vector $\hat{\mathbf{y}}$ with $\hat{\mathbf{y}}^T\hat{A} \geq \mathbf{0}^T$ satisfies $\hat{\mathbf{y}}^T\hat{\mathbf{b}} \geq 0$.

Let us write

$$\hat{\mathbf{y}} = \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \\ z \end{pmatrix},$$

where $\mathbf{u}$ is an m-component vector, $\mathbf{v}$ is an n-component vector, and z is a scalar (a number). Then the assumption $\hat{\mathbf{y}} \geq \mathbf{0}$ means $\mathbf{u} \geq \mathbf{0}$, $\mathbf{v} \geq \mathbf{0}$, and $z \geq 0$, while $\hat{\mathbf{y}}^T\hat{A} \geq \mathbf{0}^T$ translates to $\mathbf{u}^T A \geq z\mathbf{c}^T$ and $A\mathbf{v} \leq z\mathbf{b}$ (a piece of paper might be needed to check this). The condition $\hat{\mathbf{y}}^T\hat{\mathbf{b}} \geq 0$ that is to be verified says $\mathbf{u}^T\mathbf{b} \geq \mathbf{c}^T\mathbf{v}$. We distinguish two cases:

1. $z > 0$. Then the vector $\tilde{\mathbf{u}} = \frac{1}{z}\mathbf{u}$ is a nonnegative solution of the system $A^T\mathbf{y} \geq \mathbf{c}$, and $\tilde{\mathbf{v}} = \frac{1}{z}\mathbf{v}$ is a nonnegative solution of the system $A\mathbf{x} \leq \mathbf{b}$.

By the Weak Duality Theorem we have $\tilde{\mathbf{u}}^T\mathbf{b} \geq \mathbf{c}^T\tilde{\mathbf{v}}$, and thus $\mathbf{u}^T\mathbf{b} \geq \mathbf{c}^T\mathbf{v}$ as required.

2. $z = 0$. Here we make use of the feasibility of both (P) and (D) and we choose some $\mathbf{x}_0 \geq \mathbf{0}$ satisfying $A\mathbf{x}_0 \leq \mathbf{b}$ and some $\mathbf{y}_0 \geq \mathbf{0}$ satisfying $A^T\mathbf{y}_0 \geq \mathbf{c}$. Then

$$\mathbf{u}^T\mathbf{b} \geq \mathbf{u}^T(A\mathbf{x}_0) = (\mathbf{u}^TA)\mathbf{x}_0 \geq 0$$

since $\mathbf{u}^TA \geq z\mathbf{c}^T = \mathbf{0}^T$, and similarly

$$\mathbf{c}^T\mathbf{v} \leq (\mathbf{y}_0^TA)\mathbf{v} = \mathbf{y}_0^T(A\mathbf{v}) \leq 0.$$

Altogether we have derived $\mathbf{u}^T\mathbf{b} \geq \mathbf{c}^T\mathbf{v}$, which is what we needed.

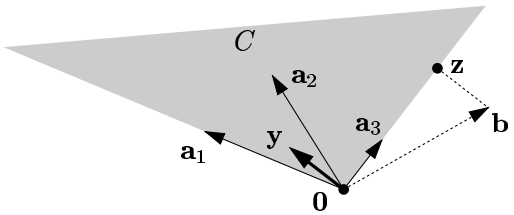
We have thus verified the assumptions of Lemma 6.4.3, and we get that the system $\hat{A}\hat{\mathbf{x}} \leq \hat{\mathbf{b}}$ has a nonnegative solution. This concludes the proof of the Duality Theorem. □

6.5 Farkas Lemma: A Geometric Proof

We are given vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n$ in $\mathbb{R}^m$, and we let C be the convex cone generated by them, i.e., the set of all linear combinations with nonnegative coefficients. Proving the Farkas lemma amounts to showing that for any vector $\mathbf{b} \notin C$ there exists a hyperplane separating it from C and passing through $\mathbf{0}$. In other words, we want to exhibit a vector $\mathbf{y} \in \mathbb{R}^m$ with $\mathbf{y}^T\mathbf{b} < 0$ and $\mathbf{y}^T\mathbf{x} \geq 0$ for all $\mathbf{x} \in C$.

This result is an instance of a *separation theorem* for convex sets. Separation theorems generally assert that disjoint convex sets can be separated by a hyperplane. There are several versions (depending on whether one requires strict or non-strict separation, etc.) and several proof strategies. Separation theorems in infinite dimensional Banach spaces are closely related to the Hahn–Banach theorem, one of the cornerstones of functional analysis.

Here we give an elementary proof of the particular result, Farkas lemma, which we need. The plan of the proof is straightforward: We let $\mathbf{z}$ be the point of C nearest to $\mathbf{b}$ (in the Euclidean distance), and we show that the vector $\mathbf{y} = \mathbf{z} - \mathbf{b}$ is as required; see an illustration:



There are two obstacles to be overcome: First, we must prove that the nearest point $\mathbf{z}$ exists, since in principle, it might happen that no point is the nearest (for example, such a situation occurs for the point 0 on the real line

and the open interval $(1,2)$; the interval contains points with distance to 0 as close to 1 as desired, but no point at distance exactly 1). And second, we have to check formally that $\mathbf{y} = \mathbf{z} - \mathbf{b}$ has the desired properties, which will be done at the end of this section.

We will need the following:

6.5.1 Fact. *Let L be a linear subspace of $\mathbb{R}^m$ and let $\mathbf{b} \in \mathbb{R}^m$ be a point. Then there exists a unique point $\mathbf{z} \in L$ such that the vector $\mathbf{z} - \mathbf{b}$ is perpendicular to L (i.e., to all vectors in it). This $\mathbf{z}$ is the (unique) point of L nearest to $\mathbf{b}$. In other words, we have $\|\mathbf{z} - \mathbf{b}\| < \|\mathbf{x} - \mathbf{b}\|$ for all $\mathbf{x} \in L \setminus \{\mathbf{z}\}$, where $\|v\| = \sqrt{\mathbf{v}^T\mathbf{v}}$ denotes the Euclidean norm of a vector $\mathbf{v} \in \mathbb{R}^m$.*

This is usually covered in courses of linear algebra, and it is not hard to prove (one can choose an orthonormal basis $(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k)$ of L and compute $\mathbf{z}$ as the orthogonal projection of $\mathbf{b}$ on L, in the form $\mathbf{z} = \sum_{i=1}^{k} (\mathbf{b}^T\mathbf{v}_i)\mathbf{v}_i$).

Next, we define a *simplicial cone* in $\mathbb{R}^m$ as a convex cone generated by some $k \leq m$ linearly independent vectors. It turns out that the existence of the nearest point is somewhat easier to prove for simplicial cones, and so the following lemma will be useful.

6.5.2 Lemma. *Let C be a convex cone in $\mathbb{R}^m$ generated by finitely many vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n$. Then C can be expressed as a union of finitely many simplicial cones.*

Proof. For every $\mathbf{x} \in C$ we are going to verify that it is contained in a simplicial cone generated by a suitable set of linearly independent vectors among the $\mathbf{a}_i$. We may assume $\mathbf{x} \neq \mathbf{0}$ (since $\{\mathbf{0}\}$ is the simplicial cone generated by the empty set of vectors).

Let $I \subseteq \{1, 2, \ldots, n\}$ be a set of minimum possible size such that $\mathbf{x}$ lies in the convex cone generated by $A_I = \{\mathbf{a}_i : i \in I\}$ (this is a standard trick in linear algebra and in convex geometry). That is, there exist positive coefficients α_i, $i \in I$, with $\mathbf{x} = \sum_{i \in I} \alpha_i \mathbf{a}_i$. The α_i are even strictly positive since if some $\alpha_i = 0$, we could delete i from I. For contradiction, we suppose that there is a nontrivial linear dependence $\sum_{i \in I} \beta_i \mathbf{a}_i = \mathbf{0}$, where not all β_i are 0. Then there exists a real λ such that all the expressions $\alpha_i - \lambda\beta_i$ are nonnegative and at least one of them is zero. (To see this, we can first consider the case where some β_i is strictly positive, we start with $\lambda = 0$, we let it grow, and see what happens. The case of a strictly negative β_i is analogous with λ decreasing from the initial value 0.) Then the equation

$$\mathbf{x} = \sum_{i \in I} (\alpha_i - \lambda\beta_i)\mathbf{a}_i$$

expresses $\mathbf{x}$ as a linear combination with positive coefficients of less than $|I|$ vectors. □

6.5.3 Lemma. *Let C be a simplicial cone in $\mathbb{R}^m$ and let $\mathbf{b} \notin C$ be a point. Then there exists a point $\mathbf{z} \in C$ nearest to $\mathbf{b}$. (It is also unique but we won't need this.)*

Proof. The distance of $\mathbf{b}$ to C is defined as $\delta = \inf_{\mathbf{x}\in C} \|\mathbf{x}-\mathbf{b}\|$. Let $(\mathbf{z}_1, \mathbf{z}_2, \ldots)$ be a sequence of points of C such that $\|\mathbf{z}_j - \mathbf{b}\|$ converges to δ as $j \to \infty$. Let us write $\mathbf{z}_j = \sum_{i=1}^{k} t_i^{(j)} \mathbf{a}_i$, where $\mathbf{a}_1, \ldots, \mathbf{a}_k$ are the linearly independent vectors generating C. By linear independence of the $\mathbf{a}_i$, the $t_i^{(j)}$ are determined uniquely.

Next, we observe that points of C at bounded distance from $\mathbf{0}$ have bounded coefficients. More precisely, we have:

> *Claim.* There exists a real number T (depending on the $\mathbf{a}_i$) such that if $\mathbf{x} = \sum_{i=1}^{k} t_i \mathbf{a}_i \in C$, then $t_i \le T\|\mathbf{x}\|$ for all $i = 1, 2, \ldots, k$.

To see this, let us consider the case $i = 1$ for simpler notation, and let L be the linear span of $\mathbf{a}_2, \ldots, \mathbf{a}_k$. Since $\mathbf{a}_1 \notin L$, by Fact 6.5.1 it has some positive distance η to L. Then

$$\|\mathbf{x}\| = \|t_1\mathbf{a}_1 + \cdots + t_k\mathbf{a}_k\| \ge \inf_{\mathbf{y}\in L} \|t_1\mathbf{a}_1 - \mathbf{y}\| = t_1 \cdot \inf_{\mathbf{y}\in L} \|\mathbf{a}_1 - \mathbf{y}\| = t_1\eta.$$

By the claim just proved, we have $0 \le t_i^{(j)} \le T\|\mathbf{z}_j\| \le T(\|\mathbf{z}_j - \mathbf{b}\| + \|\mathbf{b}\|)$, and hence each of the sequences $(t_i^{(1)}, t_i^{(2)}, \ldots)$ is bounded. Therefore, we can select indices $j_1 < j_2 < \cdots$ such that each of the sequences $(t_i^{(j_1)}, t_i^{(j_2)}, \ldots)$, $i = 1, 2, \ldots, k$, converges to some $\bar{t}_i \in [0, \infty)$. (This is a basic result in analysis, related to compactness: From every bounded infinite sequence of real numbers one can select an infinite convergent subsequence. When several sequences should be handled simultaneously, we deal with them one at a time, selecting a subsequence of a subsequence of a subsequence etc.)

We set $\mathbf{z} = \sum_{i=1}^{k} \bar{t}_i \mathbf{a}_i$. Then $\|\mathbf{z}-\mathbf{b}\| = \lim_{\ell\to\infty} \|\mathbf{z}_{j_\ell} - \mathbf{b}\| = \delta$, since the distance of a point $\mathbf{x} = \sum_{i=1}^{k} t_i \mathbf{a}_i$ from $\mathbf{b}$ depends continuously on the coefficients t_i. □

6.5.4 Corollary. *The conclusion of Lemma 6.5.3 holds for any convex cone in $\mathbb{R}^m$ generated by finitely many vectors, not only for simplicial cones.*

Proof. A given finitely generated cone C can be written as a union $C = C_1 \cup \cdots \cup C_r$ of simplicial cones. Let $\mathbf{z}_k$ be the point of C_k nearest to $\mathbf{b}$, $k = 1, 2, \ldots, r$. Then the point nearest to $\mathbf{b}$ from the set $\{\mathbf{z}_1, \ldots, \mathbf{z}_r\}$ is the nearest point of C to $\mathbf{b}$. □

Proof of Proposition 6.4.2 (Farkas Lemma). As we have announced, we consider a convex cone C generated by $\mathbf{a}_1, \ldots, \mathbf{a}_n \in \mathbb{R}^m$ and a vector $\mathbf{b} \notin C$, we let $\mathbf{z}$ be the point of C nearest to $\mathbf{b}$, and we set $\mathbf{y} = \mathbf{z} - \mathbf{b} \neq \mathbf{0}$.

First we check that $\mathbf{y}^T\mathbf{z} = 0$. Indeed, this is clear for $\mathbf{z} = \mathbf{0}$. For $\mathbf{z} \neq \mathbf{0}$, we first note that $\mathbf{z}$ has to be the point of the line $L = \{t\mathbf{z} : t \in \mathbb{R}\}$ nearest to $\mathbf{b}$. Indeed, by Fact 6.5.1 there is a unique point $\mathbf{z}' \in L$ nearest to $\mathbf{b}$. If $\mathbf{z}'$ corresponds to a nonnegative value of the parameter t, then $\mathbf{z}' \in C$ and thus $\mathbf{z}' = \mathbf{z}$. If, on the other hand, $\mathbf{z}'$ corresponds to some $t < 0$, then it is easy to

check that $\mathbf{0}$ has to be closer to $\mathbf{b}$ than $\mathbf{z}$, and so this case cannot occur. We thus get $\mathbf{z} = \mathbf{z}'$ and $\mathbf{z} - \mathbf{b}$ is perpendicular to L by Fact 6.5.1.

To verify $\mathbf{y}^T\mathbf{b} < 0$, we compute $0 < \mathbf{y}^T\mathbf{y} = \mathbf{y}^T\mathbf{z} - \mathbf{y}^T\mathbf{b} = -\mathbf{y}^T\mathbf{b}$.

Next, let $\mathbf{x} \in C$. The angle $\angle\mathbf{bzx}$ has to be at least 90 degrees, for otherwise, points on the segment $\mathbf{zx}$ sufficiently close to $\mathbf{z}$ would lie closer to $\mathbf{b}$ than $\mathbf{z}$ (a formal verification is left to the reader). Thus $0 \geq (\mathbf{b} - \mathbf{z})^T(\mathbf{x} - \mathbf{z}) = -\mathbf{y}^T\mathbf{x} + \mathbf{y}^T\mathbf{z} = -\mathbf{y}^T\mathbf{x}$. The Farkas lemma is proved. □

6.6 Farkas Lemma From the Fourier–Motzkin Elimination

The Fourier–Motzkin elimination is a stepwise method that can be used to find a solution to a linear inequality system $A\mathbf{x} \leq \mathbf{b}$, or to certify that there is no solution. It is similar in spirit to Gaussian elimination for systems of linear equations, and it is just as simple. Like in Gaussian elimination, variables are removed one at a time, but there is a price to pay: to get rid of one variable, we typically have to introduce many new inequalities, so that the method becomes impractical already for moderately large systems. The Fourier–Motzkin elimination can be considered as a simple but inefficient alternative to the simplex method. For the purpose of proving statements about systems of inequalities, efficiency is not a concern, so it is the simplicity of the Fourier–Motzkin elimination that makes it a very practical tool. Here are the details.

Claim. *Let $A\mathbf{x} \leq \mathbf{b}$ be a system with $n \geq 1$ variables and m inequalities. There is a system $A'\mathbf{x}' \leq \mathbf{b}'$ with $n - 1$ variables and at most $\max(m, m^2/4)$ inequalities, with the following properties.*

(i) $A\mathbf{x} \leq \mathbf{b}$ has a solution if and only if $A'\mathbf{x}' \leq \mathbf{b}'$ has a solution, and

(ii) each inequality of $A'\mathbf{x}' \leq \mathbf{b}'$ is a positive linear combination of some inequalities from $A\mathbf{x} \leq \mathbf{b}$.

Proof. We classify the inequalities into three groups, depending on the coefficient of x_1. We call the i-th inequality of $A\mathbf{x} \leq \mathbf{b}$ a *ceiling* if $a_{i1} > 0$, and we call it a *floor* if $a_{i1} < 0$. Otherwise (if $a_{i1} = 0$), it is a *level*. Let $C, F, L \subseteq \{1, \ldots, m\}$ collect the indices of ceilings, floors, and levels. We may assume that

$$a_{i1} = \begin{cases} 1 & \text{if } i \in C, \\ -1 & \text{if } i \in F, \\ 0 & \text{if } i \in L. \end{cases} \tag{6.9}$$

This situation can be reached by multiplying each inequality in $A\mathbf{x} \leq \mathbf{b}$ with a suitable positive number, which does not change the set of solutions.

Now we can eliminate x_1 between all pairs of ceilings and floors, by simply adding up the two inequalities for each pair.

If $\mathbf{x}'$ is the (possibly empty) vector $(x_2, \ldots, x_n)$, and $\mathbf{a}'_i$ is the (possibly empty) vector $(a_{i2}, \ldots, a_{in})$, then the following inequalities are implied by $A\mathbf{x} \leq \mathbf{b}$.

$$\mathbf{a}'^T_c \mathbf{x}' + \mathbf{a}'^T_f \mathbf{x}' \leq b_c + b_f, \quad c \in C, f \in F. \tag{6.10}$$

The level inequalities of $A\mathbf{x} \leq \mathbf{b}$ can be rewritten as

$$\mathbf{a}'^T_\ell \mathbf{x}' \leq b_\ell, \quad \ell \in L. \tag{6.11}$$

Therefore, if $A\mathbf{x} \leq \mathbf{b}$ has some solution, then the system of $|C| \cdot |F| + |L|$ inequalities in $n-1$ variables given by (6.10) and (6.11) has a solution as well. Conversely, if the latter system has a solution $\tilde{\mathbf{x}} = (\tilde{x}_2, \ldots, \tilde{x}_n)$, we can determine a suitable value x_1^* so that the vector $(x_1^*, \tilde{x}_2 \ldots, \tilde{x}_n)$ solves $A\mathbf{x} \leq \mathbf{b}$. To find x_1^*, we first observe that (6.10) is equivalent to

$$\mathbf{a}'^T_f \mathbf{x}' - b_f \leq b_c - \mathbf{a}'^T_c \mathbf{x}', \quad c \in C, f \in F.$$

This in particular implies

$$\max_{f \in F} \left(\mathbf{a}'^T_f \tilde{\mathbf{x}} - b_f \right) \leq \min_{c \in C} \left(b_c - \mathbf{a}'^T_c \tilde{\mathbf{x}} \right).$$

We let x_1^* be any value between these bounds. It follows that

$$\begin{aligned} x_1^* + \mathbf{a}'^T_c \tilde{\mathbf{x}} &\leq b_c, \quad c \in C, \\ -x_1^* + \mathbf{a}'^T_f \tilde{\mathbf{x}} &\leq b_f, \quad f \in F. \end{aligned}$$

By our assumption (6.9), we therefore have a feasible solution to the original system $A\mathbf{x} \leq \mathbf{b}$. We note that this argument also works for $C = \emptyset$ or $F = \emptyset$, with the usual convention that $\max_{t \in \emptyset} f(t) = -\infty$ and $\min_{t \in \emptyset} f(t) = \infty$.

Now we proceed to the proof of the Farkas lemma. The variant that results most naturally from the Fourier–Motzkin elimination is the following.

6.6.1 Lemma (Farkas Lemma, yet another variant). *The system of inequalities $A\mathbf{x} \leq \mathbf{b}$ has a solution if and only if each $\mathbf{y}$ with $\mathbf{y} \geq \mathbf{0}$ and $\mathbf{y}^T A = \mathbf{0}^T$ also satisfies $\mathbf{y}^T \mathbf{b} \geq 0$.*

Let us first check that this version implies the variant of Lemma 6.4.3 which we have used to prove the Duality Theorem. For this, we employ familiar arguments. Along these lines, one could establish a unifying *Farkas Lemma for everyone* and show that all three variants of the Farkas Lemma we have introduced are mutually equivalent. We leave this as an exercise.

Given an $m \times n$ matrix A, we form the matrix $\bar{A}$ and the vector $\bar{\mathbf{b}}$ according to

$$\bar{A} = \left(\frac{A}{-I_n} \right), \bar{\mathbf{b}} = \left(\frac{\mathbf{b}}{\mathbf{0}} \right).$$

Then $A\mathbf{x} \leq \mathbf{b}$ has a nonnegative solution if and only if $\bar{A}\mathbf{x} \leq \bar{\mathbf{b}}$ has a solution. The latter is equivalent, by Lemma 6.6.1, to the condition that all $\bar{\mathbf{y}} \geq \mathbf{0}$ with $\bar{\mathbf{y}}^T \bar{A} = \mathbf{0}^T$ satisfy $\bar{\mathbf{y}}^T \bar{\mathbf{b}} \geq 0$. Writing

$$\bar{\mathbf{y}} = \left(\frac{\mathbf{y}}{\mathbf{y}'} \right),$$

$\mathbf{y}$ being a vector with m components, we have

$$\bar{\mathbf{y}} \geq \mathbf{0}, \quad \bar{\mathbf{y}}^T \bar{A} = \mathbf{0}^T \quad \text{exactly if} \quad \mathbf{y} \geq \mathbf{0}, \quad \mathbf{y}^T A \geq \mathbf{0}^T$$

and

$$\bar{\mathbf{y}}^T \bar{\mathbf{b}} = \mathbf{y}^T \mathbf{b}.$$

From this and our chain of equivalences, we deduce that $A\mathbf{x} \leq \mathbf{b}$ has a nonnegative solution if and only if all $\mathbf{y} \geq \mathbf{0}$ with $\mathbf{y}^T A \geq \mathbf{0}^T$ satisfy $\mathbf{y}^T \mathbf{b} \geq 0$, and this is the statement of Lemma 6.4.3.

Proof of Lemma 6.6.1. One direction is easy. If $A\mathbf{x} \leq \mathbf{b}$ has some solution $\mathbf{x}^*$, and $\mathbf{y} \geq \mathbf{0}$ satisfies $\mathbf{y}^T A = \mathbf{0}^T$, we get

$$0 = \mathbf{0}^T \mathbf{x}^* = \mathbf{y}^T A \mathbf{x}^* \leq \mathbf{y}^T \mathbf{b}.$$

If $A\mathbf{x} \leq \mathbf{b}$ has no solution, then our task is to construct a vector $\mathbf{y}$ satisfying

$$\mathbf{y} \geq \mathbf{0}, \quad \mathbf{y}^T A = \mathbf{0}^T, \quad \text{and } \mathbf{y}^T \mathbf{b} < 0. \tag{6.12}$$

We call such a vector a *witness* (for infeasibility). To find a witness, we use induction on the number of variables. Let us first consider the base case in which the system $A\mathbf{x} \leq \mathbf{b}$ has no variables, meaning that it is of the form $\mathbf{0} \leq \mathbf{b}$ with $b_i < 0$ for some i. We set $\mathbf{y} = \mathbf{e}_i$ (the i-th unit vector), and this clearly satisfies the requirements for $\mathbf{y}$ being a witness (the condition $\mathbf{y}^T A = \mathbf{0}^T$ is vacuous, because A has no column).

If $A\mathbf{x} \leq \mathbf{b}$ has at least one variable, we perform a step of the Fourier–Motzkin elimination, resulting in the (infeasible) system $A'\mathbf{x}' \leq \mathbf{b}'$, consisting of the inequalities (6.10) and (6.11). Because the latter system has one variable less, we inductively find a witness $\mathbf{y}'$ for it. Recall that all inequalities of $A'\mathbf{x}' \leq \mathbf{b}'$ are positive linear combinations of original inequalities; equivalently, there is a matrix M with all entries nonnegative and

$$(\mathbf{0}|A') = MA, \quad \mathbf{b}' = M\mathbf{b}.$$

We claim that $\mathbf{y} = M^T \mathbf{y}'$ is a witness for the original system $A\mathbf{x} \leq \mathbf{b}$. Indeed, we have $\mathbf{y}^T A = \mathbf{y}'^T MA = \mathbf{y}'^T (\mathbf{0}|A') = \mathbf{0}^T$ and $\mathbf{y}^T \mathbf{b} = \mathbf{y}'^T M\mathbf{b} = \mathbf{y}'^T \mathbf{b}' < 0$, because $\mathbf{y}'$ is a witness for $A'\mathbf{x}' \leq \mathbf{b}'$. The condition $\mathbf{y} \geq \mathbf{0}$ follows from $\mathbf{y}' \geq \mathbf{0}$ by nonnegativity of M. □

7

Not Only the Simplex Method

Tens of different algorithms have been suggested for linear programming over the years. Most of them didn't work very well, and only very few have turned out as serious competitors to the simplex method, the historically first algorithm. But at least two methods raised great excitement at the time of discovery and they are surely worth mentioning.

The first of them, the *ellipsoid method*, cannot compete with the simplex method in practice, but it had immense theoretical significance. It is the first linear programming algorithm for which it was proved that it always runs in polynomial time (which is not known about the simplex method up to the present, and for many pivot rules it is not even true).

The second is the *interior point method*, or rather, we should say interior point *methods*, since it is an entire group of algorithms. For some of them a polynomial bound on the running time has also been proved, but moreover, these algorithms successfully compete with the simplex method in practice. It seems that for some types of linear programs the simplex method is better, while for others interior point methods are the winners.

Let us remark that several other algorithms, closely related to the simplex method, are used for linear programming as well. The *dual simplex method* can roughly be described as the simplex method applied to the dual linear program. But details of the implementation, which are crucial for the speed of the algorithm in practice, are somewhat different. The dual simplex method is particularly suitable for linear programs that in equational form have $n - m$ significantly smaller than m.

The *primal-dual method* makes pivot steps from one basis to another, but these bases need not correspond either to feasible solutions (as in the simplex method) or to feasible solutions of the dual linear program (as in the dual simplex method). This greater freedom can be useful, for instance, in approximation algorithms for combinatorial optimization problems.

7.1 The Ellipsoid Method

The ellipsoid method was invented in 1970 by Shor, Judin, and Nemirovski as an algorithm for certain nonlinear optimization problems. In 1979 Leonid Khachyian outlined, in a short note, how linear programs can be solved by this method, and in provably polynomial time. The world press made a sensation

out of this since the journalists contorted the result and presented it as an unprecedented breakthrough in practical computational methods (giving the Soviets a technological edge over the West...).[1] However, the ellipsoid method has never been interesting for the practice of linear programming—Khachyian's discovery was indeed extremely significant, but for the theory of computational complexity. It solved an open problem that many people had attacked in vain for many years. The solution was conceptually utterly different from previous approaches, which were mostly variations of the simplex method.

Input size and polynomial algorithms. In order to describe what we mean by a polynomial algorithm for linear programming, we have to define the **input size** of a linear program. Roughly speaking, it is the total number of bits needed for writing down the input to a linear programming algorithm.

First we define the *bit size* of an integer i as

$$\langle i \rangle = \lceil \log_2(|i| + 1) \rceil + 1,$$

which is the number of bits of i written in binary, including one bit for the sign. For a rational number r, i.e., a fraction $r = p/q$, the bit size is defined as $\langle r \rangle = \langle p \rangle + \langle q \rangle$. For an n-component rational vector $\mathbf{v}$ we put $\langle \mathbf{v} \rangle = \sum_{i=1}^{n} \langle v_i \rangle$, and similarly, $\langle A \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} \langle a_{ij} \rangle$ for a rational $m \times n$ matrix A. If we consider a linear program L, say in the form

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\mathbf{x} \leq \mathbf{b},$$

and if we restrict ourselves to the case of A, $\mathbf{b}$, and $\mathbf{c}$ rational (which is a reasonable assumption from a computational perspective), then the bit size of L is $\langle L \rangle = \langle A \rangle + \langle \mathbf{b} \rangle + \langle \mathbf{c} \rangle$.

We say that an algorithm is a **polynomial algorithm for linear programming** if a polynomial $p(x)$ exists such that for every linear program L with rational A, $\mathbf{b}$, and $\mathbf{c}$ the algorithm finds a correct solution in at most $p(\langle L \rangle)$

[1] The following quotation from

E. L. Lawler: The Great Mathematical Sputnik of 1979, *Math. Intelligencer* 2(1980) 191–198,

which is a remarkable article about the history of Khachyian's result, is not only of historical interest:

The *Times* story appears to have been based on certain unshakable preconceptions of its writer Malcolm W. Browne. Browne called George Dantzig of Stanford University, a great pioneering authority on linear programming, and tried to force him into various admissions. Dantzig's version of the interview bears repeating:

> "What about the traveling salesman problem?" asked Browne. "If there is a connection, I don't know what it is," said Dantzig. ("The Russian discovery proposed an approach for [solving] a class of problems related to the 'Traveling Salesman Problem," reported Browne.) "What about cryptography?" asked Browne. "If there is a connection, I don't know what it is," said Dantzig. ("The theory of codes could eventually be affected," reported Browne.) "Is the Russian method practical?" asked Browne. "No," said Dantzig. ("Mathematicians describe the discovery ...as a method by which computers can find solutions to a class of very hard problems that has hitherto been attacked on a hit-or-miss basis," reported Browne.)

steps. The steps are counted in some of the usual models of computation, for example, as steps of a Turing machine (usually the chosen computational model is not crucial; whatever is polynomial in one model, is also polynomial in other reasonable models). We stress right away that a single arithmetic operation is not counted as a single step here! We count as steps operations with single bits, and hence, addition of two k-bit integers requires order of k steps.

Let us digress briefly from linear programming and let us consider the Gauss elimination, a well-known algorithm for solving systems of linear equations. For a system $A\mathbf{x} = \mathbf{b}$, where (for simplicity) A is an $n \times n$ matrix and both A and $\mathbf{b}$ are rational, we naturally define the input size as $\langle A \rangle + \langle \mathbf{b} \rangle$. Is the Gauss elimination a polynomial algorithm? This is a tricky question! Although this algorithm needs only order of n^3 arithmetic operations, the catch is that too large intermediate values could come up during the computation, even if all entries in A and in $\mathbf{b}$ are small integers. If, for example, integers with as many as 2^n bits ensued, which can indeed happen in a cumbersome implementation of the Gauss elimination, the computation would need exponentially many steps, although it would involve only n^3 arithmetic operations. (All of this concerns *exact* computations, while many implementations use floating point arithmetic and hence the numbers are continually rounded. But then there is no guarantee that the results are correct.) We do not want to scare the reader needlessly: It is known how the Gauss elimination can be implemented in polynomial time. We want only to point out that this is not self-evident (and not too simple, either), and call attention to one kind of troubles that may develop in attempts at proving polynomiality.

The ellipsoid method, as well as some of the interior point methods, are polynomial, while the simplex method with Bland's rule (and with many other pivot rules too) is not polynomial.[2]

Strongly polynomial algorithms. For algorithms whose input is described by a sequence of integers or rationals, such as algorithms for linear programming, the number of arithmetic operations (addition, subtraction, multiplication, division, exponentiation) is also considered, together with the number of bit operations. This often gives a more realistic picture of the running time, because contemporary computers usually execute an arithmetic operation as an elementary step, provided that the operands are not too large.

A suitable implementation of the Gauss elimination is, on the one hand, a polynomial algorithm in the sense discussed above, and on the other hand, the number of arithmetic operations is bounded by a polynomial, namely by the polynomial Cn^3 for a suitable constant C, where n is the number of equations in the system and also the number of variables. The number of arithmetic operations thus depends only on n, and it is the same for input numbers with 10 bits as for input numbers with a million bits. We say that the Gauss elimination is a **strongly polynomial** algorithm for solving systems of linear equations.

A strongly polynomial algorithm for linear programming would be one that, first, would be polynomial in the sense defined above, and second, for every linear program with n variables and m constraints it would find a solution using at most $p(m+n)$ arithmetic operations, where $p(x)$ is a fixed polynomial. But no strongly polynomial algorithm for linear programming is known and finding one is a major open problem.

[2]The non-polynomiality is proved by means of the Klee–Minty cube; see Section 5.9. One has to check that an n-dimensional Klee–Minty cube can be represented by input of size polynomial in n.

The ellipsoid method is not strongly polynomial. For every natural number M one can find a linear program with only 2 variables and 2 constraints for which the ellipsoid method executes at least M arithmetic operations (the coefficients in such linear programs must have bit size tending to infinity as $M \to \infty$). In particular, the number of arithmetic operations for the ellipsoid method cannot be bounded by any polynomial in $m+n$.

Ellipsoids. A two-dimensional ellipsoid is an ellipse plus its interior. An ellipsoid in general can most naturally be introduced as an affine transformation of a ball. We let

$$B^n = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x}^T\mathbf{x} \le 1\}$$

be the n-dimensional ball of unit radius centered at $\mathbf{0}$. Then an n-dimensional **ellipsoid** is a set of the form

$$E = \{M\mathbf{x} + \mathbf{s} : \mathbf{x} \in B^n\},$$

where M is a nonsingular $n \times n$ matrix and $\mathbf{s} \in \mathbb{R}^n$ is a vector. The mapping $\mathbf{x} \mapsto M\mathbf{x} + \mathbf{s}$ is a composition of a linear isomorphism and a translation, and it is called an **affine map.**

By manipulating the definition we can describe the ellipsoid by an inequality:

$$\begin{aligned} E &= \{\mathbf{y} \in \mathbb{R}^n : M^{-1}(\mathbf{y}-\mathbf{s}) \in B^n\} = \\ &= \{\mathbf{y} \in \mathbb{R}^n : (\mathbf{y}-\mathbf{s})^T (M^{-1})^T M^{-1}(\mathbf{y}-\mathbf{s}) \le 1\} = \\ &= \{\mathbf{y} \in \mathbb{R}^n : (\mathbf{y}-\mathbf{s})^T Q^{-1}(\mathbf{y}-\mathbf{s}) \le 1\}, \end{aligned} \tag{7.1}$$

where we have set $Q = MM^T$. It is well known and easy to check that such a Q is a positive definite matrix, that is, a symmetric square matrix satisfying $\mathbf{x}^T Q\mathbf{x} > 0$ for all nonzero vectors $\mathbf{x}$. Conversely, from matrix theory it is known that each positive definite matrix Q can be factored as $Q = MM^T$ for some nonsingular square matrix M. Therefore, an equivalent definition is that an ellipsoid is a set described by (7.1) for some positive definite Q and some $\mathbf{s}$.

Geometrically, $\mathbf{s}$ is the center of the ellipsoid E. If Q is a diagonal matrix and $\mathbf{s} = \mathbf{0}$, then we have an ellipsoid in *axial position*, of the form

$$\left\{\mathbf{y} \in \mathbb{R}^n : \frac{y_1^2}{q_{11}} + \frac{y_2^2}{q_{22}} + \cdots + \frac{y_n^2}{q_{nn}} \le 1\right\}.$$

The axes of this ellipsoid are parallel to the coordinate axes. The numbers $\sqrt{q_{11}}$, $\sqrt{q_{22}}, \ldots, \sqrt{q_{nn}}$ are the lengths of the *semiaxes* of the ellipsoid E, which may sound familiar to those accustomed to the equation of an ellipse of the form $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$. As is taught in linear algebra in connection with eigenvalues, each positive definite matrix Q can be diagonalized by an orthogonal basis change. That is, there exists an orthogonal matrix T so that the matrix TQT^{-1} is diagonal, with the eigenvalues of Q on the diagonal. Geometrically, T represents a rotation of the coordinate system that brings the ellipsoid in axial position.

A lion in Sahara. A traditional mathematical anecdote gives directions for hunting a lion in Sahara. We fence all of Sahara, we divide it into two halves

by another fence, and we detect one half that has no lion in it. Then we divide the other half by a fence, and we continue in this manner until the fenced piece of ground is so small that the lion cannot move and so he is caught, or if there is no lion in it, we have proved that there was none in Sahara either. Although the qualities of this hunting guide can be disputed, for us it is essential that it gives a reasonably good description of the ellipsoid method. But in the real ellipsoid method we insist that the currently fenced piece is always an ellipsoid, even at the price that the lion can sometimes return to places from where he was expelled earlier; it is only guaranteed that the area of his territory shrinks all the time.

The ellipsoid method doesn't directly solve a linear program, but rather it seeks a solution of a system of linear inequalities $A\mathbf{x} \leq \mathbf{b}$. But as we know, this is sufficient for solving a linear program (see Section 6.1). For a simpler exposition we will first consider the following softened version of the problem:

Together with the matrix A and vector $\mathbf{b}$ we are given rational numbers $R > \varepsilon > 0$. We assume that the set $P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\}$ is contained in the ball $B(\mathbf{0}, R)$ centered at $\mathbf{0}$ with radius R. If P contains a ball of radius ε, then the algorithm has to return a point $\mathbf{y} \in P$. However, if P contains no ball of radius ε, then the algorithm may return either some $\mathbf{y} \in P$, or the answer NO SOLUTION.

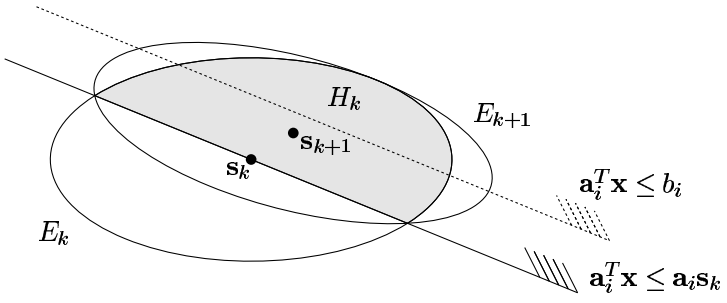
The ball $B(\mathbf{0}, R)$ thus plays the role of Sahara and we assume that the lion, if present, is at least ε large. If there is only a smaller lion in Sahara, we may catch it as well, but we need not.

Under these assumptions, the ellipsoid method generates a sequence of ellipsoids $E_0, E_1, \ldots, E_t$, where $P \subseteq E_k$ for each k, as follows:

1. Set $k = 0$ and $E_0 = B(\mathbf{0}, R)$.

2. Let the current ellipsoid E_k be of the form $E_k = \{\mathbf{y} \in \mathbb{R}^n : (\mathbf{y} - \mathbf{s}_k)^T Q_k^{-1} (\mathbf{y} - \mathbf{s}_k) \leq 1\}$. If $\mathbf{s}_k$ satisfies all inequalities of the system $A\mathbf{x} \leq \mathbf{b}$, return $\mathbf{s}_k$ as a solution; **stop.**

3. Otherwise, choose an inequality of the system that is violated by $\mathbf{s}_k$. Let it be the ith inequality; so we have $\mathbf{a}_i^T \mathbf{s}_k > b_i$. Define E_{k+1} as the ellipsoid of the smallest possible volume containing the "half-ellipsoid" $H_k = E_k \cap \{\mathbf{x} \in \mathbb{R}^n : \mathbf{a}_i^T \mathbf{x} \leq \mathbf{a}_i^T \mathbf{s}_k\}$; see a picture:



4. If the volume of E_{k+1} is smaller than the volume of a ball of radius ε, return NO SOLUTION; **stop.** Otherwise, increase k by 1 and continue with Step 2.

Let H'_k denote the intersection of the ellipsoid E_k with the half-space $\{x \in \mathbb{R}^n : \mathbf{a}_i\mathbf{x} \leq b_i\}$ defined by the ith inequality of the system. If $P \subseteq E_k$, then also $P \subseteq H'_k$, and the more so $P \subseteq H_k$. Why the smallest ellipsoid containing H_k is taken for E_{k+1}, instead of the smallest ellipsoid containing H'_k? Purely in order to simplify the analysis of the algorithm, since the equation of E_{k+1} comes out less complicated this way.

The ellipsoid E_{k+1}, i.e., the ellipsoid of the smallest volume containing H_k, is always determined uniquely. For illustration we mention that it is given by

$$\begin{aligned} \mathbf{s}_{k+1} &= \mathbf{s}_k - \frac{1}{n+1} \cdot \frac{Q\mathbf{s}_k}{\sqrt{\mathbf{s}_k^T Q \mathbf{s}_k}}, \\ Q_{k+1} &= \frac{n^2}{n^2-1}\left(Q_k - \frac{2}{n+1} \cdot \frac{Q\mathbf{s}_k\mathbf{s}_k^T Q}{\mathbf{s}_k^T Q \mathbf{s}_k}\right). \end{aligned}$$

We also leave without proof a fact crucial for the proof of correctness and efficiency of the ellipsoid method: We always have

$$\frac{\text{volume}(E_{k+1})}{\text{volume}(E_k)} \leq e^{-1/(2n+2)}.$$

Hence the volume of the ellipsoid E_k is at least $e^{k/(2n+2)}$ times smaller than the volume of the initial ball $B(\mathbf{0}, R)$. Since the volume of an n-dimensional ball is proportional to the nth power of the radius, for k satisfying $R \cdot e^{-k/n(2n+2)} < \varepsilon$ the volume of E_k is smaller than that of a ball of radius ε. Such k provides an upper bound on the maximum number t of iterations, and this yields $t \leq n(2n+2)\ln(R/\varepsilon)$. This is obviously bounded by a polynomial in $n + \langle R\rangle + \langle\varepsilon\rangle$.

So much for the simple and beautiful idea of the ellipsoid method—now we are coming to manageable but unpleasant complications. First of all, we cannot compute the ellipsoid E_{k+1} exactly, at least not in rational arithmetic since the defining formulas contain square roots. To get around this, E_{k+1} is computed only approximately with a suitable precision. But one has to be careful so that P is still guaranteed to be contained in E_{k+1}, and thus the approximate E_{k+1} has to be expanded slightly.

Another trouble arises, for example, when the same inequality is used for cutting the current ellipsoid in many iterations in a row. Then the ellipsoids may become too long (needle-like), and they have to be shortened artificially.

Yet another problem is that we don't really want to solve the softened problem with R and ε, but an arbitrary system of linear inequalities without any restrictions. Here the bound on the bit size of the entries of A and $\mathbf{b}$ come into play, through the following facts:

(E1) (If a solution exists, then there is a not too large solution.) Let $\varphi = \langle A\rangle + \langle\mathbf{b}\rangle$ denote the input size for the system $A\mathbf{x} \leq \mathbf{b}$. Then this system has a solution if and only if the system

$$\begin{aligned} A\mathbf{x} &\leq \mathbf{b} \\ -K \leq x_1 &\leq K \\ -K \leq x_2 &\leq K \\ &\vdots \\ -K \leq x_n &\leq K, \end{aligned}$$

has a solution, where $K = 2^{\varphi}$. Clearly, all solutions of the latter system are contained in the ball $B(\mathbf{0}, R)$, where $R = K\sqrt{n}$.

(E2) (If a solution exists, then the solution set of a slightly relaxed system contains a small ball.) Let us put $\eta = 2^{-5\varphi}$, $\varepsilon = 2^{-6\varphi}$, and let $\boldsymbol{\eta}$ be the n-component vector with all components equal to η. Then the system $A\mathbf{x} \leq \mathbf{b}$ has a solution if and only if the system $A\mathbf{x} \leq \mathbf{b} + \boldsymbol{\eta}$ has a solution, and in such case the solution set of the latter system contains a ball of radius ε.

It is not hard to see how these facts can be used for solving an arbitrary system $A\mathbf{x} \leq \mathbf{b}$ by the ellipsoid method. Instead of this system we solve the softened problem by the ellipsoid method, but for a new system $A\mathbf{x} \leq \mathbf{b} + \boldsymbol{\eta}$, $-K - \eta \leq x_1 \leq K + \eta, -K - \eta \leq x_2 \leq K + \eta, \ldots, -K - \eta \leq x_n \leq K + \eta$, where K, R, ε, and η are chosen suitably (first we add the constraints $-K \leq x_i \leq K$ as in (E1), and then we apply (E2) to the resulting system). It is important that the bit size of R and ε, as well as the input size of the new system, are bounded by a polynomial function of φ. Thus the ellipsoid method runs in polynomial time, and it always find a solution to $A\mathbf{x} \leq \mathbf{b}$ if it exists.

We will not prove facts (E1) and (E2) here, but we sketch the basic ideas. For (E1) we first discuss the case $n = 2$ (in the plane). Let us consider a system of m inequalities

$$a_{i1}x + a_{i2}y \leq b_i, \quad i = 1, 2, \ldots, m.$$

Let ℓ_i be the line $\{(x, y) \in \mathbb{R}^2 : a_{i1}x + a_{i2}y = b_i\}$. It is easy to calculate that the intersection of ℓ_i and ℓ_j, if it exists, has coordinates

$$\left(\frac{a_{i2}b_j - a_{j2}b_i}{a_{i2}a_{j1} - a_{i1}a_{j2}}, \frac{a_{j1}b_i - a_{i1}b_j}{a_{i2}a_{j1} - a_{i1}a_{j2}} \right).$$

If, for example, all a_{ij} and b_i are integers with absolute value at most 1000, then the coordinates of all intersections are fractions with numerators and denominators bounded by $2 \cdot 10^6$. Thus, if the solution set of the considered system of inequalities has at least one vertex, such a vertex has to lie in the square $[-2 \cdot 10^6, 2 \cdot 10^6]^2$. If the solution set has no vertex and it is nonempty, it can be shown that it has to contain one of the lines ℓ_i, and that each ℓ_i intersects the just mentioned square. Fact (E1) can be verified along these lines for the considered system with two variables. For a general system in dimension n the idea is the same, and Cramer's rule and a bound on the magnitude of the determinant of a matrix are used for estimating the coordinates of vertices of the solution set.

Fact (E2) requires more work, but the idea is similar. Each solution $\tilde{\mathbf{x}}$ of the original system $A\mathbf{x} \leq \mathbf{b}$ also satisfies the modified system $A\mathbf{x} \leq \mathbf{b} + \boldsymbol{\eta}$, and all $\mathbf{x}$ from the ball $B(\tilde{\mathbf{x}}, \varepsilon)$ satisfy it as well, because changing $\mathbf{x}$ by ε cannot change any coordinate of $A\mathbf{x}$ by more than η.

If $A\mathbf{x} \leq \mathbf{b}$ has no solution, then by a suitable variant of the Farkas lemma, namely, Lemma 6.6.1, there exists a nonnegative $\mathbf{y} \in \mathbb{R}^m$ such that $\mathbf{y}^T A = \mathbf{0}$ and $\mathbf{y}^T \mathbf{b} < 0$, and by normalizing $\mathbf{y}$ we may assume $\mathbf{y}^T \mathbf{b} = -1$. By Cramer's rule again it is shown that there also exists an $\mathbf{y}$ with not too large components, and such $\mathbf{y}$ then witnesses unsolvability for the system $A\mathbf{x} \leq \mathbf{b} + \boldsymbol{\eta}$ as well.

Here we finish the outline of the ellipsoid method. If some parts were too incomplete and hazy for the reader, we can only recommend a more extensive treatment, for instance in the excellent book

M. Grötschel, L. Lovász, L. Schrijver: *Geometric Algorithms and Combinatorial Optimization*, 2nd edition, Springer, Heidelberg 1994.

(We have taken the Sahara metaphor from there, among others.)

Why ellipsoids? They are used in the ellipsoid method since they constitute probably the simplest class of n-dimensional convex sets that is closed under non-singular affine maps. Popularly speaking, this class is rich enough to approximate all convex polyhedra including flat ones and needle-like ones. If desired, ellipsoids can be replaced by simplices, for example, but the formulas in the algorithm and its analysis become considerably more unpleasant than those for ellipsoids.

The ellipsoid method need not know all of the linear program. The system of inequalities $A\mathbf{x} \le \mathbf{b}$ can also be given by means of a **separation oracle**. This is an algorithm (black box) that accepts a point $\mathbf{s} \in \mathbb{R}^n$ as input, and if $\mathbf{s}$ is a solution of the system, it returns the answer YES, while if $\mathbf{s}$ is not a solution, it returns one (arbitrary) inequality of the system that is violated by $\mathbf{s}$. (Such an inequality separates $\mathbf{s}$ from the solution set, and hence the name separation oracle.) The ellipsoid method calls the separation oracle with the centers $\mathbf{s}_k$ of the generated ellipsoids, and it always uses the violated inequality returned by the oracle for determining the next ellipsoid.

We talk about this since a separation oracle can be implemented efficiently for some interesting optimization problems even when the full system has exponentially many inequalities or even infinitely many (so far we haven't considered infinite systems at all).

Probably the most important example of a situation where an infinite system of linear inequalities can be solved by the ellipsoid method is the **semidefinite programming.** In a semidefinite program we consider not an unknown *vector* $\mathbf{x}$, but rather an unknown square *matrix* $X = (x_{ij})_{i,j=1}^n$. We optimize a linear function of the variables x_{ij}. The constraints are linear inequalities and equations for the x_{ij}, *plus* the requirement that the matrix X has to be *positive semidefinite*. The last constraint distinguishes semidefinite programming from linear programming. It can be expressed by a system of infinitely many linear inequalities, namely, $\mathbf{a}^T X \mathbf{a} \ge 0$ for each $\mathbf{a} \in \mathbb{R}^n$. A separation oracle for this system can be constructed based on algorithms for computing eigenvalues and eigenvectors. The ellipsoid method can then approximate the optimum with a prescribed precision in polynomial time. (In reality, though, things are not quite as simple as it might seem from our mini-description.)

Numerous computational problems can be solved in polynomial time via semidefinite programming, some of them exactly and some approximately, and sometimes this yields the only known polynomial algorithm. A nice example of application of semidefinite programming is the **maximum cut problem** (MAX-CUT), in which the vertex set of a given graph $G = (V, E)$ should be divided in two parts so that the maximum possible number of edges go between the parts. Semidefinite programming is an essential component of an approximation algorithm for MAXCUT, called the Goemans–Williamson algorithm, that always computes a partition with the number of edges going between the parts at least 87,8% of the optimal number. This is the best known approximation guarantee, and most likely also the best possible one for any polynomial algorithm. More about this and related topics can be found, for instance, in the survey

> L. Lovász: Semidefinite programs and combinatorial optimization, in *Recent Advances in Algorithms and Combinatorics* (B. Reed and C. Linhares-Sales, editors), pp. 137–194, Springer, New York, 2003.

Theory versus practice. The notion of polynomial algorithm was suggested in the 1970s by Jack Edmonds as a formalized counterpart of the intuitive notion of an efficient algorithm. Today, a theoretician's first question for every algorithm is: Is it polynomial?

How is it possible that the ellipsoid method, which is polynomial, is much slower

in practice than the simplex method, which is not polynomial? One of the reasons is that even though the ellipsoid method is polynomial, the degree of the polynomial is quite high. The second and main reason is that the simplex method is slow only on specially constructed linear programs, which it almost never encounters in practice, while the ellipsoid method seldom behaves better than in the worst case. But the "good behavior on all inputs with rare exceptions" of the simplex method seems hard to capture theoretically. Moreover, a guaranteed efficiency for *all* inputs is much more satisfactory than only an empirically supported belief that an algorithm is usually fast.

The notion of polynomial algorithm thus has great shortcomings from a practical point of view. But attempts at constructing a polynomial algorithm in theory usually also leads, over time, to practically efficient algorithms. An impressive example in the area of linear programming are interior point methods.

7.2 Interior point methods

The next time linear programming made it to press headlines was in 1984. Narendra Karmakar, a researcher at IBM, suggested an algorithm that is named after him and belongs to the large family of interior point methods. He proved its polynomiality and published results of computational experiments suggesting that in practice it is much faster than the simplex method. Although his statements in the latter direction turned out to be somewhat exaggerated, interior point methods are commonly used in linear programming and often they beat the simplex method.

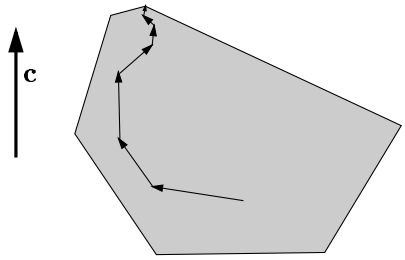
Interior point methods have many variants and they have been used at least since the 1950s for nonlinear optimization problems. In linear programming they are classified, according to the basic approach, as *central trajectory methods* (or *central path methods*), *potential reduction methods*, and *affine scaling methods*, and almost for every approach there are a *primal version*, a *dual version*, and sometimes also a *self-dual version*.

Different expositions of the same algorithm are often based on different intuition. For example, what is presented as a step of Newton's iteration method in one source, may be derived by linear approximation from the method of Lagrange multipliers in another source, and may be explained as a projection of the gradient in the set of feasible solutions. We say this so that a reader who finds in the literature something seemingly entirely different from what we say below is not confused more than necessary.

We have explained the simplex method with proofs. We have at least formulated the algorithm of the ellipsoid method and some exact statements about it. But for interior point methods we will try only to evoke a basic impression of how they work. It is mainly because these methods are complicated and they use mathematical tools that do not fit in an introductory textbook. Modern linear programming software is based on a remarkably advanced mathematics; even systems of linear equations are not solved by the good old Gauss elimination, but by a combination of several complex methods.

The simplex method crawls along the boundary of the set of feasible solutions. The ellipsoid method encircles the set of feasible solutions, and up until the last step it remains outside of it. Interior point methods walk through the

interior of the set of feasible solutions towards an optimum, carefully avoiding the boundary. Only at the very end, when they get almost to an optimum, they jump to an optimal vertex. See a schematic picture:



We indicate how the **central trajectory method**, which is currently considered the most successful, works. We consider the linear program

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\mathbf{x} \leq \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0}.$$

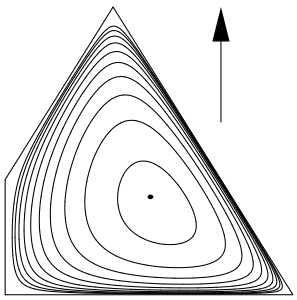
From a given interior point of the set P of feasible solutions we could try to go straight in the direction $\mathbf{c}$, but that way we would soon hit the boundary. So we have to change the direction soon enough so that the trajectory avoids the boundary but still tends to an optimum.

In order to compute a trajectory to an optimum through the interior, we consider an auxiliary family of optimization problems, parameterized by a real number $\mu \geq 0$. Each problem in this family has the same constraints as the linear program. The objective function is defined as follows:

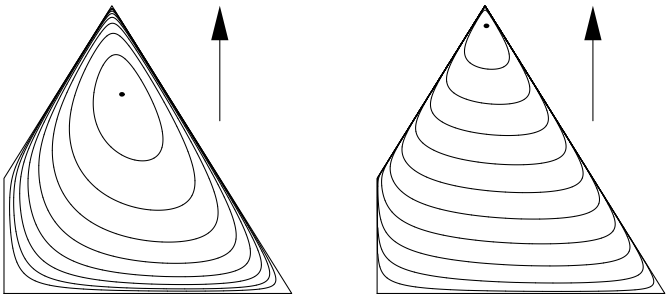
$$f_\mu(\mathbf{x}) = \mathbf{c}^T\mathbf{x} + \mu\Big(\sum_{j=1}^{n} \ln(x_j) + \sum_{i=1}^{m} \ln\left(b_i - \mathbf{a}_i^T\mathbf{x}\right)\Big),$$

where $\mathbf{a}_i$ is the ith row of the matrix A. The objective function is nonlinear since it contains logarithms. It is constructed in such a way that if $\mathbf{x}$ approaches the boundary of P, i.e., if some component approaches 0 or the difference of the right-hand side and left-hand side of some inequality approaches 0, then the objective function tends to $-\infty$. The expression in parentheses following μ in the definition of f_μ is called a *barrier function*, or more definitely, a *logarithmic barrier*.

For P bounded and with nonempty interior it can be shown that for every $\mu > 0$ the problem has a unique optimal solution, which we denote by $\mathbf{x}^*_\mu$. If μ is a very large number, the influence of the term $\mathbf{c}^T\mathbf{x}$ in the objective function is negligible and $\mathbf{x}^*_\mu$ is a point "farthest from the boundary," called the *analytic center* of P. The following picture shows (for a two-dimensional problem) contours of the function f_μ for $\mu = 100$ (the vector $\mathbf{c}$ is depicted by an arrow and the point $\mathbf{x}^*_\mu$ by a dot):



On the other hand, for small μ the point $\mathbf{x}^*_\mu$ is close to an optimum of the linear program; see illustrations for $\mu = 0.5$ and $\mu = 0.1$:

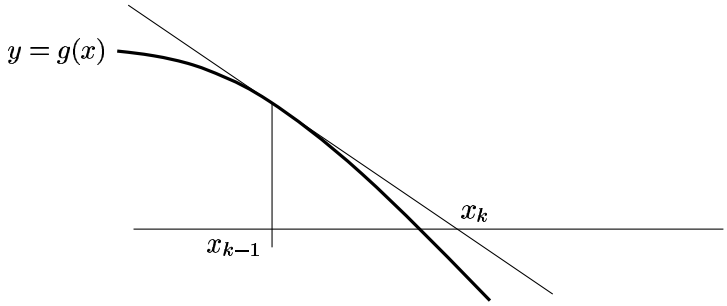


The set $\{\mathbf{x}^*_\mu : \mu > 0\}$ is called the **(primal) central trajectory**, and the algorithm tries to follow it and thus reach an optimum.

The points $\mathbf{x}^*_\mu$ are not computed exactly; it would be too difficult and it is not even needed. We choose a suitable decreasing sequence $\mu_1 > \mu_2 > \cdots$ tending to 0. Having an interior point $\mathbf{x}_{k-1} \in P$ from the $(k{-}1)$st iteration, in the kth iteration we want to make a step towards the point $\mathbf{x}^*_{\mu_k}$. This means that we want to go to the maximum of $f_{\mu_k}(\mathbf{x})$. This function is continuously differentiable within P, and the maximum is where the gradient of f_{μ_k} equals 0 (in other words, all the first partial derivatives are 0 there). Hence $\mathbf{x}^*_{\mu_k}$ is the (unique) zero of the vector function

$$F(\mathbf{x}) = \left(\frac{\partial f_{\mu_k}(\mathbf{x})}{\partial x_1}, \frac{\partial f_{\mu_k}(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial f_{\mu_k}(\mathbf{x})}{\partial x_n} \right)^T .$$

One of the algorithms for computing zeros of differentiable functions is **Newton's method**. One step of it for a function $g\colon \mathbb{R} \to \mathbb{R}$ of a single variable is shown in a picture:



We approximate g by the tangent at the point x_{k-1}, and we determine the next point x_k as the intersection of this tangent with the x-axis. Under suitable assumptions on g and on the initial point x_0, the sequence $(x_0, x_1, \ldots)$ converges to a zero of g very rapidly. Newton's method can also be generalized to a function $\mathbb{R}^n \to \mathbb{R}^n$, whose graph is approximated by a tangent hyperplane in each step.

In the method of central trajectory the point $\mathbf{x}_k$ is obtained from $\mathbf{x}_{k-1}$ by one step of Newton's method for the function F introduced above. But we do not really make the whole step; rather, we go only 60% of the way, say, in direction to the point where Newton's method would normally go. It turns out that this point can be found as a solution of a certain system of linear equations. A large part of the theory of interior point methods is concerned with solving this system and similar ones efficiently, because this is the computationally most demanding part.

The currently fastest variants do not even use Newton's method, but higher-order methods that approximate the function F by an algebraic surface of higher degree, computed from higher derivatives of F.

The way we have described the method of central trajectory so far, it would need an initial interior point of the set of feasible solutions. Such a point could be found by a two-phase procedure, similar to finding an initial feasible basis in the simplex method. But more often a *primal-dual version* is employed, where we work in a higher-dimensional space, whose coordinates are variables of the original primal linear program as well as variables of the dual linear program and additional auxiliary variables. The initial point then need not correspond to a feasible solution either for the primal or for the dual linear program. A feasible solution is reached at the very end, together with reaching an optimum. But we will not go into further technical details and we conclude the sketch of interior-point methods here.

Several interior-point methods for linear programming are known to be (weakly) polynomial algorithms. The total number of bit operations for the best of these algorithms is bounded by $O(n^3L)$, where L is the maximum bit size of coefficients in the linear program and n is the number of variables (for equational form). The maximum number of iterations before reaching an optimum is $O(L\sqrt{n})$. On the other hand, examples are known (based on the Klee–Minty cube, again!) where any any algorithm based on following the central path must make $\Omega(\sqrt{n}\log n)$ iterations; see

> A. Deza, E . Nematollahi, and T. Terlaky: How good are interior point methods? Klee–Minty cubes tighten iteration-complexity bounds, Technical Report, McMaster University, 2004.

In practice the number of iterations seems to be bounded by a constant or by $O(\log n)$ in almost all cases. This is somewhat similar to the situation for the simplex method, where the worst-case behavior is much worse than the behavior for typical inputs.

Interior point methods have been used with success not only for linear programming, but also for semidefinite programming and other important classes of optimization problems, such as convex quadratic programming. The reader can find more about this, e.g., in the survey paper

> F. A. Potra, S. J. Wright: Interior-point methods, *Journal of Computational and Applied Mathematics* 124(2000), pp. 281–302.

(at time of writing this text it was accessible at web pages of the authors), or in several books cited there.

8

More Applications

Here we have collected several applications of linear programming, and in particular, of the Duality Theorem. They are slightly more advanced than those in Chapters 2 and 3, but we have tried to keep everything very concrete and on as elementary level as possible, and we hope that even a mathematically unexperienced reader will have no problems enjoying these small gems.

8.1 Zero-Sum Games

The Colonel Blotto game. Colonel Blotto and his opponent are preparing for a battle over three mountain passes. Each of them has five regiments at disposal. The one who sends more regiments to a pass occupies it, but when the same number of regiments meet, there will be a draw. Finally, the one who occupies more passes than the other wins the battle, with a draw occurring if both occupy the same number of passes. Given that all three passes have very similar characteristics, the strategies independently pursued by both Colonel Blotto and his opponent are the following: First they partition their five regiments into three groups. For example, the partition $(0, 1, 4)$ means that one pass will be attacked by 4 regiments, another pass by 1 regiment, and one pass will not be attacked at all. Then, the groups are assigned to the passes randomly; that is, each of the $3! = 6$ possible assignments of groups to passes is equally likely.

The partitions of Colonel Blotto and his opponent therefore determine *winning probabilities* for both of them (in general, these do not add up to one because of possible draws). Both Colonel Blotto and his opponent want to bias the difference of these probabilities into their direction as much as possible. How should they choose their partitions?

This is an instance of a finite two player **zero-sum game**. In such a game, each of the two players has a finite set of possible strategies (in our case, the partitions), and each pair of opposing strategies leads to a *payoff* known to both players. In our case, we define the payoff as Colonel Blotto's winning probability minus the opponent's winning probability. Whatever one of the players wins, the other player loses, and this explains the term zero-sum game. To some extent, it has become a part of common vocabulary.

When we number the strategies $1, 2, \ldots, n$ for the first player and $1, 2, \ldots, m$

for the second player, the payoffs can be recorded in the form of an $n \times m$ *payoff matrix*. In the Colonel Blotto game, the payoff matrix looks as follows, with the rows corresponding to the strategies of Colonel Blotto and the columns to the strategies of the opponent.

	$(0,0,5)$	$(0,1,4)$	$(0,2,3)$	$(1,1,3)$	$(1,2,2)$
$(0,0,5)$	0	$-\frac{1}{3}$	$-\frac{1}{3}$	-1	-1
$(0,1,4)$	$\frac{1}{3}$	0	0	$-\frac{1}{3}$	$-\frac{2}{3}$
$(0,2,3)$	$\frac{1}{3}$	0	0	0	$\frac{1}{3}$
$(1,1,3)$	1	$\frac{1}{3}$	0	0	$-\frac{1}{3}$
$(1,2,2)$	1	$\frac{2}{3}$	$-\frac{1}{3}$	$\frac{1}{3}$	0

For example, when Colonel Blotto chooses $(0,1,4)$ and his opponent chooses $(0,0,5)$, then Colonel Blotto wins (actually, without fighting) if and only if his two nonempty groups arrive at the two passes left unattended by his opponent. The probability for this to happen is $\frac{1}{3}$. With probability $\frac{2}{3}$, there will be a draw, so the difference of the winning probabilities is $\frac{1}{3} - 0 = \frac{1}{3}$.

Not knowing the opponent's behavior, Colonel Blotto might want to choose a strategy that guarantees the *highest* payoff in the *worst case*. The only candidate for such a strategy is $(0,2,3)$: No matter what the opponent does, Colonel Blotto will get a payoff of at least 0 with this strategy, while all other strategies lead to negative payoff in the worst case. (Anticipating that a spy of the opponent might find out about his plans, he must reckon that the worst case will actually happen. The whole game is not a particularly cheerful matter anyway.)

Similarly, the opponent wants to choose a strategy that guarantees the *lowest* payoff (for Colonel Blotto) in the worst case. It turns out that $(0,2,3)$ is also the unique such choice for the opponent, because it guarantees that Colonel Blotto will receive payoff at most 0, while all other strategies allow him to achieve positive payoff, if he happens to guess or spy out the opponent's strategy.

Here comes the interesting feature of this example: The strategy selected by the opponent is actually among the ones anticipated by the pessimistic Colonel Blotto, and vice versa. In other words, both see their worst expectations come true, exactly those on which they pessimistically based their choice of strategy. We say that the two strategies are *best responses* against each other. This equivalently means that the payoff value associated with the pair $((0,2,3),(0,2,3))$ is a "saddle point" of the payoff matrix: It is a *minimum* in its row and a *maximum* in its column.

Seeing the worst case happen might shatter hopes for a better outcome of the battle, but on the other hand, it is a relief. After the battle has been fought, neither Colonel Blotto nor his opponent will have to regret their choice. Because they already had best responses against each other, both know that they could not have improved their situation by switching to another strategy, given the other one sticks to his strategy. Even if both had found out through spies about the other one's plan in advance, neither of them would have had

an incentive to change his own plan. In this sense, strategies that are best responses against each other form a pair of *stable* strategies, and we say that such a pair is a **Nash equilibrium** of the game.

The Rock-Paper-Scissors game. Alice and Bob independently choose a hand gesture indicating either a rock, or a piece of paper, or a pair of scissors. If both players indicate the same item, the game is a draw, and otherwise, there is a cyclic pattern: Scissors beats paper (by cutting it), paper beats rock (by wrapping it up), rock beats scissors (by making it blunt). Assuming that the payoff to Alice is 1 if she wins, 0 if there is a draw, and -1 if she loses, the payoff matrix is

	rock	paper	scissors
rock	0	-1	1
paper	1	0	-1
scissors	-1	1	0

.

In this example the payoff matrix has no saddle point, and consequently, there is no Nash equilibrium according to the above definition. The loser will regret not to have chosen the item that beats the one indicated by the winner, so there is no pair of stable strategies. But when we generalize the notion of a strategy, there *is* a way for both players to avoid regret: Both should decide *randomly*, selecting each of the items with probability 1/3. Even this strategy may lose, of course, but still there is no reason for regret, since with the same probability 1/3, it could have won, and the fact that it didn't is not a fault of the strategy but just bad luck. Using these *mixed* strategies, both Alice and Bob can guarantee *expected payoff* 0, and none of them can do better by unilaterally switching to a different behavior. We say that we have a *mixed Nash equilibrium*.

The surprising fact is that *every* zero-sum game has a mixed Nash equilibrium! The corresponding mixed strategy of each players is defined by a probability distribution over the player's original set of *pure* strategies. These distributions might not be uniform as in the very simple Rock-Paper-Scissors game, but they exist and can be computed through linear programming. In the Colonel Blotto game, we even found a *pure* Nash equilibrium: a mixed one in which each player chooses some pure strategy with probability 1.

We might also consider this latter pure Nash equilibrium as being properly mixed, since the involved strategies take random decisions. But we regard these decisions as "hardwired" into the strategies and the payoff matrix. Alternatively, we could consider each assignment of regiments to passes as a pure strategy, and then the strategies considered at the beginning of the section would be mixed. These are two different views (models) of the game, and we are free to investigate either one, although one may be more convenient or more realistic than the other.

Existence and computation of a mixed Nash equilibrium. Let us repeat the setup of zero-sum games in a more formal manner. We have two players, and we stick to calling them Alice and Bob. Alice has a set of $n \geq 1$ pure strategies at her disposal, while Bob has a set of $m \geq 1$ pure strategies.

Then there is an $n \times m$ payoff matrix M of real numbers such that m_{ij} is

Alice's gain (and Bob's loss) when Alice's i-th strategy is played against Bob's j-th strategy. For concreteness, we may think of Bob having to pay $\in m_{ij}$ to Alice. Of course, the situation is symmetric in that m_{ij} might be negative, in which case Alice has to pay $\in -m_{ij}$ to Bob.

A **mixed strategy** of a player is a probability distribution over his or her set of pure strategies. We encode a mixed strategy of Alice by an n-dimensional vector of probabilities

$$\mathbf{x} = (x_1, \ldots, x_n), \quad \sum_{i=1}^{n} x_i = 1, \ \mathbf{x} \geq \mathbf{0},$$

and a mixed strategy of Bob by an m-dimensional vector of probabilities

$$\mathbf{y} = (y_1, \ldots, y_m), \quad \sum_{j=1}^{m} y_j = 1, \ \mathbf{y} \geq \mathbf{0}.$$

Given mixed strategies $\mathbf{x}$ and $\mathbf{y}$ of Alice and Bob, the *expected payoff* (expected gain of Alice) when $\mathbf{x}$ is played against $\mathbf{y}$ is

$$\begin{aligned} &\sum_{i,j} m_{ij} \operatorname*{Prob}_{\mathbf{x},\mathbf{y}}[\text{Alice plays } i, \text{Bob plays } j] \\ &= \textstyle\sum_{i,j} m_{ij} \operatorname{Prob}_{\mathbf{x}}[\text{Alice plays } i] \cdot \operatorname{Prob}_{\mathbf{y}}[\text{Bob plays } j] \\ &= \textstyle\sum_{i,j} m_{ij} x_i y_j \\ &= \mathbf{x}^T M \mathbf{y}. \end{aligned}$$

Just like in the Colonel Blotto game, let us see what happens if Alice tries to optimize her behavior under the pessimistic assumption that Bob can guess what she is up to. Alice figures that if she plays some fixed strategy $\mathbf{x}$, her expected gain may in the worst case be no larger than

$$\min_{\mathbf{y}} \mathbf{x}^T M \mathbf{y}, \tag{8.1}$$

namely if Bob happens to play the best response $\mathbf{y}$ against $\mathbf{x}$. To maximize the expected gain Alice can secure, regardless of Bob's response, she should therefore play a strategy $\tilde{\mathbf{x}}$ that *maximizes* the minimum in (8.1).

We note that all minima and maxima exist, because they are taken over compact sets—the probability distributions of the players.

Bob, with the same reasoning, finds that in playing a fixed strategy $\mathbf{y}$, he might lose up to

$$\max_{\mathbf{x}} \mathbf{x}^T M \mathbf{y} \tag{8.2}$$

in expectation, namely if Alice plays the best response $\mathbf{x}$ to $\mathbf{y}$. To minimize the expected loss Bob may incur, he should play a strategy $\tilde{\mathbf{y}}$ that *minimizes* the maximum in (8.2).

With $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$ as just described, we get by definition

$$\min_{\mathbf{y}} \tilde{\mathbf{x}}^T M \mathbf{y} \leq \tilde{\mathbf{x}}^T M \tilde{\mathbf{y}} \leq \max_{\mathbf{x}} \mathbf{x}^T M \tilde{\mathbf{y}}. \tag{8.3}$$

Therefore, when Alice plays $\tilde{\mathbf{x}}$ and Bob plays $\tilde{\mathbf{y}}$, Alice gains at least what she calculated, while Bob loses at most what he calculated (in expectation). But can it really happen that Alice, say, gains strictly *more* than what she calculated? No! The next theorem states that the inequalities in (8.3) are actually equalities, so both players gain or lose exactly what they calculated.

8.1.1 Theorem. *For any $n \times m$ payoff matrix M,*

$$\max_{\mathbf{x}} \min_{\mathbf{y}} \mathbf{x}^T M \mathbf{y} = \min_{\mathbf{y}} \max_{\mathbf{x}} \mathbf{x}^T M \mathbf{y}.$$

Moreover, there exist mixed strategies $\tilde{\mathbf{x}}, \tilde{\mathbf{y}}$ such that

$$\max_{\mathbf{x}} \min_{\mathbf{y}} \mathbf{x}^T M \mathbf{y} = \min_{\mathbf{y}} \tilde{\mathbf{x}}^T M \mathbf{y} = \tilde{\mathbf{x}}^T M \tilde{\mathbf{y}} = \max_{\mathbf{x}} \mathbf{x}^T M \tilde{\mathbf{y}} = \min_{\mathbf{y}} \max_{\mathbf{x}} \mathbf{x}^T M \mathbf{y},$$

and $\tilde{\mathbf{x}}, \tilde{\mathbf{y}}$ can be computed through linear programming.

The number $\tilde{\mathbf{x}}^T M \tilde{\mathbf{y}}$ is called the **game value**, and the pair of mixed strategies $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ is said to form a *Nash equilibrium* of the game. By construction, a mixed Nash equilibrium is stable: Even if Alice knew Bob's mixed strategy $\tilde{\mathbf{y}}$ in advance, she would not be able to do better than choosing $\tilde{\mathbf{x}}$ (by the equality $\tilde{\mathbf{x}}^T M \tilde{\mathbf{y}} = \max_{\mathbf{x}} \mathbf{x}^T M \tilde{\mathbf{y}}$). Bob is in the same situation. Thus $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$ are *best responses* with respect to each other, and even in retrospect, the players will not regret their choices.

Proof. The only possible candidates for $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$ are those considered earlier; that is, $\tilde{\mathbf{x}}$ maximizing $\min_{\mathbf{y}} \mathbf{x}^T M \mathbf{y}$ and $\tilde{\mathbf{y}}$ minimizing $\max_{\mathbf{x}} \mathbf{x}^T M \mathbf{y}$. It remains to show how they can be computed, and that the desired equalities hold.

Bob's best response to a *fixed* mixed strategy $\mathbf{x}$ of Alice can be found by solving a linear program in the variables $y_1, \ldots, y_m$:

$$\begin{array}{ll} \text{minimize} & \mathbf{x}^T M \mathbf{y} \\ \text{subject to} & \sum_{i=1}^m y_i = 1, \\ & \mathbf{y} \geq \mathbf{0}. \end{array} \tag{8.4}$$

Let $\beta(\mathbf{x})$ denote the optimum value of this linear program. Now consider Alice's task of finding a mixed strategy $\mathbf{x}$ for which $\beta(\mathbf{x})$—the expression (8.1)— is maximized. This is not a linear program, because $\beta(\mathbf{x})$ is not a linear function, but the solution to another linear program. Luckily, we can circumvent this issue by using linear programming duality. Let $\mathbf{1} = (1, \ldots, 1)$. The dual of (8.4) also has optimum value $\beta(\mathbf{x})$, and it can be written as

$$\begin{array}{ll} \text{maximize} & x_0 \\ \text{subject to} & M^T \mathbf{x} - \mathbf{1} x_0 \geq \mathbf{0}, \end{array}$$

which has just one variable x_0. This can be seen through the dualization recipe from Section 6.2. Let us just recall the pattern here: We have one equality constraint in the primal problem, for which we get one unrestricted variable in the dual. For each of the m restricted variables in the primal, we have one inequality constraint in the dual. Before applying the recipe, we need to convert

the primal problem into a maximization problem; after a little rewriting, the resulting dual problem then assumes the form given here.

Note that the dual problem has only one variable x_0, while $x_1, \ldots, x_n$ are regarded as fixed numbers in it. However, since we want to maximize $\beta(\mathbf{x})$ over all mixed strategies $\mathbf{x}$ of Alice, we set up a new linear program that optically looks exactly like the previous one, but in which $x_1, \ldots, x_n$ are regarded as variables (this works only because the constraints happen to be linear in $x_0, x_1, \ldots, x_n$):

$$\begin{array}{ll} \text{maximize} & x_0 \\ \text{subject to} & M^T\mathbf{x} - \mathbf{1}x_0 \geq \mathbf{0}, \\ & \sum_{i=1}^n x_i = 1, \\ & \mathbf{x} \geq \mathbf{0}. \end{array} \tag{8.5}$$

If $(\tilde{x}_0, \tilde{\mathbf{x}})$ denotes an optimal solution to this problem, we have by construction

$$\tilde{x}_0 = \underbrace{\min_{\mathbf{y}} \tilde{\mathbf{x}}^T M \mathbf{y}}_{\beta(\tilde{\mathbf{x}})} = \underbrace{\max_{\mathbf{x}} \min_{\mathbf{y}} \mathbf{x}^T M \mathbf{y}}_{\max_{\mathbf{x}} \beta(\mathbf{x})}. \tag{8.6}$$

In a symmetric fashion, we can derive a linear program for solving Bob's task of computing a best strategy $\tilde{\mathbf{y}}$. We obtain the problem

$$\begin{array}{ll} \text{minimize} & y_0 \\ \text{subject to} & M\mathbf{y} - \mathbf{1}y_0 \leq \mathbf{0}, \\ & \sum_{i=1}^m y_i = 1, \\ & \mathbf{y} \geq \mathbf{0} \end{array} \tag{8.7}$$

in the variables $y_0, y_1, \ldots, y_m$. If we use $\alpha(\mathbf{y})$ as a shorthand for the expression (8.2), we get that an optimal solution $(\tilde{y}_0, \tilde{\mathbf{y}})$ satisfies

$$\tilde{y}_0 = \underbrace{\max_{\mathbf{x}} \mathbf{x}^T M \tilde{\mathbf{y}}}_{\alpha(\tilde{\mathbf{y}})} = \underbrace{\min_{\mathbf{y}} \max_{\mathbf{x}} \mathbf{x}^T M \mathbf{y}}_{\min_{\mathbf{y}} \alpha(\mathbf{y})}. \tag{8.8}$$

The punchline is that the two linear programs (8.5) and (8.7) are dual to each other! Again, our dualization recipe shows this. It follows that both programs have the same optimum value $\tilde{x}_0 = \tilde{y}_0$. Putting together (8.6), (8.8), as well as our previously derived inequality (8.3) proves the theorem. □

Rock-Paper-Scissors revisited. To kill the time between their rare public appearances, Santa Claus and the Easter Bunny play the rock-paper-scissors game against each other. The Easter Bunny, however, is unable to indicate a pair of scissors with his paw and is therefore limited to two pure strategies. The payoff matrix in this variant is

	rock	paper
rock	0	−1
paper	1	0
scissors	−1	1

.

We already see that Santa Claus should never play rock: for any possible gesture of the Easter Bunny, paper is a better strategy.

Let us apply the machinery we have just developed to find optimal mixed strategies for Santa Claus and the Easter Bunny. Recall that Santa Claus has to solve the linear program (8.5) to find the probability distribution $\tilde{\mathbf{x}} = (\tilde{x}_1, \tilde{x}_2, \tilde{x}_3)$ that determines his optimal strategy. At the same time, he will compute the game value $\tilde{x}_0$, his expected gain.

The linear program is

$$\begin{array}{lrrrrrrrrrl} \text{maximize} & x_0 & & & & & & & & & \\ \text{subject to} & & & & x_2 & - & x_3 & - & x_0 & \geq & 0 \\ & - & x_1 & & & + & x_3 & - & x_0 & \geq & 0 \\ & & x_1 & + & x_2 & + & x_3 & & & = & 1 \\ & & & & & & & x_1, x_2, x_3 & & \geq & 0 \end{array}$$

A (unique) optimal solution is $(\tilde{x}_0, \tilde{x}_1, \tilde{x}_2, \tilde{x}_3) = (1/3, 0, 2/3, 1/3)$.

The Easter Bunny's problem (8.7) is

$$\begin{array}{lrrrrrrrrl} \text{minimize} & y_0 & & & & & & & & \\ \text{subject to} & & & - & y_2 & - & y_0 & \leq & 0 \\ & & y_1 & & & - & y_0 & \leq & 0 \\ & - & y_1 & + & y_2 & - & y_0 & \leq & 0 \\ & & y_1 & + & y_2 & & & = & 1 \\ & & & & & & y_1, y_2 & \geq & 0 \end{array}$$

A (unique) optimal solution is $(\tilde{y}_0, \tilde{y}_1, \tilde{y}_2) = (1/3, 1/3, 2/3)$.

To verify that the primal and dual solutions that we claim are optimal, you don't need to solve any linear programs: The complementary slackness Theorem 6.1.2 gives a simple way of checking this.

Let us summarize: If both play optimally, Santa Claus wins 1/3 on average (see a scientific explanation of why Santa Claus can afford to bring more presents!). Both play paper with probability 2/3. With the remaining probability 1/3, Santa Claus plays scissors, while the Easter Bunny plays rock. This result is a simple but still nontrivial application of zero-sum game theory. In retrospect, the original rock-paper-scissors game might appear rather boring, but this is relative: There is a *World RPS Society* (`http://www.worldrps.com/`) that holds an annual rock-paper-scissors world championship and sells a book on how to play the game.

Choosing numbers. Here is another game, which seems fun to actually play and where the optimal mixed strategies are not at all obvious. Each of the two players independently writes down a number between 1 and 6. Then the numbers are compared. If they are equal, the game is a draw. If the numbers differ by one, the player with the smaller number gets €2 from the one with the larger number. If the two numbers differ by two or more, the one with the larger number gets €1. We want to challenge the reader to compute the optimal mixed strategies for this game (by symmetry, they are the same for both players).

The Colonel Blotto game has apparently been first considered by the mathematician Emile Borel in 1921 (he also served as a French minister of the navy,

although only for several months). It can be considered for any number of regiments; for 6 regiments, there is still a Nash equilibrium defined by a pair of pure strategies, but for 7 regiments (this is the case stated in Borel's original paper) it becomes necessary to mix. Borel's paper does not mention the name "Colonel Blotto"; this name appears in Hubert Phillips' *Week-end Problems Book*, a collection of puzzles from 1933.

In his paper, Borel considers *symmetric* games in general. A symmetric game is defined by a payoff matrix M with $M^T = -M$ and $m_{ii} = 0$ for all i. Borel erroneously states that for large enough $n = m$, one can construct symmetric games in which each player can secure a positive expected payoff, knowing the other player's mixed strategy. He concludes that it needs psychology on top of mathematics to play zero-sum games.

Only in 1926, John von Neumann (not knowing about Borel's work and its pessimistic conclusion) formally established Theorem 8.1.1.

Bimatrix games. An important generalization of finite zero-sum games are *bimatrix games* in which both Alice and Bob want to maximize the payoff with respect to a payoff matrix of their own, A for Alice, and B for Bob (in the zero-sum case, $A = -B$). A bimatrix game also has at least one mixed Nash equilibrium, but finding it requires techniques beyond linear programming. Moreover, Nash equilibria in bimatrix games are in general not as satisfactory as in zero-sum games, and there is no such thing as "the" game value. We know that in a Nash equilibrium, no player has an incentive to *unilaterally* switch to a different behavior. Yet, it may happen that *both* can increase their payoff by switching simultaneously, a situation that obviously cannot occur in a zero-sum game. This means that the Nash equilibrium was not optimal from the point of view of *social welfare*, and no player has a real desire of being in this particular Nash equilibrium. It may even happen that all Nash equilibria are of this suboptimal nature. Here is an example.

At each of the two department stores in town, *All The Best Deals* and *Buyer's Paradise*, the respective owner needs to decide whether to launch an advertisement campaign for the upcoming Christmas sale. If one store runs a campaign while the competitor doesn't, the expected extra revenue obtained from customers switching their preferred store (€500,000, say) and new customers (€100,000) easily outweighs the cost of the campaign (€200,000). If, on the other hand, both stores advertise, let us assume that the campaigns more or less neutralize themselves, with extra revenue coming only from new customers in an almost saturated market (€80,000 for each of the stores).

Listing the total revenues a_{ij}, b_{ij} as pairs, in units of 1,000€, we obtain the following matrix, with rows corresponding to the strategies of *All The Best Deals*, and columns to *Buyer's Paradise*.

	advertise	don't advertise
advertise	$(-120,-120)$	$(400,-500)$
don't advertise	$(-500,400)$	$(0,0)$

If the store owners were friends, they would agree on running no campaign in order to save money. But in real life, the store owners do not communicate, and if they do, they mistrust each other. In this situation, rational behavior will force both stores to waste money on campaigns. To see this, put yourself in the position of one of the store owners. Assuming your competitor will not advertise, you definitely want to advertise in order to profit from the extra revenue. Assuming the other store will advertise, you must advertise as well, in order not to lose customers. This means that you will advertise in any case. We say that the strategy "advertise" *strictly dominates* the strategy "don't advertise", so it would be irrational to play the latter.

Because the other store owner reaches the same conclusion, there will be two almost useless campaigns in the end. In fact, the pair of strategies (advertise, advertise) is the unique Nash equilibrium of the game (mixing does not help), but it is suboptimal with respect to social welfare.

In general bimatrix games, the players might not be able to reach the social optimum through rational reasoning, even if this optimum corresponds to an equilibrium of the game. This is probably the most serious deficiency of bimatrix games as models of real-world situations. An example is the *battle of the sexes*. A couple wants to go out at night. He prefers the boxing match, while she prefers the opera, but both prefer going together over going alone. If both are to decide independently where to go, there is no rational way of reaching the social optimum (namely both going out together, no matter where).

When the advertisement game is repeatedly played (after all, there is a Christmas sale every year), the situation changes. In the long run, wasting money every year is such a bad prospect that the following more cooperative behavior makes sense: In the first year, refrain from advertising, and in later years just do what the competitor did the year before. This strategy is known as TIT FOR TAT. If both stores adopt this policy, they will never waste money on campaigns; but even if one store deviates from it, there is no possibility of exploiting the competitor in the long run. It is easy to see that after a possible first loss, the one playing TIT FOR TAT can "pay" for any further loss, due to a previous loss of the competitor.

The Prisoner's Dilemma. The advertisement game is a variation of the well-known *prisoner's dilemma*, in which two convicts charged with a crime committed together are independently offered (somewhat unethical) plea bargains; if both stay silent, missing evidence will only lead to minor punishment. If both tell on the other, there will be some bearable punishment. But if—and this is the unethical part—exactly one of the two tells on the other, the betrayer will be rewarded and goes free, while the one that remains silent receives a heavy penalty. As before, rational behavior will force both convicts to tell on each other, even if they have nothing to do with the crime. The prosecuting attorney can be sure to win the case.

A popular introduction with many interesting connections is

> W. Poundstone: *Prisoner's Dilemma: John von Neumann, Game Theory, and the Puzzle of the Bomb*, Doubleday, New York 1992

(and the 1964 Stanley Kubrick movie *Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb* might also widen one's horizons in this context).

A general introduction to game theory is

> J. Bewersdorff: *Luck, Logic, and White Lies*, AK Peters, Wellesley 2004.

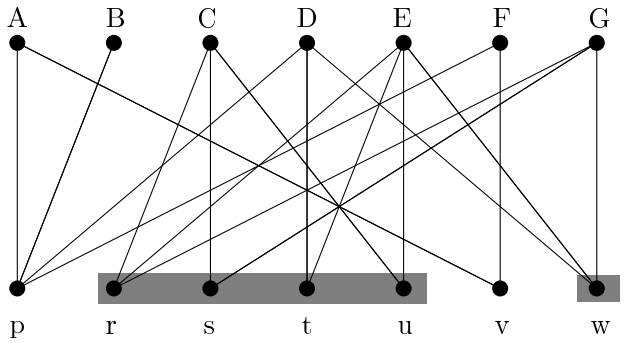
This book also contains the references to the work by Borel and von Neumann that we have mentioned above.

8.2 Matchings and vertex covers in bipartite graphs

Let us come back to the job assignment problem from Section 3.2. There, the human resources manager of a company was confronted with the problem of filling seven positions with seven employees, where every employee has a score (reflecting qualification) for each position he or she is willing to accept. We found that the manager can use linear programming to find an assignment of employees to positions (a perfect matching) that maximizes the sum of scores. We have also promised to show how the manager can fill the positions optimally if there are *more* employees than positions.

Here we will first disregard the scores and ask, under what conditions does the task have any solution at all. That is, whether there is any assignment of positions to employees such that every employee gets a position she or he is willing to accept and each position is filled. We know we can decide this by linear programming (we just invent some arbitrary scores, say all scores 100), but here we ask for a mathematical condition.

For example, let us consider the graph from Section 3.2 but slightly modified (some of the people changed their mind):



There is *no* assignment in this situation. This is not immediately obvious, but it becomes obvious once we look at the set $\{r, s, t, u, w\}$ of jobs (marked). Indeed, the set of people willing to take *any* of these 5 jobs is $\{C, D, E, G\}$. This is only 4 people, and so they cannot be assigned to 5 different jobs.

The next theorem, known as *Hall's Theorem* or the *Marriage Theorem*, states that if no assignment exists, we can *always* find such a simple "reason": a subset of k jobs such that the total number of employees willing to take any of them is smaller than k.

Before we formally state and prove this theorem, in the language of bipartite graphs, we need to recall the notions of maximum matching (Section 3.2) and minimum vertex cover (Section 3.3).

A matching in a graph $G = (V, E)$ is a set $E' \subseteq E$ of edges with the property that each vertex is incident to (or covered by) at most one edge in E'. A matching is maximum if it has the largest number of edges among all matchings in G.

A vertex cover of $G = (V, E)$ is a set $V' \subseteq V$ of vertices with the property that each edge is incident to at least one vertex in V'. A vertex cover is minimum if it has the smallest number of edges among all vertex covers of G.

Hall's Theorem gives necessary and sufficient conditions for the existence of a *best possible* matching in a bipartite graph, namely a matching that covers *all* vertices in one class of the vertex bipartition.

8.2.1 Theorem (Hall's theorem). *Let $G = (V, E)$ be a bipartite graph with bipartition $V = X \dot\cup Y$. For $T \subseteq X$, we define the neighborhood $N(T) \subseteq Y$ of T as the set*

$$N(T) = \{w \in V : \{v, w\} \in E \text{ for some } v \in T\}.$$

If for every $T \subseteq X$, $|N(T)| \geq |T|$ holds, then G has a matching that covers all vertices in X.

To prove Hall's Theorem, we use linear programming duality. (Combinatorial proofs are also available, actually simpler than the forthcoming proof, but we want to illustrate a typical use of duality on a simple example.) Actually, we derive the following statement, from which Hall's Theorem easily follows.

8.2.2 Theorem (König's theorem). *Let $G = (V, E)$ be a bipartite graph. The size of a maximum matching in G equals the size of a minimum vertex cover of G. Moreover, a maximum matching in G and a minimum vertex cover of G can be computed by linear programming.*

Let us first show how Hall's Theorem follows from König's Theorem. Afterwards, we present the material necessary to prove the latter theorem.

Proof of Theorem 8.2.1. Let $G = (X \dot\cup Y, E)$ be a graph with $|N(T)| \geq |T|$ for all $T \subseteq X$. We will show that any minimum vertex cover of G has the largest possible size $n_1 = |X|$. König's Theorem then implies that G has a matching of size n_1, and this matching obviously covers X.

For contradiction, suppose that there is a vertex cover C with k vertices from X and less than $n_1 - k$ vertices from Y, for some k. The set $T = X \setminus C$ has size $n_1 - k$ and satisfies $|N(T)| \geq n_1 - k$ by the assumption. But this implies that there is a vertex $w \in N(T)$ that is not in $C \cap Y$. Since this vertex has some neighbor $v \in T$, the edge $\{v, w\}$ is not covered by C, a contradiction. □

Totally unimodular matrices. A matrix A is called **totally unimodular** if every square submatrix of A (obtained from A by deleting some rows and some columns) has determinant 0, 1, or -1.

In particular, A has entries from $\{-1, 0, 1\}$. Such matrices are interesting since an integer program with a totally unimodular constraint matrix is easy to solve by simply solving its LP relaxation, as we will show in Lemma 8.2.4 below. Let us start with a preparatory step.

8.2.3 Lemma. *Let A be a totally unimodular matrix, and consider the matrix $\bar{A}$ obtained from A by appending a unit vector $\mathbf{e}_i$ as a new last column. Then $\bar{A}$ is totally unimodular as well.*

Proof. Let us fix a $k \times k$ submatrix Q of $\bar{A}$. If Q is a submatrix of A, then $\det(Q) \in \{-1, 0, 1\}$ by total unimodularity of A. Otherwise, we pick column k of Q (corresponding to the newly added column in $\bar{A}$) and compute $\det(Q)$ according to Laplace expansion on this column. Recall that

$$\det(Q) = \sum_{j=1}^{k} (-1)^{j+k} q_{jk} \det(Q^{jk}),$$

where Q^{jk} is the matrix resulting from Q by removing row j and column k. By construction, this latter column may be $\mathbf{0}$ (and we get $\det(Q) = 0$), or there is exactly one nonzero entry $q_{\ell k} = 1$. In that case,

$$\det(Q) = (-1)^{\ell+k} \det(Q^{\ell k}) \in \{-1, 0, 1\},$$

since $Q^{\ell k}$ is a submatrix of A. □

Here comes the promised application of total unimodularity.

8.2.4 Lemma. *Let us consider a linear program with n nonnegative variables and m inequalities of the form*

$$\begin{array}{ll} \textit{maximize} & \mathbf{c}^T\mathbf{x} \\ \textit{subject to} & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}, \end{array}$$

where $\mathbf{b} \in \mathbb{Z}^m$. If A is totally unimodular, and if the linear program has an optimal solution, then it also has an integral *optimal solution $\mathbf{x}^* \in \mathbb{Z}^n$.*

Proof. We first transform the linear program into equational form. The resulting system of equality constraints is $\bar{A}\bar{\mathbf{x}} = \mathbf{b}$, with $\bar{A} = (A|I_m)$ and $\bar{\mathbf{x}} \in \mathbb{R}^{n+m}$. Then we solve the problem using the simplex method. Let $\bar{\mathbf{x}}^*$ be an optimal basic feasible solution, associated with the feasible basis $B \subseteq \{1, 2, \ldots, n+m\}$. Then we know that the nonzero entries of $\bar{\mathbf{x}}^*$ are given by

$$\bar{\mathbf{x}}^*_B = \bar{A}_B^{-1}\mathbf{b};$$

see Section 5.5.

By Cramer's rule, the entries of $\bar{A}_B^{-1}$ can be written as rational numbers with common denominator $\det(\bar{A}_B)$. The matrix $\bar{A}_B$ is a square submatrix of $\bar{A}$, where $\bar{A}$ is totally unimodular by repeated application of Lemma 8.2.3. Since $\bar{A}_B$ is nonsingular, we get $\det(\bar{A}_B) \in \{-1, 1\}$, and the integrality of $\bar{\mathbf{x}}^*$ follows. □

Incidence matrices of bipartite graphs. Here is the link between total unimodularity and König's Theorem. Let $G = (X \dot\cup Y, E)$ be a bipartite graph with n vertices $v_1, \ldots, v_n$ and m edges $e_1, \ldots, e_m$. The **incidence matrix** of G is the matrix $A \in \mathbb{R}^{n \times m}$ defined by

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \in e_j \\ 0 & \text{otherwise.} \end{cases}$$

8.2.5 Lemma. *Let $G = (X \dot\cup Y, E)$ be a bipartite graph. The incidence matrix A of G is totally unimodular.*

Proof. We proceed by induction on the size k of the square submatrices of A. For $k = 1$, all 1×1 submatrices have determinant 0 or 1 by definition of the incidence matrix. Now we consider $k > 1$ and a $k\times k$ submatrix Q. Since the columns of Q correspond to edges, each column of Q has at most two nonzero entries (which are 1). If there is a column with only zero entries, we get $\det(Q) = 0$, and if there is a column with only one nonzero entry, we can expand the determinant by this column (as in the proof of Lemma 8.2.3) and get that up to sign, $\det(Q)$ equals the determinant of a $(k-1)\times(k-1)$ submatrix Q'. By induction, $\det(Q') \in \{-1, 0, 1\}$, so the same holds for Q.

Finally, if every column of Q contains precisely two nonzero entries, we claim that $\det(Q) = 0$. To see this, we observe that the sum of all rows of Q corresponding to vertices in X is the vector $(1, \ldots, 1)$, since for each column of Q, exactly one of its two nonzero entries comes from a vertex in X. For the same reason, we get $(1, \ldots, 1)$ by summing up the rows for vertices in Y, and it follows that the rows of Q are linearly dependent. □

Now we are ready to prove König's Theorem.

Proof of Theorem 8.2.2. We first consider the integer program

$$\begin{array}{ll} \text{maximize} & \sum_{j=1}^{m} x_j \\ \text{subject to} & A\mathbf{x} \leq \mathbf{1} \\ & \mathbf{x} \geq 0 \\ & \mathbf{x} \in \mathbb{Z}^m, \end{array}$$

where A is the incidence matrix of G. In this integer program, the row of A corresponding to vertex v_i induces the constraint

$$\sum_{j : e_j \ni v_i} x_j \leq 1.$$

This implies that $x_j \in \{0, 1\}$ for all j, and that the edges e_j with $\tilde{x}_j = 1$ in an optimal solution $\tilde{\mathbf{x}}$ form a maximum matching in G.

Next we consider the integer program

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{n} y_i \\ \text{subject to} & A^T\mathbf{y} \geq \mathbf{1} \\ & \mathbf{y} \geq 0 \\ & \mathbf{y} \in \mathbb{Z}^n, \end{array}$$

where A is as before the incidence matrix of G. In this integer program, the row of A^T corresponding to edge e_j induces the constraint

$$\sum_{i : v_i \in e_j} y_i \geq 1.$$

This implies that in any *optimal* solution $\tilde{\mathbf{y}}$ we have $\tilde{y}_i \in \{0, 1\}$ for all i, since any larger value could be decreased to 1. But then the vertices v_i with $\tilde{y}_i = 1$ in an optimal solution $\tilde{\mathbf{y}}$ form a minimum vertex cover of G.

To summarize, the optimum value σ of the first integer program is the size of a maximum matching in G, and the optimum value τ of the second integer program is the size of a minimum vertex cover in G.

In both integer programs, we may now drop the integrality constraints without affecting the optimum values: A, and therefore also A^T, are totally unimodular by Lemma 8.2.5, so Lemma 8.2.4 applies. But the resulting *linear* programs are dual to each other; the Duality Theorem thus proves $\sigma = \tau$, and this is the first part of Theorem 8.2.2.

To get a maximum matching from the first linear program, we simply need to find an *integral* optimal solution. When we use the simplex method to solve

the linear program, we get this for free, see Lemma 8.2.4 and in particular the claim towards the end of its proof. Otherwise, we can apply Theorem 4.2.3(ii) to construct a basic feasible solution from any given optimal solution, and this basic feasible solution will be integral. A minimum vertex cover is obtained from the second linear program in the same fashion. □

The previous proof shows more: Given edge weights $w_1, \ldots, w_m$, any optimal solution to the integer program

$$\begin{array}{ll} \text{maximize} & \sum_{j=1}^{m} w_j x_j \\ \text{subject to} & A\mathbf{x} \leq \mathbf{1} \\ & \mathbf{x} \geq 0 \\ & \mathbf{x} \in \mathbb{Z}^m, \end{array}$$

corresponds to a *maximum-weight matching* in G. Since we can as before relax the integrality constraints without affecting the optimum value, an integral optimal solution to the relaxation can be found and yields a maximum-weight matching in G. This solves the optimal job assignment problem if there are more employees than jobs.

The fact that a linear program with totally unimodular constraint matrix and integral right-hand side has an integral optimal solution implies something much stronger: Since every vertex of the feasible region is optimal for some objective function (see Section 4.4), we know that *all* vertices of the feasible region are integral. We say that the feasible region forms an *integral polyhedron*.

Such integrality results together with linear programming duality can yield interesting and powerful *minmax* theorems. König's theorem is one such example. Another classical minmax theorem that can be proved along these lines is the *max-flow-min-cut theorem* that we have briefly touched upon in Section 2.2.

To state this theorem, we consider a network modeled by a directed graph $G = (V, E)$ with edge capacities w_e. In section 2.2, we have interpreted them as maximum transfer rates of data links. Given two designated vertices, the source s and the sink t, the *maximum flow* value is the maximum rate at which data can flow from s to t through the network.

The *minimum cut* value, on the other hand, is the minimum total capacity of any set of data links whose breakdown disconnects t from s.

The max-flow-min-cut theorem states that the maximum flow value is *equal* to the minimum cut value. One of several proofs writes both values as optimal values of linear programs that are dual to each other and have integral optimal solutions.

Total dual integrality and Edmonds' matching polytope. There is another frequently used criterion that ensures integrality of the feasible region. A system $A\mathbf{x} \leq \mathbf{b}$ of linear inequalities is said to be *totally dual integral* (TDI) if the linear program

$$\begin{array}{ll} \text{minimize} & \mathbf{b}^T\mathbf{y} \\ \text{subject to} & A^T\mathbf{y} = \mathbf{c} \\ & \mathbf{y} \geq \mathbf{0} \end{array}$$

has an integral optimal solution $\tilde{\mathbf{y}}$ whenever $\mathbf{c}$ is an *integral* vector for which an optimal solution exists. It can then be shown that the (primal) problem

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} \leq \mathbf{b} \end{array}$$

has an integral optimal solution for *all* $\mathbf{c}$, provided that A is rational and $\mathbf{b}$ is integral. The virtue here is that we only need to prove something for integral $\mathbf{c}$ in order to get a statement for all $\mathbf{c}$.

The TDI notion is connected to total unimodularity as follows. A matrix A is totally unimodular if and only if the system $\{A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ is TDI for all integral vectors $\mathbf{b}$.

The most prominent example of a TDI system is Edmonds' linear description of the *matching polytope*. For a (not necessarily bipartite) graph $G = (V, E)$, the matching polytope is the convex hull of all *incidence vectors* of matchings. As in Section 3.2, the incidence vector of a matching $M \subseteq E$ is the $|E|$-dimensional vector $\mathbf{x}$ with

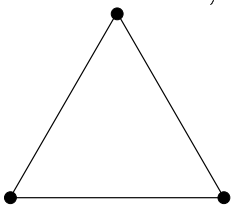
$$x_e = \begin{cases} 1, & \text{if } e \in M \\ 0, & \text{otherwise.} \end{cases}$$

Any such vector satisfies the inequalities

$$\begin{array}{rcll} x_e & \geq & 0, & e \in E \\ \sum_{e \ni v} x_e & \leq & 1, & v \in V. \end{array} \tag{8.9}$$

If G is a bipartite graph, we have shown above that the polytope P defined by these inequalities is integral (Lemmas 8.2.5 and 8.2.4). It follows that every vertex of P is actually the incidence vector of some matching. Consequently, P coincides with the matching polytope in the bipartite case, and the system (8.9) is an explicit inequality description of the matching polytope.

For nonbipartite graphs, this fails. Indeed, if we consider the triangle



then the polytope P defined by (8.9) has the five vertices $\mathbf{0}, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \frac{1}{2}(\mathbf{e}_1+\mathbf{e}_2+\mathbf{e}_3)$ and is therefore not integral.

Yet, there is a larger inequality system that leads to an integral polytope. This is based on the observation that every subset $A \subseteq V$ of odd size $2k+1$ supports at most k edges of any fixed matching. Therefore, the *odd subset inequalities*

$$\sum_{e \subseteq A} x_e \leq (|A| - 1)/2, \quad A \subseteq E, |A| \text{ odd} \tag{8.10}$$

hold for all incidence vectors of matchings. It can now be shown that the system of inequalities in (8.9) and (8.10) is TDI, and the implication is that these inequalities already define the matching polytope for a general graph G. In the triangle case, the only nontrivial odd subset inequality is the one for $A = V$:

$$\sum_{e \in E} x_e \leq 1.$$

This inequality cuts off the fractional vertex $\frac{1}{2}(\mathbf{e}_1 + \mathbf{e}_2 + \mathbf{e}_3)$ and leaves the tetrahedron that is the convex hull of $\mathbf{0}$, $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$, which is an integral polytope and, at the same time, the matching polytope of the triangle.

We now have an exponentially large but explicit inequality description of the matching polytope of a general graph. Having such a description for an integral polyhedron is an indication that it might be possible to optimize a linear function

over the polyhedron in polynomial time. In the case of matchings in general graphs, this can indeed be done. As shown by Edmonds in 1965, a maximum-weight matching in a general graph can be found in polynomial time, but the known algorithms are much more involved than the ones for the bipartite case discussed in Section 3.2.

8.3 Machine Scheduling

In the back office of the copy shop *Copy & Paste*, the operator is confronted with n copy jobs submitted by customers the night before. For processing them, he has m photocopying machines with different features at his disposal. For all i, j, the operator quickly estimates how long it would take the i-th machine to process the j-th job, and he makes a table of the resulting running times, like this:

	Single B&W	Duplex B&W	Duplex Color
Master's thesis, 90 pages two-sided, 10 B&W copies	—	45 min	60 min
All the best deals flyer, 1 page one-sided, 10,000 B&W copies	2h 45 min	4h 10 min	5h 30 min
Buyer's paradise flyer, 1 page one-sided, 10,000 B&W copies	2h 45 min	4h 10 min	5h 30 min
Obituary, 2 pages two-sided, 100 B&W copies	—	2 min	3 min
Party platform, 10 pages two-sided, 5,000 color copies	—	—	3h 30 min

Since the operator can go home as soon as all jobs have been processed, his goal is to find an assignment of jobs to machines (a *schedule*) such that the **makespan**—the time needed to finish all jobs—is minimized. In our example, this is not hard to figure out: For the party platform, there is no choice between machines. We can also observe that it is advantageous to use both B&W machines for processing the two flyers, no matter where the thesis and the obituary go. Given this, the makespan is at least 4h 55 min if the thesis is processed on the B&W machine, so it is better put on the color machine to achieve the optimum makespan of 4h 30 min (with the obituary running on the B&W machine).

In general, finding the optimum makespan is computationally difficult (NP-hard). The obvious approach of trying all possible schedules is of course not a solution for a larger number n of jobs. What we show in this section is that the operator can quickly compute a schedule whose makespan is at most *twice* as long as the optimum makespan. All he needs for that are some linear programming skills.

We should emphasize that in this scheduling problem the jobs are considered *indivisible*, and so each job must be processed on a single machine. This, in a sense, is what makes the problem difficult. As we will soon see, an optimal "fractional schedule", where a single job could be divided among several machines in arbitrary ratios, can be found efficiently by linear programming.

Two integer programs for the scheduling problem. Let us identify the m machines with the set $M := \{1, \ldots, m\}$ and the n jobs with the set $J := \{m+1, \ldots, m+n\}$.

Let d_{ij} denote the running time of job $j \in J$ on machine $i \in M$. We assume $d_{ij} > 0$. To simplify notation, we also assume that any machine can process any job: An infeasible assignment of job j to machine i can be modeled by a large running time $d_{ij} = K$. If K is larger than the sum of all "real" running times, the optimal schedule will avoid infeasible assignments, given that there is a feasible schedule at all.

With these notions, the following integer program in the variables t and $x_{ij}, i \in M, j \in J$ computes an assignment of jobs to machines that minimizes the makespan.

$$
\begin{array}{lrcll}
\text{minimize} & t & & & \\
\text{subject to} & \sum_{i \in M} x_{ij} & = & 1 & \text{for all } j \in J \\
& \sum_{j \in J} d_{ij} x_{ij} & \leq & t & \text{for all } i \in M \\
& x_{ij} & \geq & 0 & \text{for all } i \in M, j \in J \\
& x_{ij} & \in & \mathbb{Z} & \text{for all } i \in M, j \in J.
\end{array}
$$

Under the integrality constraints, the conditions $\sum_{i \in M} x_{ij} = 1$ and $x_{ij} \geq 0$ imply that $x_{ij} \in \{0, 1\}$ for all i, j. With the interpretation that $x_{ij} = 1$ if job j is assigned to machine i, and $x_{ij} = 0$ otherwise, the first n equations stipulate that each job is assigned to exactly one machine. The next m inequalities make sure that no machine needs more time than t to finish all jobs assigned to it. Minimizing t leads to equality for at least one of the machines, so the best t is indeed the makespan of an optimal schedule.

As we have already seen in Section 3.3 (the minimum vertex cover problem), solving the *LP relaxation* obtained from an integer program by deleting the integrality constraints can be a very useful step towards an approximate solution of the original problem. In our case, this approach needs an additional twist: we will relax another integer program, obtained by adding *redundant constraints* to the program above. This is similar to the *cutting plane* approach briefly discussed in Section 3.2: After dropping integrality, the added constraints are not redundant anymore and lead to a better LP relaxation.

Let t_{opt} be the makespan of an optimal schedule. If $d_{ij} > t_{\text{opt}}$, then we know that job j cannot run on machine i in any optimal schedule, so we may add the constraint $x_{ij} = 0$ to the integer program without affecting its validity. More generally, if T is any upper bound on t_{opt}, we can write down the following integer program that has the same optimal solutions as the original one.

$$
\begin{array}{lrcll}
\text{minimize} & t & & & \\
\text{subject to} & \sum_{i \in M} x_{ij} & = & 1 & \text{for all } j \in J \\
& \sum_{j \in J} d_{ij} x_{ij} & \leq & t & \text{for all } i \in M \\
& x_{ij} & \geq & 0 & \text{for all } i \in M, j \in J \\
& x_{ij} & = & 0 & \text{for all } i \in M, j \in J \text{ with } d_{ij} > T \\
& x_{ij} & \in & \mathbb{Z} & \text{for all } i \in M, j \in J.
\end{array}
$$

But we do not know t_{opt}, so what is the value of T we use for the relaxation? There is no need to specify this right here; for the time being, you can imagine

that we set $T = \max_{ij} d_{ij}$, a value that will definitely work, because it makes our second integer program coincide with the first one.

A good schedule from the LP relaxation. As already indicated, the first step is to solve the LP relaxation, denoted by LPR(T), of our second integer program:

$$\begin{array}{lrcll} \text{minimize} & t & & & \\ \text{subject to} & \sum_{i \in M} x_{ij} & = & 1 & \text{for all } j \in J \\ & \sum_{j \in J} d_{ij} x_{ij} & \leq & t & \text{for all } i \in M \\ & x_{ij} & \geq & 0 & \text{for all } i \in M, j \in J \\ & x_{ij} & = & 0 & \text{for all } i \in M, j \in J \text{ with } d_{ij} > T. \end{array}$$

Unlike in the vertex cover application, we cannot work with any optimal solution to the relaxation, though: We need a *basic feasible* optimal solution; see Section 4.2. To be more precise, we rely on the following property of a basic feasible solution; see Theorem 4.2.3.

8.3.1 Assumption. *The columns of the constraint matrix A corresponding to the nonzero variables $\tilde{x}_{ij}$ in the optimal solution are linearly independent.*

In case the simplex method is used to solve the relaxation, such a solution comes for free (the column of A corresponding to t could actually be added to the set of columns in the assumption, then, but this is not needed). Otherwise, we can easily construct a solution satisfying the assumption from any given optimal solution, according to the recipe in the proof of Theorem 4.2.3.

At this point the reader may wonder why we would want to use an algorithm different from the simplex method here, in particular when we are searching for a basic feasible optimal solution. The reason is of theoretical nature: We want to prove that a schedule whose makespan is at most twice the optimum can be found *in polynomial time*. As we have pointed out in the introductory part of Chapter 7, the simplex method is not known to run in polynomial time for any pivot rule, and for most pivot rules it simply does not run in polynomial time. For the theoretical result we want, we therefore better use one of the polynomial-time methods for solving linear programs, sketched in Chapter 7. For practical purposes, the simplex method will do, of course.

In general terms, what we are trying to develop here is a polynomial-time *approximation algorithm* for an NP-hard problem. Since complexity theory indicates that we will not be able to solve the problem exactly within reasonable time bounds, it is quite natural to ask for an approximate solution that can be obtained in polynomial time. The quality of an approximate solution is typically measured by the *approximation factor*, the ratio between the value of the approximate solution and the value of an optimal solution. In our approximation algorithm for the scheduling problem, this factor will be at most 2.

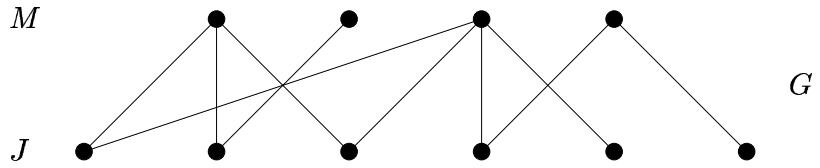
Let us fix the values $\tilde{t}$ and $\tilde{x}_{ij}$ of the variables in some optimal solution of the LP relaxation. We now consider the bipartite graph $G = (M \cup J, E)$, with

$$E = \{\{i, j\} \subseteq M \cup J \mid \tilde{x}_{ij} > 0\}.$$

For an arbitrary optimal solution, this graph could easily be a (boring) complete bipartite graph, but under Assumption 8.3.1 it becomes more interesting.

8.3.2 Lemma. *In any subgraph of G (obtained by deleting edges, or vertices with their incident edges), the number of edges is at most the number of vertices.*

The graph G for $m = 4$ machines and $n = 6$ jobs might look like this, for example:



Proof. Let A be the constraint matrix of the LP relaxation. It has one row for each machine, one for each job, and one for each runtime d_{ij} exceeding T. The columns of A corresponding to the nonzero variables $\tilde{x}_{ij}$ (equivalently, to the edges of G) are linearly independent by our assumption.

Now we consider any subgraph of G, with vertex set $M' \cup J' \subseteq M \cup J$ and edge set $E' \subseteq E$. Let A' be the submatrix obtained from A by restricting to rows corresponding to $M' \cup J'$, and to columns corresponding to E'.

Claim. *The columns of A' are linearly independent.*

> To see this, we first observe that the columns of A corresponding to E' are linearly independent, simply because $E' \subseteq E$. Any variable $x_{ij}, \{i,j\} \in E'$, occurs in the inequality for machine $i \in M'$, and in the equation for job $j \in J'$, but in no other equation, since $\tilde{x}_{ij} > 0$ implies $d_{ij} \leq T$. This means that the columns of A corresponding to E' have zero entries in all rows except for those corresponding to machines or jobs in $M' \cup J'$. Hence, these columns remain linearly independent even when we restrict them to the rows corresponding to $M' \cup J'$.

By the claim, we have $|E'| \leq |M' \cup J'|$, and this is the statement of the lemma. □

This structural result about G allows us to find a good schedule.

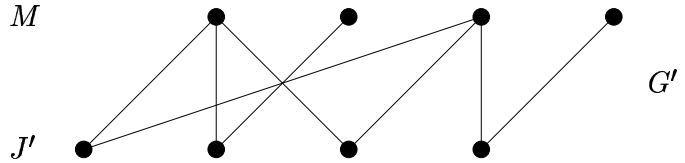
8.3.3 Lemma. *Let $T \geq 0$ be such that the LP relaxation* LPR(T) *is feasible, and suppose that a feasible solution is given that satisfies Assumption 8.3.1 and has value $t = \tilde{t}$. Then we can efficiently construct a schedule of makespan at most $\tilde{t} + T$.*

Proof. To construct a schedule, we need to assign each job to some machine. We begin with the jobs j that have degree one in the graph G, and we assign each such j to its unique neighbor i. By the construction of G, we have $\tilde{x}_{ij} = 1$ in this case. If machine i has been assigned a set S_i of jobs in this way, it processes these jobs in time

$$\sum_{j \in S_i} d_{ij} = \sum_{j \in S_i} d_{ij}\tilde{x}_{ij} \leq \sum_{j \in J} d_{ij}\tilde{x}_{ij} \leq \tilde{t}.$$

So each machine can handle the jobs assigned by this partial schedule in time $\tilde{t}$.

Next we remove all assigned jobs and their incident edges from G. This leaves us with a subgraph $G' = (M \cup J', E')$. In G', all vertices of degree one are machines. In the example depicted above, two jobs have degree one, and their deletion results in the following subgraph G'.

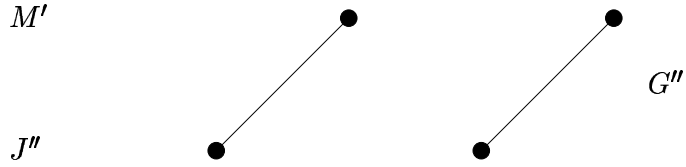


We will show that we can find a matching in G' that covers all the remaining jobs. If we assign the jobs according to this matching, each machine gets at most one additional job, and this job can be processed in time at most T by the construction of our second integer program. Therefore, the resulting full schedule has makespan at most $\tilde{t} + T$.

To find the matching, we use Lemma 8.3.2. The lemma implies that each connected component of G' is either a tree, or a tree with one extra edge connecting two of its vertices. In the latter case, the component has exactly one cycle of even length, because G' is bipartite. Therefore, we can match all jobs occurring on cycles, and after removing the vertices of all cycles, we are left with a subgraph G'', all of whose connected components are trees, *with at most one vertex per tree being a former neighbor of a cycle vertex.*

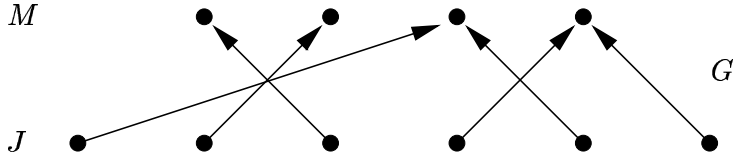
It follows that in every tree of G'', at most one vertex can be a job of degree one, since all other degree-one vertices already had degree one in G' and are therefore machines.

In our example, the graph G' is connected and has one cycle of length 4. After removing the cycle, we obtain the following graph G'':



The matching of the remaining jobs is easy. We root any tree in G'' at its unique job of degree one (or at any vertex if there is no such job), and we match every job to one of its children in the rooted tree. For this, we observe that there cannot be an isolated job in G'', since the removal of cycles can affect only one of the at least two neighbors of the job in G'. □

For our running example, here is a complete assignment of jobs to machines obtained from the described procedure.



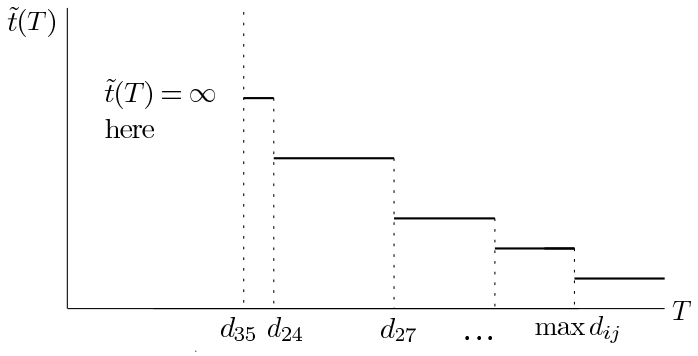
Choosing the parameter T. How good is the bound we get from the previous lemma? We will assume that $\tilde{t}$ is the value of an optimal basic feasible solution to the LP relaxation with parameter T. Then $\tilde{t}$ is a lower bound for the optimum makespan t_{opt}, so this part of the bound looks promising. But when we recall that T must be an *upper* bound for t_{opt} in order for our second integer program to be valid, it seems that we would have to choose $T = t_{\text{opt}}$ to get makespan at most $2t_{\text{opt}}$. But this cannot be done, since we have argued above that it is hard to compute t_{opt} (and if it were easy, there would be no need for an approximate solution, anyway).

Luckily, there is a way out. Reading Lemma 8.3.3 carefully, we see that T only needs to be chosen so that the *LP relaxation* LPR(T) is feasible, and there is no need for the second *integer* program to be feasible.

If LPR(T) is feasible, Lemma 8.3.3 allows us to construct a schedule with makespan at most $\tilde{t} + T$, so the best T is the one that minimizes $\tilde{t} + T$ subject to LPR(T) being feasible. Since $\tilde{t}$ depends on T, we make this explicit now by writing $\tilde{t} = \tilde{t}(T)$. If LPR(T) is infeasible, we set $\tilde{t}(T) = \infty$.

How to find the best T. We thus seek a point T^* in which the function $f(T) = \tilde{t}(T) + T$ attains a minimum.

First we observe that $\tilde{t}(T)$ is a step function like in the following picture:



Indeed, let us start with the value $T = \max_{ij} d_{ij}$, where $\tilde{t}(T)$ attains the smallest possible value, and let us decrease T continuously. The value of $\tilde{t}(T)$ may change (jump up) only at moments when a new constraint of the form $x_{ij} = 0$ appears in LPR(T), and this happens only for $T = d_{ij}$. Between these values the function $\tilde{t}(T)$ stays constant.

Consequently, the function $f(T) = \tilde{t}(T) + T$ is linearly increasing on each interval between two consecutive d_{ij}'s, and the minimum is attained at some d_{ij}. So we can compute the minimum, and the desired best value T^*, by solving at most mn linear programs of the form LPR(T), with T ranging over all d_{ij}. Under our convention that $\tilde{t}(T) = \infty$ if LPR(T) is infeasible, the minimum will be attained at a value T^* with LPR(T^*) feasible.

8.3.4 Theorem. *Let T^* be the value of T that minimizes $\tilde{t}(T) + T$. With $T = T^*$, the algorithm in the proof of Lemma 8.3.3 computes a schedule of makespan at most $2t_{\text{opt}}$.*

Proof. We know that for $T = t_{\text{opt}}$, the second integer program is feasible and has optimum value t_{opt}. Hence LPR(t_{opt}) is feasible as well and its optimum value can be only smaller: $\tilde{t}(t_{\text{opt}}) \leq t_{\text{opt}}$. We thus have

$$\begin{aligned} \tilde{t}(T^*) + T^* &= \min_T \Big(\tilde{t}(T) + T \Big) \\ &\leq \tilde{t}(t_{\text{opt}}) + t_{\text{opt}} \\ &\leq 2t_{\text{opt}}. \end{aligned}$$

□

The 2-approximation algorithm for the scheduling problem is adapted from the paper

> J.K. Lenstra, D.B. Shmoys, É. Tardos: Approximation algorithms for scheduling unrelated parallel machines, *Mathematical Programming* 46(1990), pp. 259–271.

The paper also proves that it is NP-hard to approximate the optimum makespan with a factor less than 3/2.

Aiming at simplicity, we have presented a somewhat inefficient version of the algorithm. A considerably faster version, with the same approximation guarantee, can be obtained if we do not minimize the function $T \mapsto \tilde{t}(T) + T$, but instead we only look for the smallest T with $\tilde{t}(T) \leq T$. Such a value of T still guarantees $\tilde{t}(T) + T \leq 2t_{\text{opt}}$, but it can be found by binary search over the sorted sequence of the d_{ij}. In this way, it suffices to solve LPR(T) for $O(\log mn)$ values of T, instead of mn values as in our presentation. See the paper quoted above for details.

8.4 Upper Bounds for Codes

Error-correcting codes. Alice and Bob, playing zero-sum games against each other over the Internet, have to communicate moves. They have agreed on encoding each move by a sequence $\mathbf{w} \in \{0,1\}^n$ of n bits which we call a **word.** Since both are worried about their moves being misunderstood due to typos, they have chosen an **error-correcting code** with parameter r. Such a code has the property that Bob, say, can correctly interpret the n-bit word sent by Alice, even if Alice has made up to r typos. A typo is a wrong bit at any of the n positions.

If $r > 0$, it is clear that not all words can be used to encode moves. For example, if $n = 3$ and $r = 1$, $\mathcal{C} = \{000, 111\}$ is a largest error-correcting code. It can correct one typo, since Bob can deduce the word Alice intended to send, by simply counting the number of 1's in the word he receives. If this number is at most one, Alice meant 000, otherwise she wanted to send 111. $\mathcal{C}$ is largest since among any set of three words from $\{0,1\}^3$, there are two that differ in at most two positions, for example 000 and 011. When Bob receives 010, he cannot tell whether Alice intended 000 or 011, since the received word is "one typo away" from both of them.

In general, the chosen set $\mathcal{C}$ of words (the code) must have the property that any two distinct words in $\mathcal{C}$ differ in at least $2r + 1$ of the n bits. Indeed,

assuming that there are two distinct words $\mathbf{w}', \mathbf{w}'' \in \mathcal{C}$ that differ in at most $2r$ bits, we consider any word $\mathbf{w}$ resulting from $\mathbf{w}'$ by flipping exactly half of the bits (rounded down) that distinguish $\mathbf{w}'$ from $\mathbf{w}''$. When Bob receives the word $\mathbf{w}$, he cannot tell whether Alice meant $\mathbf{w}'$ or $\mathbf{w}''$, because $\mathbf{w}$ can be obtained from both of them by making at most r typos.

On the other hand, if any two distinct code words differ by at least $2r+1$ bits, we do have an error-correcting code: In this case, for any word $\mathbf{w} \in \{0,1\}^n$ there is at most one code word from which $\mathbf{w}$ can be obtained through r or less typos, and this must be the word that was intended when $\mathbf{w}$ was received.

Given that n and r are fixed, it is natural for Alice and Bob to choose an error-correcting code $\mathcal{C} \subseteq \{0,1\}^n$ with the largest possible number of code words. That way, they maximize the number of different moves that can be communicated using the code.

The problem of finding the largest codes for given n and r can *in principle* be solved by complete enumeration: We can go through all possible subsets of $\{0,1\}^n$ and output the largest one that gives an error-correcting code. However, this method becomes practically infeasible already for very small n. It turns out that the problem, for arbitrary n and r, is computationally difficult (NP-hard). Starting from very moderate values of n and r, the maximum code sizes are not exactly known, except for few lucky cases. Tightening the known upper and lower bounds on maximum sizes of error-correcting codes is the topic of ongoing research in coding theory.

In this section we present a technique for proving upper bounds based on linear programming. When this technique was introduced by Philippe Delsarte in 1973, it provided upper bounds of unprecedented quality.

Terminology. The **Hamming distance** of two words $\mathbf{w}, \mathbf{w}' \in \{0,1\}^n$ is the number of bits in which $\mathbf{w}$ differs from $\mathbf{w}'$,

$$d_H(\mathbf{w}, \mathbf{w}') := |\{j \in \{1, \ldots, n\} : w_j \neq w'_j\}|.$$

The Hamming distance can be interpreted as the number of typos "necessary" to transform $\mathbf{w}$ into $\mathbf{w}'$. The **weight** of $\mathbf{w} \in \{0,1\}^n$ is the number of 1's in $\mathbf{w}$,

$$|\mathbf{w}| := |\{j \in \{1, \ldots, n\} : w_j = 1\}|.$$

Finally, for $\mathbf{w}, \mathbf{w}' \in \{0,1\}^n$, we define their sum modulo 2 as the word

$$\mathbf{w} \oplus \mathbf{w}' = ((w_1 + w'_1) \bmod 2, \ldots, (w_n + w'_n) \bmod 2) \in \{0,1\}^n.$$

These three notions are interrelated by the formula

$$d_H(\mathbf{w}, \mathbf{w}') = |\mathbf{w} \oplus \mathbf{w}'|. \tag{8.11}$$

8.4.1 Definition. *A code $\mathcal{C} \subseteq \{0,1\}^n$ has* **distance** *d if $d_H(\mathbf{w}, \mathbf{w}') \geq d$ for any two distinct words $\mathbf{w}, \mathbf{w}'$ in $\mathcal{C}$. For $n, d \geq 0$, let $A(n,d)$ denote the maximum cardinality of a code $\mathcal{C} \subseteq \{0,1\}^n$ with distance d.*

By our above discussion, an error-correcting code with parameter r corresponds to a code of distance $d = 2r+1$, and the maximum size of such a code is $A(n, 2r+1)$.

Special cases. For all n, we have $A(n,1) = 2^n$, because any code has distance 1. The case $d = 2$ is slightly more interesting. By choosing $\mathcal{C}$ as the set of all words of even weight, we see that $A(n,2) \geq 2^{n-1}$. But we actually have $A(n,2) = 2^{n-1}$, since it is easy to show by induction that every code with more than 2^{n-1} words contains two words of Hamming distance 1.

Given the simplicity of the cases $d = 1, 2$, it may come as a surprise that already for $d = 3$, little is known. This is the setup for error-correcting codes with one typo allowed. Exact values of $A(n,3)$ have only been determined up to $n \leq 16$; for $n = 17$, for example, the known bounds are

$$5312 \leq A(17,3) \leq 6552.$$

The sphere packing bound. For any n and d, a simple upper bound on $A(n,d)$ can be obtained by a *volume argument*. Let us motivate this with a real-life analogy. The local grocery is exhibiting a large glass box filled with peas, and the person to make the most accurate estimate of the number of peas in the box wins a prize. Without any counting, you can conclude that the number of peas is bounded above by the volume of the box divided by the volume of a single pea (assuming that all the peas have the same volume).

The same kind of argument can be used for the number $A(n,d)$, where we may assume in our application that $d = 2r+1$ is odd. Let us fix any code $\mathcal{C}$ of distance d. Now we think of the set $\{0,1\}^n$ as the glass box, and of the $|\mathcal{C}|$ *Hamming balls*

$$B(\mathbf{w}, r) := \{\mathbf{w}' \in \{0,1\}^n : d_H(\mathbf{w}, \mathbf{w}') \leq r\}, \quad \mathbf{w} \in \mathcal{C},$$

as the peas. Since the code has distance d, all these Hamming balls are disjoint and correspond in our analogy to peas. Consequently, their number cannot be larger than the total number of words (the volume of the box) divided by the number of words in a single Hamming ball (the volume of a pea). The number of words at Hamming distance exactly i from $\mathbf{w}$ is $\binom{n}{i}$. This implies

$$|B(\mathbf{w}, r)| = \sum_{i=0}^{r} \binom{n}{i},$$

and the following upper bound on $A(n, 2r+1)$ is obtained.

8.4.2 Lemma (Sphere packing bound). *For all n and r,*

$$A(n, 2r+1) \leq \left\lfloor \frac{2^n}{\sum_{i=0}^{r} \binom{n}{i}} \right\rfloor.$$

For example, the sphere packing bounds gives $A(17,3) \leq \lfloor 131072/18 \rfloor = 7281$.

In the following theorem, which is the main result of this section, an upper bound on $A(n,d)$ is expressed as an optimum of a certain linear program.

8.4.3 Theorem (The Delsarte bound). *For integers n, i, t with $0 \leq i, t \leq n$, let us put*

$$K_t(n,i) = \sum_{j=0}^{\min(i,t)} (-1)^j \binom{i}{j} \binom{n-i}{t-j}.$$

Then for every n and d, $A(n,d)$ is bounded above by the optimum value of the following linear program in variables $x_0, x_1, \ldots, x_n$:

$$\begin{array}{ll} \textit{maximize} & x_0 + x_1 + \cdots + x_n \\ \textit{subject to} & x_0 = 1 \\ & x_1 = x_2 = \cdots = x_{d-1} = 0 \\ & \sum_{i=0}^{n} K_t(n,i) \cdot x_i \geq 0, \qquad t = 1, \ldots, n \\ & x_0, x_1, \ldots, x_n \geq 0. \end{array}$$

Example. Using the sphere packing bound, we have previously found $A(17,3) \leq 7281$. To compute the Delsarte bound, we solve the linear program in the theorem (after eliminating x_0, x_1, x_2, which are actually constants, we have 15 nonnegative variables and 17 constraints). The optimum value is $32763/5 = 6552\frac{3}{5}$, which implies $A(17,3) \leq 6553$. The current best upper bound is 6552, an improvement only by 1!

Towards an explanation. The proof of the Delsarte bound will proceed as follows. With every subset $\mathcal{C} \subseteq \{0,1\}^n$ we associate nonnegative real quantities $\tilde{x}_0, \tilde{x}_1, \ldots, \tilde{x}_n$ such that $|\mathcal{C}| = \tilde{x}_0 + \cdots + \tilde{x}_n$. Then we will show that whenever $\mathcal{C}$ is a code with distance d, the $\tilde{x}_i$ constitute a a feasible solution of the linear program in the theorem. It follows that the maximum of the linear program is at least as large as the size of any existing code $\mathcal{C}$ (but of course, it may be larger, since not every feasible solution necessarily corresponds to a code).

Given $\mathcal{C} \subseteq \{0,1\}^n$, the $\tilde{x}_i$ are defined by

$$\tilde{x}_i = \frac{1}{|\mathcal{C}|} \cdot \Big|\{(\mathbf{w}, \mathbf{w}') \in \mathcal{C}^2 : d_H(\mathbf{w}, \mathbf{w}') = i\}\Big|, \quad i = 0, \ldots, n.$$

Thus, $\tilde{x}_i$ is the number of ordered pairs of code words with Hamming distance i, divided by the total number of code words. Since any of the $|\mathcal{C}|^2$ ordered pairs contributes to exactly one of the $\tilde{x}_i$, we have

$$\tilde{x}_0 + \tilde{x}_1 + \cdots + \tilde{x}_n = |\mathcal{C}|.$$

Some of the constraints in the linear program in Theorem 8.4.3 are now easy to understand. We clearly have $\tilde{x}_0 = 1$, since every $\mathbf{w} \in \mathcal{C}$ has zero distance only to itself. The constraints $\tilde{x}_1 = 0$ through $\tilde{x}_{d-1} = 0$ express the assumption that $\mathcal{C}$ has distance d; that is, there are no pairs of code words with Hamming distance between 1 and $d-1$. Interestingly, this is the *only* place in the proof of the Delsarte bound where the assumption of $\mathcal{C}$ being a code with distance d is used.

The remaining set of constraints is considerably harder to derive, and it lacks a really intuitive explanation. Thus, to prove Theorem 8.4.3, we have to establish the following.

8.4.4 Proposition. *Let $\mathcal{C} \subseteq \{0,1\}^n$ be an arbitrary set, let $\tilde{x}_i = \tilde{x}_i(\mathcal{C})$ be defined as above, and let $t \in \{1, \ldots, n\}$. Then we have the inequality*

$$\sum_{i=0}^{n} K_t(n,i) \cdot \tilde{x}_i \geq 0.$$

In the next lemma, $I \subseteq \{1,2,\ldots,n\}$ is a set of indices, and we write $d_H^I(\mathbf{w},\mathbf{w}')$ for the number of indices $i \in I$ with $w_i \neq w'_i$ (thus, the components outside I are ignored).

8.4.5 Lemma. *Let $I \subseteq \{1,2,\ldots,n\}$ be an arbitrary nonempty set of indices, and let $\mathcal{C} \subseteq \{0,1\}^n$. Then the number of pairs $(\mathbf{w},\mathbf{w}') \in \mathcal{C}^2$ with $d_H^I(\mathbf{w},\mathbf{w}')$ even is at least as large as the number of pairs $(\mathbf{w},\mathbf{w}') \in \mathcal{C}^2$ with $d_H^I(\mathbf{w},\mathbf{w}')$ odd. (In probabilistic terms, if we choose $\mathbf{w},\mathbf{w}' \in \mathcal{C}$ independently at random, then the probability that they differ in an even number of positions from I is at least as large as the probability that they differ in an odd number of positions from I.)*

Proof. Let us write $|\mathbf{w}|_I = |\{i \in I : w_i = 1\}|$, and let us set

$$\mathcal{E} = \{\mathbf{w} \in \mathcal{C} : |\mathbf{w}|_I \text{ is even}\}, \quad \mathcal{O} = \{\mathbf{w} \in \mathcal{C} : |\mathbf{w}|_I \text{ is odd}\}.$$

We observe that if $d_H^I(\mathbf{w},\mathbf{w}')$ is even, then $|\mathbf{w}|_I$ and $|\mathbf{w}'|_I$ have the same parity, and so $\mathbf{w}$ and $\mathbf{w}'$ are both in $\mathcal{E}$ or both in $\mathcal{O}$. On the other hand, for $d_H^I(\mathbf{w},\mathbf{w}')$ odd, one of $\mathbf{w}$, $\mathbf{w}'$ lies in $\mathcal{E}$ and the other one in $\mathcal{O}$. So the assertion of the lemma is equivalent to $|\mathcal{E}|^2 + |\mathcal{O}|^2 \geq 2 \cdot |\mathcal{E}| \cdot |\mathcal{O}|$, which follows by expanding $(|\mathcal{E}| - |\mathcal{O}|)^2 \geq 0$. □

8.4.6 Corollary. *For every $\mathcal{C} \subseteq \{0,1\}^n$ and every $\mathbf{v} \in \{0,1\}^n$ we have*

$$\sum_{(\mathbf{w},\mathbf{w}') \in \mathcal{C}^2} (-1)^{(\mathbf{w} \oplus \mathbf{w}')^T \mathbf{v}} \geq 0.$$

Proof. This is just another way of writing the statement of Lemma 8.4.5. Indeed, if we set $I = \{i : v_i = 1\}$, then $(\mathbf{w} \oplus \mathbf{w}')^T \mathbf{v} = d_H^I(\mathbf{w},\mathbf{w}')$, and hence the sum in the corollary is exactly the number of pairs $(\mathbf{w},\mathbf{w}')$ with $d_H^I(\mathbf{w},\mathbf{w}')$ even minus the number of pairs with $d_H^I(\mathbf{w},\mathbf{w}')$ odd.

The corollary also has a quick algebraic proof, which some readers may prefer. It suffices to note that $(\mathbf{w} \oplus \mathbf{w}')^T \mathbf{v}$ has the same parity as $(\mathbf{w} + \mathbf{w}')^T \mathbf{v}$ (addition modulo 2 was replaced by ordinary addition of integers), and so

$$\begin{aligned}
\sum_{(\mathbf{w},\mathbf{w}') \in \mathcal{C}^2} (-1)^{(\mathbf{w} \oplus \mathbf{w}')^T \mathbf{v}} &= \sum_{(\mathbf{w},\mathbf{w}') \in \mathcal{C}^2} (-1)^{(\mathbf{w} + \mathbf{w}')^T \mathbf{v}} \\
&= \sum_{(\mathbf{w},\mathbf{w}') \in \mathcal{C}^2} (-1)^{\mathbf{w}^T \mathbf{v}} \cdot (-1)^{\mathbf{w}'^T \mathbf{v}} \\
&= \left(\sum_{\mathbf{w} \in \mathcal{C}} (-1)^{\mathbf{w}^T \mathbf{v}} \right)^2 \geq 0.
\end{aligned}$$

□

Proof of Proposition 8.4.4. To prove the tth inequality in the proposition, i.e., $\sum_{i=0}^n K_t(n,i) \cdot \tilde{x}_i \geq 0$, we sum the inequality in Corollary 8.4.6 over all

$\mathbf{v} \in \{0,1\}^n$ of weight t. Interchanging the summation order, we obtain

$$\begin{aligned} 0 &\leq \sum_{\mathbf{v}\in\{0,1\}^n:\,|\mathbf{v}|=t} \sum_{(\mathbf{w},\mathbf{w}')\in\mathcal{C}^2} (-1)^{(\mathbf{w}\oplus\mathbf{w}')^T\mathbf{v}} \\ &= \sum_{(\mathbf{w},\mathbf{w}')\in\mathcal{C}^2} \sum_{\mathbf{v}\in\{0,1\}^n:\,|\mathbf{v}|=t} (-1)^{(\mathbf{w}\oplus\mathbf{w}')^T\mathbf{v}}. \end{aligned}$$

To understand the last expression, let us fix $\mathbf{u} = \mathbf{w} \oplus \mathbf{w}'$ and write $S(\mathbf{u}) = \sum_{\mathbf{v}\in\{0,1\}^n:\,|\mathbf{v}|=t}(-1)^{\mathbf{u}^T\mathbf{v}}$. In this sum, the $\mathbf{v}$ with $\mathbf{u}^T\mathbf{v} = j$ are counted with sign $(-1)^j$. How many $\mathbf{v}$ of weight t and with $\mathbf{u}^T\mathbf{v} = j$ are there? Let $i = |\mathbf{u}| = d_H(\mathbf{w},\mathbf{w}')$ be the number of 1's in $\mathbf{u}$. In order to form such a $\mathbf{v}$, we need to put j ones to positions where $\mathbf{u}$ has 1's and $t-j$ ones to positions where $\mathbf{u}$ has 0's. Hence the number of these $\mathbf{v}$ is $\binom{i}{j}\binom{n-i}{t-j}$, and

$$S(\mathbf{u}) = \sum_{j=0}^{\min(i,t)} (-1)^j \binom{i}{j}\binom{n-i}{t-j},$$

which we recognize as $K_t(n,i)$. So we have

$$0 \leq \sum_{(\mathbf{w},\mathbf{w}')\in\mathcal{C}^2} K_t(n, d_H(\mathbf{w},\mathbf{w}')),$$

and it remains to note that the number of times $K_t(n,i)$ appears in this sum is $|\mathcal{C}| \cdot \tilde{x}_i$. This finishes the proof of Proposition 8.4.4 and thus also of Theorem 8.4.3. □

A small strengthening of the Delsarte bound. We have seen that Theorem 8.4.3 yields $A(17,3) \leq 6553$. We show how the inequalities in the theorem can be slightly strengthened using a parity argument, which leads to the the best known upper bound $A(17,3) \leq 6552$. Similar tricks can improve the Delsarte bound in some other cases as well, but the improvements are usually minor.

For contradiction let us suppose that $n = 17$ and there is a code $\mathcal{C} \subseteq \{0,1\}^n$ of distance 3 with $|\mathcal{C}| = 6553$. The size of $\mathcal{C}$ is odd, and we note that for every code of odd size and every t, the last inequality in the proof of Corollary 8.4.6 can be strengthened to

$$\left(\sum_{\mathbf{w}\in\mathcal{C}} (-1)^{\mathbf{w}^T\mathbf{v}}\right)^2 \geq 1,$$

since an odd number of values from $\{-1,1\}$ cannot sum to zero. If we propagate this improvement through the proof of Proposition 8.4.4, we arrive at the following inequality for the $\tilde{x}_i$:

$$\sum_{i=0}^{n} K_t(n,i) \cdot \tilde{x}_i \geq \frac{\binom{n}{t}}{|\mathcal{C}|}.$$

Since in our particular case we suppose $|\mathcal{C}| = 6553$, we can replace the constraints $\sum_{i=0}^n K_t(n,i) \cdot x_i \geq 0$, $t = 1,2,\ldots,n$, in the linear program in Theorem 8.4.3 by $\sum_{i=0}^n K_t(n,i) \cdot x_i \geq \binom{n}{t}/6553$, and the $\tilde{x}_i$ defined by our $\mathcal{C}$ remain a feasible solution. However, the optimum of this modified linear program is only $6552\frac{3}{5}$, which contradicts the assumption $|\mathcal{C}| = 6553$. This proves $A(17,3) \leq 6552$.

The paper

M. R. Best, A. E. Brouwer, F. J. MacWilliams, A. M. Odlyzko, and N. J. A. Sloane: Bounds for binary codes of length less than 25, *IEEE Trans. Inform. Theory* 24 (1978) 81–93.

describes this particular strengthening of the Delsarte bound and some similar approaches. A continually updated table of the best known bounds for $A(n,d)$ for small n and d is maintained by Andries Brouwer at

`http://www.win.tue.nl/~aeb/codes/binary-1.html`.

The Delsarte bound explained. The result in Theorem 8.4.3 goes back to the thesis of Philippe Delsarte:

P. Delsarte: An algebraic approach to the association schemes of coding theory, *Philips Res. Repts. Suppl.* 10 (1973)

Here we sketch Delsarte's original proof. At comparable level of detail, his proof is more involved than the ad hoc proof above (from the paper by Best et al.). On the other hand, Delsarte's proof is more systematic and, even more importantly, it can be extended to prove a stronger result, which we mention below.

For $i \in \{0,\dots,n\}$, let M_i be the $2^n \times 2^n$ matrix defined by[1]

$$(M_i)_{\mathbf{v},\mathbf{w}} = \begin{cases} 1 & \text{if } d_H(\mathbf{v},\mathbf{w}) = i, \\ 0 & \text{otherwise.} \end{cases}$$

The set of matrices of the form

$$\sum_{i=0}^{n} y_i M_i, \quad y_0,\dots,y_n \in \mathbb{R}$$

is known to be closed under addition and scalar multiplication (this is clear), and under matrix multiplication (this has to be shown). A set of matrices closed under these operations is called a **matrix algebra**. In our case, one speaks of the **Bose–Mesner algebra of the Hamming association scheme**. The matrix multiplication turns out to be commutative on this algebra, and this is known to imply a strong condition: The M_i have a *common* diagonalization, meaning that there is an orthogonal matrix U with $U^T M_i U$ diagonal for all i.

Once we know this, it is a matter of patience to find such a matrix U. For example, the matrix U defined by

$$U_{\mathbf{v},\mathbf{w}} = \frac{1}{2^{n/2}}(-1)^{\mathbf{v}^T\mathbf{w}}$$

will do. First we have to check (this is easy) that this matrix is indeed orthogonal, meaning that $U^T U = I_n$.

For the entries of $U^T M_i U$, we can derive the formula

$$(U^T M_i U)_{\mathbf{v},\mathbf{w}} = \frac{1}{2^n} \sum_{\substack{(\mathbf{u},\mathbf{u}')\in(\{0,1\}^n)^2 \\ d_H(\mathbf{u},\mathbf{u}')=i}} (-1)^{\mathbf{u}^T\mathbf{v}+\mathbf{u}'^T\mathbf{v}}.$$

If $\mathbf{v} \neq \mathbf{w}$, this evaluates to 0, showing that $U^T M_i U$ is indeed a diagonal matrix. To prove this, we let j be any index for which $v_j \neq w_j$. In the sum, we can then pair up the terms for $(\mathbf{u},\mathbf{u}')$ and $(\mathbf{u}\oplus\mathbf{e}_j, \mathbf{u}'\oplus\mathbf{e}_j)$, with $\mathbf{e}_j$ being the word with a 1 exactly at position j. This pairing covers all terms of the sum, and paired-up

[1] We assume that rows and columns are indexed by the words from $\{0,1\}^n$.

terms are easily seen to cancel each other. (If you didn't believe $U^TU = I_n$, this can be shown with an even simpler pairing argument along these lines.)

On the diagonal, we get

$$\begin{aligned}(U^TM_iU)_{\mathbf{w},\mathbf{w}} &= \frac{1}{2^n}\sum_{\substack{(\mathbf{u},\mathbf{u}')\in(\{0,1\}^n)^2\\ d_H(\mathbf{u},\mathbf{u}')=i}}(-1)^{(\mathbf{u}\oplus\mathbf{u}')^T\mathbf{w}}\\ &= \sum_{\substack{\mathbf{v}\in\{0,1\}^n\\ |\mathbf{v}|=i}}(-1)^{\mathbf{v}^T\mathbf{w}} = K_i(n,|\mathbf{w}|)\end{aligned}$$

(for the last equality see the proof of Proposition 8.4.4) since any $\mathbf{v}$ of weight i can be written in the form $\mathbf{v} = \mathbf{u}\oplus\mathbf{u}'$ in 2^n different ways, one for each $\mathbf{u}\in\{0,1\}^n$.

Let us next fix a code $\mathcal{C}$ and look at a specific matrix in the Bose–Mesner algebra. For this, we define the values

$$\tilde{y}_i = \frac{|\{(\mathbf{w},\mathbf{w}')\in\mathcal{C}^2 : d_H(\mathbf{w},\mathbf{w}') = i\}|}{2^n\binom{n}{i}}, \quad i = 0,\ldots,n.$$

We note that $\tilde{y}_i$ is the probability that a randomly chosen pair of words with Hamming distance i is a pair of code words. Moreover, $\tilde{y}_i$ is related to our earlier quantity $\tilde{x}_i$ via

$$\tilde{y}_i = \frac{|\mathcal{C}|}{2^n\binom{n}{i}}\tilde{x}_i. \tag{8.12}$$

Here comes Delsarte's main insight.

8.4.7 Lemma. *The matrix $\tilde{M} = \sum_{i=0}^n \tilde{y}_iM_i$ is positive semidefinite.*

Proof. We first observe that

$$\tilde{M}_{\mathbf{v},\mathbf{w}} = \tilde{y}_i,$$

where $i = d_H(\mathbf{v},\mathbf{w})$. We will express $\tilde{M}$ as a positive linear combination of matrices that are obviously positive semidefinite. We start with the matrix $X^{\mathcal{C}}$ defined by

$$X^{\mathcal{C}}_{\mathbf{v},\mathbf{w}} = \begin{cases}1 & \text{if } (\mathbf{v},\mathbf{w})\in\mathcal{C}^2,\\ 0 & \text{otherwise.}\end{cases}$$

This matrix is positive semidefinite, since it can be written in the form

$$X^{\mathcal{C}} = \mathbf{x}^{\mathcal{C}}(\mathbf{x}^{\mathcal{C}})^T,$$

where $\mathbf{x}^{\mathcal{C}}$ is the characteristic vector of $\mathcal{C}$:

$$\mathbf{x}^{\mathcal{C}}_{\mathbf{w}} := \begin{cases}1 & \text{if } \mathbf{w}\in\mathcal{C},\\ 0 & \text{otherwise.}\end{cases}$$

Let Π be the automorphism group of $\{0,1\}^n$, consisting of the $n!2^n$ bijections that permute indices and swap 0's with 1's at selected positions. With τ uniformly chosen at random from Π, we obtain[2]

$$\begin{aligned}\mathrm{Prob}\Big[(\mathbf{v},\mathbf{w})\in\tau(\mathcal{C})^2\Big] &= \frac{1}{|\Pi|}\sum_{\pi\in\Pi}\Big[(\mathbf{v},\mathbf{w})\in\pi(\mathcal{C})^2\Big]\\ &= \frac{1}{|\Pi|}\sum_{\pi\in\Pi}X^{\pi(\mathcal{C})}_{\mathbf{v},\mathbf{w}}.\end{aligned}$$

[2]The *indicator variable* $[P]$ of a statement P has value 1 if P holds and 0 otherwise.

On the other hand,

$$\mathrm{Prob}\Big[(\mathbf{v},\mathbf{w}) \in \tau(\mathcal{C})^2\Big] = \mathrm{Prob}\Big[(\tau^{-1}(\mathbf{v}),\tau^{-1}(\mathbf{w})) \in \mathcal{C}^2\Big] = \tilde{y}_i,$$

since τ^{-1} is easily shown to map $(\mathbf{v},\mathbf{w})$ to a *random* pair of words with Hamming distance i.

This shows that

$$\tilde{M} = \frac{1}{|\Pi|}\sum_{\pi\in\Pi} X^{\pi(C)}$$

is a positive linear combination of positive semidefinite matrices and is therefore positive semidefinite itself. The lemma is proved. □

After diagonalization of $\tilde{M}$ by the matrix U, the statement of Lemma 8.4.7 can equivalently be written as

$$\sum_{i=0}^{n} \tilde{y}_i (U^T M_i U)_{\mathbf{w},\mathbf{w}} = \sum_{i=0}^{n} \tilde{y}_i K_i(n,|\mathbf{w}|) \geq 0, \quad \mathbf{w} \in \{0,1\}^n.$$

This is true since diagonalization preserves the property of being positive semidefinite, which for diagonal matrices is equivalent to nonnegativity of all diagonal entries.

Taking into account the relation (8.12) between the $\tilde{y}_i$ and our original $\tilde{x}_i$, this implies the following linear constraints for any code $\mathcal{C}$.

$$\sum_{i=0}^{n} \tilde{x}_i K_i(n,t) \Big/ \binom{n}{i} \geq 0, \quad t = 1,\ldots,n. \tag{8.13}$$

Observing that

$$\binom{t}{j}\binom{n-t}{i-j} \Big/ \binom{n}{i} = \binom{i}{j}\binom{n-i}{t-j} \Big/ \binom{n}{t},$$

we get (cf. the definition of K_t in Theorem 8.4.3)

$$K_i(n,t) \Big/ \binom{n}{i} = K_t(n,i) \Big/ \binom{n}{t}.$$

Under this equation, the inequalities in (8.13) are equivalent to those in Proposition 8.4.4 and we recover the Delsarte bound.

Beyond the Delsarte bound. Alexander Schrijver generalized Delsarte's approach and improved the upper bounds on $A(n,d)$ significantly in many cases:

> A. Schrijver: New code upper bounds from the Terwilliger algebra and semidefinite programming, *IEEE Trans. Inform. Theory* 51 (2005) 2859–2866.

His work uses semidefinite programming instead of linear programming.

8.5 Sparse Solutions of Linear Systems

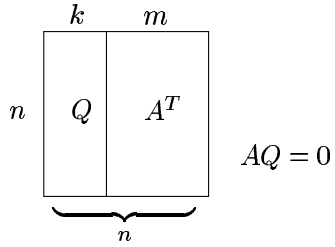
A coding problem. We begin with discussing error-correcting codes again, but this time we want to send a sequence $\mathbf{w} \in \mathbb{R}^k$ of k real numbers. Or rather, not we but a deep-space probe needs to transmit its priceless measurements

represented by $\mathbf{w}$ back to Earth. We want to make sure that all components of $\mathbf{w}$ can be recovered correctly even if some fraction, say 8%, of the transmitted numbers are corrupted, due to random errors or even maliciously (imagine that the secret *Brotherhood For Promoting the Only Truth* can somehow tamper with the signal slightly in order to document the presence of supernatural phenomena in outer space). We admit *gross errors*; that is, if the number 3.1415 is sent and it gets corrupted, it can be received as 2152.66, or 3.1425, or -10^{11}, or any other real number.

Here is a way of encoding $\mathbf{w}$: We choose a suitable $n > k$ and a suitable $n \times k$ *encoding matrix* Q of rank k, and we send the vector $\mathbf{z} = Q\mathbf{w} \in \mathbb{R}^n$. Because of the errors, the received vector is not $\mathbf{z}$ but $\tilde{\mathbf{z}} = \mathbf{z} + \mathbf{x}$, where $\mathbf{x} \in \mathbb{R}^n$ is a vector with at most $r = \lfloor 0.08n \rfloor$ nonzero components. We ask, under what conditions on Q can $\mathbf{z}$ be recovered from $\tilde{\mathbf{z}}$?

Somewhat counterintuitively, we will concentrate on the task of finding the "error vector" $\mathbf{x}$. Indeed, once we know $\mathbf{x}$, we can compute $\mathbf{w}$ by solving the system of linear equations $Q\mathbf{w} = \mathbf{z} = \tilde{\mathbf{z}} - \mathbf{x}$. The solution, if one exists, is unique, since we assume that Q has rank k and hence the mapping $\mathbf{w} \mapsto Q\mathbf{w}$ is injective.

Sparse solutions to underdetermined linear systems. In order to compute $\mathbf{x}$, we first reformulate the recovery problem. Let $m = n - k$ and let A be an $m \times n$ matrix such that $AQ = 0$. That is, considering the k-dimensional linear subspace of $\mathbb{R}^n$ generated by the columns of Q, the rows of A form a basis of its orthogonal complement. The following picture illustrates the dimensions of the matrices:



In the recovery problem we have $\tilde{\mathbf{z}} = Q\mathbf{w} + \mathbf{x}$. Multiplying both sides by A from the left, we obtain $A\tilde{\mathbf{z}} = AQ\mathbf{w} + A\mathbf{x} = A\mathbf{x}$. Denoting $\mathbf{b} = A\tilde{\mathbf{z}}$, we thus get that the unknown $\mathbf{x}$ has to satisfy the system of linear equations $A\mathbf{x} = \mathbf{b}$. We have $m = n - k$ equations and $n > m$ unknowns; the system is *underdetermined* and it has infinitely many solutions. In general, not all of these solutions can appear as an error vector in the decoding problem (we note that the multiplication by A above is not necessarily an equivalent transformation and so it may give rise to spurious solutions). However, we seek a solution $\mathbf{x}$ with the extra property $|\mathrm{supp}(\mathbf{x})| \leq r$, where we introduce the notation

$$\mathrm{supp}(\mathbf{x}) = \{i \in \{1, 2, \ldots, n\} : x_i \neq 0\}.$$

As we will see, under suitable conditions relating n, m, r and A, such a *sparse* solution to $A\mathbf{x} = \mathbf{b}$ turns out to be unique (and thus it has to be the desired error vector), and it can be computed efficiently by linear programming!

Let us summarize the resulting problem once again:

Sparse solution to underdetermined system of linear equations

Given an $m\times n$ matrix A with $m < n$, a vector $\mathbf{b} \in \mathbb{R}^m$, and an integer r, find an $\mathbf{x} \in \mathbb{R}^n$ such that

$$A\mathbf{x} = \mathbf{b} \quad \text{and} \quad |\text{supp}(\mathbf{x})| \le r \tag{8.14}$$

if one exists.

The coding problem above is only one among several important practical problems leading to the computation of sparse solutions to underdetermined systems. We will mention other applications at the end of this section. From now on, we call any $\mathbf{x}$ satisfying (8.14) a *sparse solution* to $A\mathbf{x} = \mathbf{b}$. (Warning: this shouldn't be confused with solutions to *sparse systems* of equations, which is an even more popular topic in numerical mathematics and scientific computing.)

A linear algebra view. There is a simple necessary and sufficient condition guaranteeing that there is at most one sparse solution to $A\mathbf{x} = \mathbf{b}$.

8.5.1 Observation. *With n,m,r fixed, the following two conditions on an $m\times n$ matrix A are equivalent:*

(i) *The system $A\mathbf{x} = \mathbf{b}$ has at most one sparse solution $\mathbf{x}$ for every $\mathbf{b}$.*

(ii) *Every $2r$ or fewer columns of A are linearly independent.*

Proof. To prove the (more interesting) implication (ii)⇒(i), let us assume that $\mathbf{x}'$ and $\mathbf{x}''$ are two different sparse solutions to $A\mathbf{x} = \mathbf{b}$. Then $\mathbf{y} = \mathbf{x}' - \mathbf{x}'' \neq \mathbf{0}$ has at most $2r$ nonzero components and satisfies $A\mathbf{y} = A\mathbf{x}' - A\mathbf{x}'' = \mathbf{0}$, and hence it defines a nontrivial linear dependence of at most $2r$ columns of A.

To prove (i)⇒(ii), we essentially reverse the above argument. Supposing that there exists nonzero $\mathbf{y} \in \mathbb{R}^n$ with $A\mathbf{y} = \mathbf{0}$ and $|\text{supp}(\mathbf{y})| \le 2r$, we write $\mathbf{y} = \mathbf{x}' - \mathbf{x}''$, where both $\mathbf{x}'$ and $\mathbf{x}''$ have at most r nonzero components. For example, $\mathbf{x}'$ may agree with $\mathbf{y}$ in the first r nonzero components and have 0's elsewhere, and $\mathbf{x}'' = \mathbf{x}' - \mathbf{y}$ has the remaining at most r nonzero components of $\mathbf{y}$ with opposite sign. We set $\mathbf{b} = A\mathbf{x}'$, so that $\mathbf{x}'$ is a sparse solution to $A\mathbf{x} = \mathbf{b}$, and we note that $\mathbf{x}''$ is another sparse solution since $A\mathbf{x}'' = A\mathbf{x}' - A\mathbf{y} = A\mathbf{x}' = \mathbf{b}$. □

Let us note that (ii) implies that, in particular, $m \ge 2r$. On the other hand, if we choose a "random" $2r\times n$ matrix A, we almost surely have every $2r$ columns linearly independent.[3] So in the coding problem, if we set n so that $n = k + 2r$, choose A random, and let the columns of Q form a basis of

[3] In this book we don't want to assume or introduce the knowledge required to state and prove this claim rigorously. Instead, we offer the following semiformal argument relying on a famous but nontrivial theorem. The condition of linear independence of every $2r$ columns can be reformulated as $\det(A_I) \neq 0$ for every $2r$-element $I \subset \{1,2,\dots,n\}$. Now for I fixed, $\det(A_I)$ is a polynomial of degree $2r$ in the $2rn$ entries of A (it really depends only on $4r^2$ entries but never mind), and the set of the matrices A with $\det(A_I) = 0$ is the zero set of this polynomial in $\mathbb{R}^{2rn}$. The zero set of any nonzero polynomial is very "thin"; by Sard's

the orthogonal complement of the row space of A, we may seem to be done—random A has almost surely every $2r$ columns linearly independent, and in such case, assuming that no more than r errors occurred, the sparse error vector $\mathbf{x}$ is always determined uniquely, and so is the original message $\mathbf{w}$.

Efficiency? But a major question remains—how can we *find* the unknown sparse solution $\mathbf{x}$? Unfortunately, it turns out that the problem of computing a sparse solution to $A\mathbf{x} = \mathbf{b}$ is difficult (NP-hard) in general, even for A satisfying the conditions of Observation 8.5.1.

Since the problem of finding a sparse solution to $A\mathbf{x} = \mathbf{b}$ is important and computationally difficult, several heuristic methods have been proposed for solving it at least approximately and at least in some cases. One of them, described next, turned out to be considerably more powerful than others.

Basis pursuit. A sparse solution $\mathbf{x}$ is "small" in the sense of having few nonzero components. The idea is to look for $\mathbf{x}$ that is "small" in another sense that is easier to deal with, namely, with small $|x_1| + |x_2| + \cdots + |x_n|$. The last quantity is commonly denoted by $\|\mathbf{x}\|_1$ and called the **ℓ_1-norm** of $\mathbf{x}$ (while $\|\mathbf{x}\| = \|\mathbf{x}\|_2 = \sqrt{x_1^2 + \cdots + x_n^2}$ is the usual Euclidean norm, which can also be called the ℓ_2-norm).[4] We thus arrive at the following optimization problem (usually called *basis pursuit* in the literature):

$$\text{minimize } \|\mathbf{x}\|_1 \text{ subject to } \mathbf{x} \in \mathbb{R}^n \text{ and } A\mathbf{x} = \mathbf{b}. \tag{BP}$$

By a trick we have learned in Section 2.4, this problem can be reformulated as a linear program:

$$\begin{array}{ll} \text{minimize} & u_1 + u_2 + \cdots + u_n \\ \text{subject to} & A\mathbf{x} = \mathbf{b} \\ & -\mathbf{u} \le \mathbf{x} \le \mathbf{u} \\ & \mathbf{x}, \mathbf{u} \in \mathbb{R}^n, \ \mathbf{u} \ge \mathbf{0}. \end{array} \tag{BP$'$}$$

To check the equivalence of (BP) and (BP$'$), we just note that in an optimal solution to (BP$'$) we have $u_i = |x_i|$ for every i.

The basis pursuit approach to finding a sparse solution of $A\mathbf{x} = \mathbf{b}$ thus consists of computing an optimal solution $\mathbf{x}^*$ to (BP) by linear programming, and hoping that, with some luck, this $\mathbf{x}^*$ might also be the sparse solution or at least close to it.

At first sight it is not clear why basis pursuit should have any chance of finding a sparse solution. After all, the desired sparse solution might have few huge components, while $\mathbf{x}^*$, a minimizer of the ℓ_1 norm, might have all components nonzero but tiny.

---

theorem, it has Lebesgue measure 0. Hence the matrices A with $\det(A_I) = 0$ for at least one I correspond to points in $\mathbb{R}^{2nr}$ lying on the union of $\binom{n}{2r}$ zero sets, each of measure 0, and altogether they have measure 0. Therefore, such matrices appear with zero probability in any "reasonable" continuous distribution on matrices, for example, if the entries of A are chosen independently and uniformly from the interval $[-1, 1]$.

[4]The letter ℓ here can be traced back to the surname of Henri Lebesgue, the founder of modern integration theory. A certain space of integrable functions on $[0, 1]$ is denoted by $L_1(0, 1)$ in his honor, and ℓ_1 is a "pocket version" of this space consisting of countable sequences instead of functions.

Surprisingly, experiments revealed that basis pursuit actually performs excellently, and it usually finds the sparse solution exactly even in conditions that don't look very favorable. Later these findings were confirmed by rather precise theoretical results. Here we consider the following particular case of such results:

8.5.2 Theorem (Guaranteed success of basis pursuit). *Let*

$$m = \lfloor 0.75n \rfloor,$$

and let A be a random $m \times n$ matrix, where each entry is drawn from the standard normal distribution $N(0,1)$ and the entries are mutually independent.[5] *Then with probability at least $1 - e^{-cm}$, where $c > 0$ is a positive constant, the matrix A has the following property:*

> *If $\mathbf{b} \in \mathbb{R}^m$ is such that the system $A\mathbf{x} = \mathbf{b}$ has a solution $\tilde{\mathbf{x}}$ with at most $r = \lfloor 0.08n \rfloor$ nonzero components, then $\tilde{\mathbf{x}}$ is a unique optimum solution to (BP).*

For brevity, we call a matrix A with the property as in the theorem **BP-exact** (more precisely, we should say "BP-exact for r," where r specifies the maximum number of nonzero components). For a BP-exact matrix A we can thus find a sparse solution to $A\mathbf{x} = \mathbf{b}$ exactly and efficiently, by solving the linear program (BP′).

Returning to the coding problem from the beginning of the section, we immediately obtain:

8.5.3 Corollary. *Let k be a sufficiently large integer, let us set $n = 4k$, $m = 3k$, let a random $m \times n$ matrix A be generated as in Theorem 8.5.2, and let Q be an $n \times k$ matrix of rank k with $AQ = 0$ (in other words, the column space of Q is the orthogonal complement of the row space of A). Then the following holds with probability overwhelmingly close to 1: If Q is used as a coding matrix to transmit a vector $\mathbf{w} \in \mathbb{R}^k$, by sending the vector $\mathbf{z} = Q\mathbf{w} \in \mathbb{R}^n$, then even if any at most 8% of the entries of $\mathbf{z}$ are corrupted, we can still reconstruct $\mathbf{w}$ exactly and efficiently, by solving the appropriate instance of (BP′).*

Drawing the elements of A from the standard normal distribution is not the only known way of generating a BP-exact matrix. Results similar to Theorem 8.5.2, perhaps with worse constants, can be proved by known techniques for random matrices with other distributions. The perhaps simplest such distribution is obtained by choosing each entry to be $+1$ or -1, each with probability $\frac{1}{2}$ (and again with all entries mutually independent).

[5] We recall that the distribution $N(0,1)$ has density given by the Gauss "bell curve" $\frac{1}{\sqrt{2\pi}} e^{-x^2/2}$. How can we generate a random number with this distribution? This is implemented in many software packages, and methods for doing it can be found, for instance, in

D. Knuth: *The Art of Computer Programming*, Vol. 2: Seminumerical Algorithms, Addison-Wesley, Reading, Massachusetts, 1973.

A somewhat unpleasant feature of Theorem 8.5.2 and of similar results is that they provide a BP-exact matrix only with high probability. No efficient method for *verifying* that a given matrix is BP-exact is known at present, and so we cannot be absolutely sure. In practice this is not really a problem, since the probability of failure (i.e., of generating a matrix that is not BP-exact) can be made very small by choosing the parameters appropriately, much smaller than the probability of sudden death of all people in the team that wants to compute the sparse solution, for instance. Still, it would be nice to have explicit constructions of BP-exact matrices with good parameters.

Random errors. In our coding problem, we allow for completely arbitrary (even malicious) errors; all we need is that there aren't too many errors. However, in practice one may often assume that the errors occur at random positions, and we want to be able to decode correctly only with high probability, that is, for most (say 99.999%) of the $\binom{n}{r}$ possible locations of the r errors. It turns out that considerably stronger numerical bounds can be obtained in this setting: For example, Theorem 8.5.2 tells us that for $m = \lfloor 0.75n \rfloor$ and $k = n - r$, we are guaranteed to fix *any* $0.08n$ errors, but it turns out that we can also fix *most* of the possible r-tuples of errors for r as large as $0.36n$! For a precise statement see the paper of Donoho quoted near the end of the section.

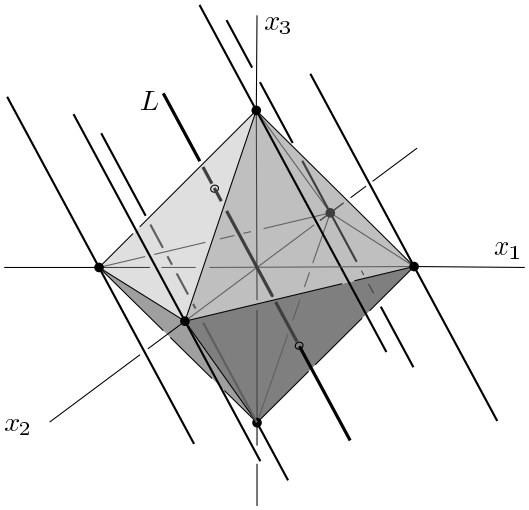
Geometric meaning of BP-exactness. Known proofs of Theorem 8.5.2 or similar results use a good deal of geometric probability and high-dimensional geometry, knowledge which we want neither to assume nor to introduce in this book. We thus have to omit a proof. Instead, we present geometric characterizations of BP-exact matrices that are starting points of existing proofs.

We begin with a straightforward geometric reformulation of the definition of BP-exactness. For its statement we need to recall the *crosspolytope*, a convex polytope already mentioned in Section 4.3. We will denote the n-dimensional crosspolytope by B_1^n, which should suggest that it is the unit ball of the ℓ_1-norm:

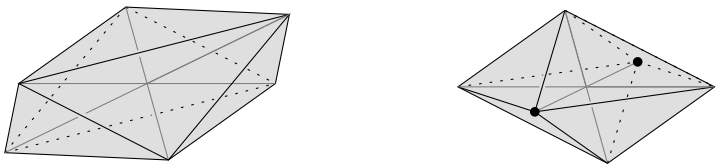
$$B_1^n = \{\mathbf{x} \in \mathbb{R}^n : ||\mathbf{x}||_1 \leq 1\}.$$

8.5.4 Lemma (Reformulation of BP-exactness). *Let A be an $m \times n$ matrix, $m < n$, let $r \leq m$, and let $L = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{0}\}$ be the kernel (null space) of A. Then A is BP-exact for r if and only if the following holds: For every $\mathbf{z} \in \mathbb{R}^n$ with $||\mathbf{z}||_1 = 1$ (i.e., $\mathbf{z}$ is a boundary point of the crosspolytope) and $|\mathrm{supp}(\mathbf{z})| \leq r$, the affine subspace $L + \mathbf{z}$ intersects the crosspolytope only at $\mathbf{z}$; that is, $(L + \mathbf{z}) \cap B_1^n = \{\mathbf{z}\}$.*

Let us discuss an example with $n = 3$, $m = 2$, and $r = 1$, about the only sensible setting for which one can draw a picture. If the considered 2×3 matrix A has full rank, which we may assume, then the kernel L is a one-dimensional linear subspace of $\mathbb{R}^3$, that is, a line passing through the origin. The points $\mathbf{z}$ coming into consideration have at most $r = 1$ nonzero coordinate, and they lie on the boundary of the regular octahedron B_1^3, and hence they are precisely the 6 vertices of B_1^3:



The line L through the origin is drawn thick, and the condition in the lemma says that each of the 6 translates of L to the vertices should only touch the crosspolytope. Another way of visualizing this is to look at the projection of B_1^3 to the plane perpendicular to L. Each of the translates of L is projected to a point, and the crosspolytope is projected to a convex polygon. The condition then means that all the 6 vertices should appear on the boundary in the projection, as in the left picture below,



while the right picture corresponds to a bad L (the condition is violated at the two vertices marked by dots that lie inside the projection). In general, of course, L is not a line but a k-dimensional linear subspace of $\mathbb{R}^n$, and the considered points $\mathbf{z}$ are not only vertices of B_1^n, but they can lie in all $(r-1)$-dimensional faces of B_1^n. Indeed, we note that the points $\mathbf{z}$ on the surface of B_1^n with at most r nonzero components are exactly the points of the union of all $(r-1)$-dimensional faces, omitting the easy proof of this fact (but look at least at the case $n = 3$, $r = 2$).

Proof of Lemma 8.5.4. First we assume that A is BP-exact, we consider a $\mathbf{z}$ with $\|\mathbf{z}\|_1 = 1$ and $|\mathrm{supp}(\mathbf{z})| \le r$, and we set $\mathbf{b} = A\mathbf{z}$. Then the system $A\mathbf{x} = \mathbf{b}$ has a sparse solution, namely $\mathbf{z}$, and hence $\mathbf{z}$ has to be the unique point among all solutions to $A\mathbf{x} = \mathbf{b}$ minimizing the ℓ_1-norm. Noting that the set of all solutions to $A\mathbf{x} = \mathbf{b}$ is exactly the affine subspace $L + \mathbf{z}$, we get that $\mathbf{z}$ is the only point in $L+\mathbf{z}$ with ℓ_1-norm at most 1. That is, $(L+\mathbf{z}) \cap B_1^n = \{\mathbf{z}\}$ as claimed.

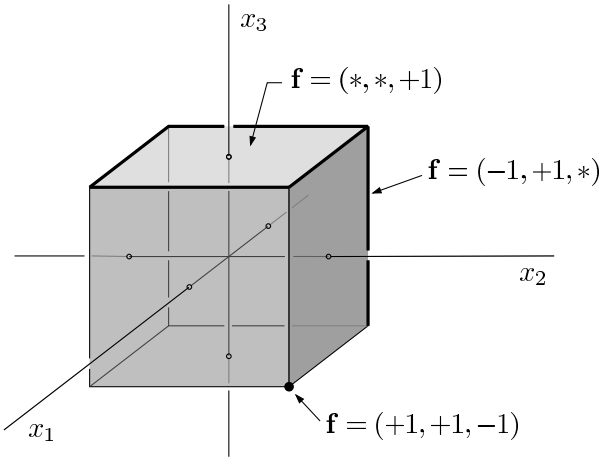
Conversely, we assume that L satisfies the condition in the lemma and we consider a $\mathbf{b} \in \mathbb{R}^m$. Let us suppose that the system $A\mathbf{x} = \mathbf{b}$ has a solution $\tilde{\mathbf{x}}$ with at most r nonzero components. If $\tilde{\mathbf{x}} = \mathbf{0}$, then $\mathbf{b} = \mathbf{0}$, and clearly, $\mathbf{0}$ is

also the only optimum of (BP). For $\tilde{\mathbf{x}} \neq \mathbf{0}$, we set $\mathbf{z} = \frac{\tilde{\mathbf{x}}}{\|\tilde{\mathbf{x}}\|_1}$. Then $\|\mathbf{z}\| = 1$ and $|\mathrm{supp}(\mathbf{z})| \leq r$, and so by the assumption, $\mathbf{z}$ is the only point in $L+\mathbf{z}$ of ℓ_1-norm at most 1. By re-scaling we get that $\tilde{\mathbf{x}}$ is the only point in $L+\tilde{\mathbf{x}}$ of ℓ_1-norm at most $\|\tilde{\mathbf{x}}\|_1$, and since $L+\tilde{\mathbf{x}}$ is the set of all solutions to $A\mathbf{x} = \mathbf{b}$, we get that A is BP-exact. □

Another nice geometric interpretation of BP-exactness speaks about the n-dimensional cube $[-1,1]^n$ and about its faces. We recall, again without proof, that the faces of the n-dimensional cube exactly correspond to vectors $\mathbf{f} \in \{-1,+1,*\}$. The face described by a particular vector $\mathbf{f}$ is the point set

$$\{\mathbf{x} \in [-1,1]^n : x_i = f_i \text{ for all } i \text{ with } f_i \neq *\}.$$

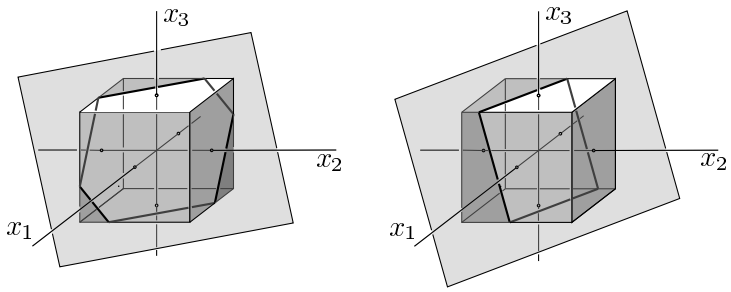
The dimension k of this face equals the number of $*$'s in $\mathbf{f}$, and geometrically, the face is a copy of the k-dimensional cube. This is illustrated below for the familiar case $n = 3$, for three marked faces:



For simplicity, we will state an equivalent condition for BP-exactness only for matrices A such that the problem (BP), i.e., the minimization of $\|\mathbf{x}\|_1$ subject to $A\mathbf{x} = \mathbf{b}$, has a unique optimum for every $\mathbf{b}$. It turns out that for random matrices A one need not worry about this condition at all, since it is a "general position" condition, which is satisfied with probability 1 (for matrices drawn from a "reasonable" continuous distribution). A proof is similar to the one outlined in a footnote after Observation 8.5.1. It begins with the observation that if the minimum of $\|\mathbf{x}\|_1$ subject to $A\mathbf{x} = \mathbf{b}$ is *not* unique, then the linear subspace $L = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{0}\}$ has to have a line (1-dimensional subspace) in common with one of the 2^n hyperplanes of the form $\mathbf{q}^T\mathbf{x} = 0$, with $\mathbf{q} \in \{-1,+1\}^n$, and for a random A this happens with probability 0.

8.5.5 Proposition (Another geometric view of BP-exactness). *Let $m < n$ and let A be an $m \times n$ matrix of rank m such that for every $\mathbf{b} \in \mathbb{R}^m$ the optimization problem (BP) has at most one optimal solution. Then A is BP-exact with parameter r if and only if the m-dimensional linear subspace R of $\mathbb{R}^n$ generated by the rows of A intersects each of the $(n-r)$-dimensional faces of the n-dimensional cube $[-1,1]^n$.*

We remark that R can also be described as $L^\perp$, the orthogonal complement of the subspace L considered in Lemma 8.5.4. In the situation of our earlier example with $n = 3$, $m = 2$, and $r = 1$, R is a 2-dimensional subspace of $\mathbb{R}^3$, that is, a plane through the origin, and the condition says that this plane should intersect all 2-dimensional faces of the 3-cube. There are 6 such faces, and thus the slice $R \cap [-1,1]^3$ should be a hexagon. The next picture shows a "good" case on the left and a "bad" case on the right.



Proof of Proposition 8.5.5. One possible proof is geometric, based on separation of convex sets with disjoint interiors by a hyperplane. We present another proof, which is a straightforward, although not effortless, application of the Duality Theorem of linear programming.

First we formulate a linear program dual to (BP′) using the recipe in Section 6.2. We recall that (BP′) asks for minimizing $\sum_{i=1}^n u_i$ subject to $A\mathbf{x} = \mathbf{b}$, $-\mathbf{u} \le \mathbf{x} \le \mathbf{u}$, and $\mathbf{u} \ge \mathbf{0}$. We rearrange the constrains $-u_i \le x_i$ to $x_i + u_i \ge 0$, and $x_i \le u_i$ to $-x_i + u_i \ge 0$. We then have the following scheme of the linear program:

	x_1 … x_n	u_1 … u_n		
y_1 … y_m	A	0	$=$	$\mathbf{b}$
v_1'	$+1$	$+1$		
v_1''	-1	$+1$		
	⋮	⋮	$\ge$	$\mathbf{0}$
v_n'	$+1$	$+1$		
v_n''	-1	$+1$		

$\mathbf{0}$	1 … 1

objective function

There are $2n$ variables and $m + 2n$ constraints (not counting the nonnegativity constraints $\mathbf{u} \ge \mathbf{0}$), and so the matrix has $m + 2n$ rows and $2n$ columns. Thus the dual linear program has $m + 2n$ variables, which we denote by $y_1, \ldots, y_m$ and $v_1', v_1'', \ldots, v_n', v_n''$, as indicated on the left-hand side of the picture, and it

goes as follows:

$$\begin{array}{ll} \text{maximize} & \mathbf{b}^T\mathbf{y} \\ \text{subject to} & A^T\mathbf{y} + \mathbf{v}' - \mathbf{v}'' = \mathbf{0} \\ & \mathbf{v}' + \mathbf{v}'' \leq \mathbf{1} \\ & \mathbf{v}', \mathbf{v}'' \geq \mathbf{0} \end{array}$$

(with $\mathbf{1}$ denoting the vector of all 1's). Next, we observe that under the constraints $v_i' \geq 0$, $v_i'' \geq 0$, and $v_i' + v_i'' \leq 1$, the difference $v_i' - v_i''$ can attain all values in $[-1, 1]$. Therefore, an equivalent form of the dual linear program is

$$\text{maximize } \mathbf{b}^T\mathbf{y} \text{ subject to } -\mathbf{1} \leq A^T\mathbf{y} \leq \mathbf{1}. \tag{DBP}$$

Now we are ready to prove the equivalence in the proposition.

First, we assume that A is BP-exact, and we prove that the row space of A intersects every $(n-r)$-dimensional face of the n-dimensional cube. Let us fix such a face F, specified by a vector $\mathbf{f} \in \{-1, +1, *\}$. Let us choose $\tilde{\mathbf{x}} \in F$ with $\tilde{x}_i = f_i$ for $f_i = \pm 1$ and $\tilde{x}_i = 0$ for $f_i = *$, and let us set $\mathbf{b} = A\tilde{\mathbf{x}}$. Since $\tilde{\mathbf{x}}$ has exactly r nonzero components, it is an optimal solution of (BP) by the assumptions (the unique optimum, in fact, but we don't need this here). So by the Duality Theorem, the linear program (DBP) has an optimal solution $\tilde{\mathbf{y}}$ for which the value of the objective function equals that in (BP); that is, $\mathbf{b}^T\tilde{\mathbf{y}} = \|\tilde{\mathbf{x}}\|_1$. We rewrite the left-hand side: $\mathbf{b}^T\tilde{\mathbf{y}} = \tilde{\mathbf{x}}^T A^T\tilde{\mathbf{y}} = \tilde{\mathbf{x}}^T\mathbf{v}$, where we put $\mathbf{v} = A^T\tilde{\mathbf{y}}$. Clearly, $\mathbf{v}$ is in the row space of A, and it remains to check that it lies in F. We have $-\mathbf{1} \leq \mathbf{v} \leq \mathbf{1}$ since $\tilde{\mathbf{y}}$ is a feasible solution of (DBP). We note that for *any* vector $\mathbf{v}$ with $-\mathbf{1} \leq \mathbf{v} \leq \mathbf{1}$ we have $\mathbf{v}^T\tilde{\mathbf{x}} \leq \sum |\tilde{x}_i| = \|\tilde{\mathbf{x}}\|_1$, with equality only if $v_i = \operatorname{sign}(\tilde{x}_i)$ for all i with $\tilde{x}_i \neq 0$. This means that $\mathbf{v} \in F$ as desired.

To prove the converse implication in the proposition, we essentially reverse the same procedure. We assume that the row space R of A intersects all $(n-r)$-dimensional faces of the n-cube. Given $\mathbf{b}$ such that $A\mathbf{x} = \mathbf{b}$ has a sparse solution $\tilde{\mathbf{x}}$ with at most r-nonzero components, we want to check that $\tilde{\mathbf{x}}$ is an optimum of (BP). By the Duality Theorem of linear programming (this time weak duality suffices!), it is enough to exhibit a feasible solution $\tilde{\mathbf{y}}$ to the dual linear program (DBP) with the same value of the objective function, i.e., with $\mathbf{b}^T\tilde{\mathbf{y}} = \|\tilde{\mathbf{x}}\|_1$. Let us set $f_i = *$ if $\tilde{x}_i = 0$ and $f_i = \operatorname{sign}(\tilde{x}_i)$ otherwise, and let us consider the $(n-r)$-dimensional face F of the cube described by $\mathbf{f}$ (more precisely, if $\tilde{x}$ has fewer than r nonzero components, we change some $*$'s in $\mathbf{f}$ to 1's, say, so that $\mathbf{f}$ has precisely $n-r$ $*$'s). By the assumption, the row space of A intersects F at some point $\mathbf{v}$, which means that there exists $\tilde{\mathbf{y}} \in \mathbb{R}^m$ such that $A^T\tilde{\mathbf{y}} = \mathbf{v}$. Since $-\mathbf{1} \leq \mathbf{v} \leq \mathbf{1}$, $\tilde{\mathbf{y}}$ is a feasible solution to (DBP). We note that $\mathbf{v} \in F$ implies that $v_i = \operatorname{sign}(\tilde{x}_i)$ whenever $\tilde{x}_i \neq 0$, and so for the value of the objective function we obtain $\mathbf{b}^T\tilde{\mathbf{y}} = \tilde{\mathbf{x}}^T A^T\tilde{\mathbf{y}} = \tilde{\mathbf{x}}^T\mathbf{v} = \sum_{i=1}^n \operatorname{sign}(\tilde{x}_i) \cdot \tilde{x}_i = \|\tilde{\mathbf{x}}\|_1$. Proposition 8.5.5 is proved. □

Intuition for BP-exactness. We don't have means for proving Theorem 8.5.2, but now, using the proposition just proved, we can at least try to convey some intuition on why a claim like Theorem 8.5.2 is plausible, and what kind of calculations are needed to prove it.

A random matrix A defines a random m-dimensional subspace[6] R, and for proving Theorem 8.5.2, we need to verify that R is very likely to intersect every $(n-r)$-dimensional face of the cube. The first key point is that it suffices to show that R intersects a *fixed* $(n-r)$-dimensional face with probability at least $1-p$ with p much smaller than $\left(2^r\binom{n}{r}\right)^{-1}$. Indeed, then the probability that R fails to intersect any of the $2^r\binom{n}{r}$ faces of dimension $(n-r)$ is much smaller than 1 as needed.

We thus need to bound above the probability that R misses a fixed $(n-r)$-dimensional face, say the face $F_0 = \{\mathbf{x} \in [-1,1]^n : x_1 = x_2 = \cdots = x_r = 1\}$. This is an $(n-r)$-dimensional cube, with center at distance $\sqrt{r}$ from the origin. Now a high-dimensional cube is quite a large body: For the considered dimension $n-r$ it has diameter $2\sqrt{n-r}$. So it shouldn't be entirely surprising that a random m-dimensional subspace R is extremely unlikely to miss F_0 if n, m, r are in appropriate relations. In a real proof this has to be verified by calculations, of course.

References. Basis pursuit was introduced in

> S. Chen, D. L. Donoho, and M. A. Saunders: Atomic decomposition by basis pursuit, *SIAM J. Scientific Computing* 20,1(1999) 33–61.

To understand the term basis pursuit, one should keep in mind that a classical approach to finding a "good" solution to an underdetermined system $A\mathbf{x} = \mathbf{b}$ would be to minimize $||\mathbf{x}||_2$, rather than $||\mathbf{x}||_1$ (a "least squares" or "generalized inverse" method), which typically yields a solution with many nonzero components and is much less successful in applications such as the decoding problem. In minimizing the ℓ_1-norm instead, one obtains a basic solution with only few nonzero components.

Interestingly, several groups of researchers independently arrived at the concept of BP-exactness and obtained the following general version of Theorem 8.5.2: *For every constant $\alpha \in (0,1)$ there exists $\beta = \beta(\alpha) > 0$ such that a random $\lfloor \alpha n \rfloor \times n$ matrix A is BP-exact for $r = \lfloor \beta n \rfloor$ with probability exponentially close to* 1. Combined results of two of these groups can be found in

> E. J. Candès, M. Rudelson, T. Tao, and R. Vershynin: Error correction via linear programming, *Proc. 46th IEEE Symposium on Foundations of Computer Science* (FOCS), 2005, pages 295–308.

A third independent proof was given by Donoho. Later, and by yet another method, he obtained the strongest known quantitative bounds, including those in Theorem 8.5.2 (the previously mentioned proofs yield much smaller constant for $\alpha = 0.75$ than 0.08). Among his several papers on the subject we quote

> D. Donoho: Neighborly polytopes and sparse solutions of underdetermined linear equations, preprint, 2005,

where he establishes a connection to the classical theory of convex polytopes using a result in the spirit of Lemma 8.5.4 and the remarks following it (but more elaborate). Through this connection he obtained an interesting *upper* bound on $\beta(\alpha)$.

[6] The question "What is a random m-dimensional subspace?" doesn't lack delicacy. For us, the simplest way out is to define a random m-dimensional subspace as the row space of a random $m \times n$ matrix with independent normal entries as in Theorem 8.5.2. Fortunately, this turns out to be equivalent to the usual (and "right") definition, which is as follows. One fixes a particular m-dimensional subspace R_0, say the span of the first m vectors of the standard basis, and defines a random subspace as a random rotation of R_0. This may not sound like a great progress, since we have just used the equally problematic-looking notion of random rotation. But the group $\mathrm{SO}(n)$ of all rotations in $\mathbb{R}^n$ around the origin is a compact group and hence it has a unique invariant probability measure (Haar measure), which defines "random rotation" satisfactorily and uniquely.

For example, in the setting of Theorem 8.5.2, there exists *no* $\lfloor 0.75n \rfloor \times n$ BP-exact matrix at all with $r > 0.25n$ (assuming n large).

We remark that our notation is a compromise among the notations of the papers quoted above and doesn't follow either of them exactly, and that the term "BP-exact" is ours.

More applications of sparse solutions to underdetermined systems. The problem of computing a sparse solution to a system of linear equations arises in *signal processing*. The signal considered may be a recording of a sound, a measurement of seismic waves, a picture taken by a digital camera, and many other things. A classical method of analyzing signals is *Fourier analysis*, which from linear-algebraic point of view means expressing a given periodic function in a basis consisting of the functions $1, \cos x, \sin x, \cos 2x, \sin 2x, \cos 3x, \sin 3x, \ldots$ (the closely related *cosine transform* is used in the *JPEG encoding* of digital pictures). These functions are linearly independent, and so the expression (Fourier series) is unique. In the more recent *wavelet* analysis[7] one typically has a larger collection of basic functions, the wavelets, which can be of various kinds, depending on the particular application. They are no longer linearly independent, and hence there are many different ways of representing a given signal as a linear combination of wavelets. So one looks for a representation satisfying some additional criteria, and sparsity (small number of nonzero coefficients) is a very natural criterion: It leads to an economic (compressed) representation, and sometimes it may also help in analyzing or filtering the signal. For example, let us imagine that there is a smooth signal that has a nice representation by sine and cosine functions, and then an impulsive noise made of "spike" functions is added to it. We let the basic functions be sines and cosines *and* suitable spike functions, and by computing a sparse representation in such a basis we can often isolate the noise component very well, something which the classical Fourier analysis cannot do. Thus, we naturally arrive at computing sparse solutions to underdetermined linear systems.

Another source is computer tomography, where one has an unknown vector $\mathbf{x}$ (each x_i is the density of some small area of tissue, say), and the CT scanner measures various linear combinations of the x_i, corresponding to rays through the tissue in various directions. Sometimes there are reasons to expect that only a small number of the pixels will have values x_i different from the background level, and when we want to reconstruct $\mathbf{x}$ from the scan, we again ask for a sparse solution of a linear system. (More realistically, although less intuitively, we don't expect a small number of nonzero *pixels*, but rather a small number of significantly nonzero coefficients in a suitable wavelet representation.)

8.6 Transversals of *d*-Intervals

This section describes an application of the duality theorem in discrete geometry and combinatorics. We begin with a particular geometric result. Then we discuss concepts appearing in the proof in a more general context.

Helly-type and Gallai-type theorems. First let $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ be a family of closed intervals on the real line such that every two of the intervals intersect. It is easily seen that there exists a point common to all of the intervals in $\mathcal{I}$: Indeed, the rightmost among the left endpoints of the I_i is such a point. In more detail, writing $I_i = [a_i, b_i]$ and setting $a = \max\{a_1, \ldots, a_n\}$, we necessarily have $a_i \le a \le b_i$ for all i, since $a_i \le a$ is immediate from the definition of a,

[7] Indeed, the newer picture encoding standard JPEG 2000 employs wavelets.

and if we had $a > b_i$ for some i, then $I_i = [a_i, b_i]$ would be disjoint from the interval beginning with a.

The statement just proved is a special (one-dimensional) case of a beautiful and important theorem of Helly: *If $C_1, C_2, \ldots, C_n$ are convex sets in $\mathbb{R}^d$ such that any at most $d+1$ of them have a point in common, then there is a point common to all of the C_i.* We will not prove this result; it is mentioned as a background against which the forthcoming results can be better appreciated.

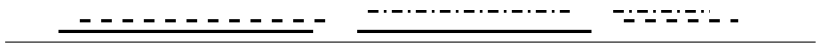
It is easily seen that in general we cannot replace $d+1$ by any smaller number in Helly's theorem. For example, in the plane, the assumption of Helly's theorem requires every three of the sets to have a common point, and pairwise intersections are not enough. To see this, we consider n lines in general position. They are convex sets, every two of them intersect, but no three have a common point.

Let us now consider planar convex sets of a special kind, namely, circular disks. One can easily draw three disks such that every two intersect, but no point is common to all three. However, there is a theorem (Gallai's) for pairwise intersecting disks in the spirit of Helly's theorem: *If $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ is a family of disks in the plane in which every two disks intersect, then there exist 4 points such that each D_i contains at least one of them.* With the (best possible) constant 4 this is a quite difficult theorem, but it is not too hard to prove a similar theorem with 4 replaced by some large constant. The reader is invited to solve this as a puzzle.

A set of points as in the theorem, that intersects every member of $\mathcal{D}$, is called a **transversal** of $\mathcal{D}$ (sometimes one also speaks of *piercing* or *stabbing* $\mathcal{D}$ by a small number of points). Thus pairwise intersecting disks in the plane always have a 4-point transversal, and Helly's theorem asserts that $(d+1)$-wise intersecting convex sets in $\mathbb{R}^d$ have a one-point transversal.

What conditions on a family of sets guarantee that it has a small transversal? This fairly general question subsumes many interesting particular problems and it has been the subject of much research. Here we consider a one-dimensional situation. At the beginning of the section we dealt with a family of intervals, and now we admit intervals with some bounded number of "holes".

Transversals for pairwise intersecting *d*-intervals. For an integer $d \geq 1$, a ***d*-interval** is defined as the union of d closed intervals on the real line. The following picture shows three pairwise intersecting 2-intervals (drawn solid, dashed, and dash-dotted, respectively) with no point common to all three:



Thus, we cannot expect a one-point transversal for pairwise intersecting d-intervals. But the following theorem shows the existence of a transversal whose size depends only on d:

8.6.1 Theorem. *Let $\mathcal{J}$ be a finite family of d-intervals such that $J_1 \cap J_2 \neq \emptyset$ for every $J_1, J_2 \in \mathcal{J}$. Then $\mathcal{J}$ has a transversal of size $2d^2$; that is, there exist $2d^2$ points so that each d-interval of $\mathcal{J}$ contains at least one of them.*

At first sight it is not obvious that there is any bound at all for the size of the transversal that depends only on d. This was first proved in 1970, with a bound exponential in d, in

> A. Gyárfás, J. Lehel: A Helly-type problem in trees, in *Combinatorial Theory and its Applications*, P. Erdős, A. Rényi, and V.T. Sós, editors, North-Holland, Amsterdam, 1970, pages 571–584.

The best bound known at present is d^2, and it has been established using algebraic topology in

> T. Kaiser: Transversals of d-intervals, *Discrete Comput. Geom.* 18(1997), 195–203.

We are going to prove a bound that is worse by a factor of 2, but the proof presented here is much simpler. It comes from

> N. Alon: Piercing d-intervals, *Discrete Comput. Geom.* 19(1998) 333–334,

following a general method developed in

> N. Alon, D. Kleitman: Piercing convex sets and the Hadwiger–Debrunner (p,q)-problem, *Adv. Math.* 96(1992) 103–112.

It is also known that the bound cannot be improved below a constant multiple of $\frac{d^2}{\log d}$; see

> J. Matoušek: Lower bounds on the transversal numbers of d-intervals, *Discrete Comput. Geom.* 26(2001) 283–287.

Working towards a proof of the theorem, we first show that in a family of pairwise intersecting d-intervals, some point has to be contained in "many" members of the family. For reasons that will become apparent later, we formulate it for a finite *sequence* of d-intervals, so that repetitions are allowed.

8.6.2 Lemma. *Let $J_1, J_2, \ldots, J_n$ be d-intervals such that $J_i \cap J_j \neq \emptyset$ for all $i,j \in \{1,2,\ldots,n\}$. Then there is an endpoint of some J_i that is contained in at least $n/2d$ of the d-intervals.*

Proof. First let us assume that no two d-intervals share an endpoint. For every i,j we have $J_i \cap J_j \neq \emptyset$, and this implies that an endpoint of J_i is contained in J_j or an endpoint of J_j is contained in J_i (or both). Thus, there are at least $\binom{n}{2} + n \geq n^2/2$ ordered pairs (p,j) such that p is an endpoint of J_i and it is contained in J_j (we note that by the assumption, p determines i uniquely, and so no pair (p,j) is counted more than once). Since there are at most $2nd$ endpoints, some endpoint has to be contained in at least $(n^2/2)/2nd = n/4d$ of the considered d-intervals.

This simple argument has two shortcomings: It works only if all endpoints are distinct, and it yields only half of the bound claimed in the lemma. The following "real" proof of the lemma is a little more complicated but addresses these two shortcomings.

Let T be the set of all ordered triples (p,i,j) such that p is one of the at most d *left* endpoints of J_i, and $p \in J_j$. We want to bound the size of T from below. Let us fix $i \leq j$ for a moment. Let p be the leftmost point of the (nonempty) intersection $J_i \cap J_j$. Clearly, p is a left endpoint of J_i or J_j, and

thus T contains one of the triples (p,i,j) or (p,j,i). So every pair i,j, $i \le j$, contributes at least one member of T, and thus $|T| \ge \binom{n}{2} + n \ge n^2/2$. Since there are at most dn possible values for the pair (p,i), it follows that there exist p_0 and i_0 such that $(p_0,i_0,j) \in T$ for at least $n/2d$ values of j. This means that p_0 is contained in at least $n/2d$ of the J_j. □

Next, we show that for every family of pairwise intersecting d-intervals we can distribute weights on the endpoints so that every d-interval has relatively large weight (compared to the total weight of all points).

8.6.3 Lemma. *Let $\mathcal{J}$ be a finite family of pairwise intersecting d-intervals, and let P denote the set of endpoints of the d-intervals in $\mathcal{J}$. Then there are nonnegative real numbers x_p, $p \in P$, such that*

$$\begin{aligned} \sum_{p \in J \cap P} x_p &\ge 1 \text{ for every } J \in \mathcal{J}, \text{ and} \\ \sum_{p \in P} x_p &\le 2d. \end{aligned}$$

Before proving the lemma, let us see how it implies Theorem 8.6.1. Given $\mathcal{J}$ and the weights as in the lemma, we want to choose a set $X \subseteq P$ such that $|X| \le 2d^2$, and each of the $|X|+1$ open intervals into which X divides the real axis contains points of P of total weight less than $1/d$. Such an X can be selected from P by a simple left-to-right scan: Let $p_1 < p_2 < \cdots < p_m$ be the points of P. We consider them one by one in this order, and we include p_i into X whenever $\sum_{j: p_\ell < p_j \le p_i} x_{p_j} \ge \frac{1}{d}$, where p_ℓ is the last point already included in X (and we formally put $p_\ell = -\infty$ if no point has yet been included in X). It is clear that none of the open intervals determined by X contains weight $1/d$ or larger, and the bound on the size of X follows easily, using that the total weight of all points of P is at most $2d$.

We claim that X is a transversal of $\mathcal{J}$. Indeed, considering a $J \in \mathcal{J}$, at least one of the d components of J is an interval containing points of P of total weight at least $1/d$, and so it contains a point of X.

Proof of Lemma 8.6.3 by duality. We formulate the problem of choosing the weights x_p as a linear program with variables x_p, $p \in P$:

$$\begin{array}{ll} \text{minimize} & \sum_{p \in P} x_p \\ \text{subject to} & \sum_{p \in J \cap P} x_p \ge 1 \text{ for every } J \in \mathcal{J}, \\ & \mathbf{x} \ge \mathbf{0}. \end{array}$$

The linear program is certainly feasible; for instance, $x_p = 1$ for all p is a feasible solution. We would like to show that the optimum is at most $2d$.

Using the dualization recipe from Section 6.2, we find that the dual linear program has variables y_J, $J \in \mathcal{J}$, and it looks as follows:

$$\begin{array}{ll} \text{maximize} & \sum_{J \in \mathcal{J}} y_J \\ \text{subject to} & \sum_{J: p \in J \cap P} y_J \le 1 \text{ for every } p \in P, \\ & \mathbf{y} \ge \mathbf{0}. \end{array}$$

The dual linear program is feasible, too, since $\mathbf{y} = \mathbf{0}$ is feasible. Then by the Duality Theorem both the primal and the dual linear programs have optimal solutions, $\mathbf{x}^*$ and $\mathbf{y}^*$, respectively, that yield the same value of the objective functions.

We may assume that $\mathbf{y}^*$ is rational: Indeed, we may take it to be a basic feasible solution, and all basic feasible solutions are rational since all coefficients of the linear program are rational.

We now have some rational weight y_J^* for every $J \in \mathcal{J}$ such that no point of P is contained in d-intervals of total weight exceeding 1, and we want to show that the total weight of all $J \in \mathcal{J}$ cannot be larger than $2d$.

Lemma 8.6.2 tells us that if all the d-intervals had the same weight, then there is a point contained in d-intervals of weight at least $W/2d$, where W is the total weight of all d-intervals. Our weights need not be all equal, but we will pass to the case of equal weights by replicating each d-interval a suitable number of times.

Let D be a common denominator of all the rational numbers y_J^*, $J \in \mathcal{J}$, and let $y_J^* = \frac{r_J}{D}$, with r_J integral. Let $(J_1, J_2, \ldots, J_n)$ be a sequence that includes each d-interval $J \in \mathcal{J}$ exactly r_J-times (thus $n = \sum_{J\in\mathcal{J}} r_J$). By Lemma 8.6.2 there is a point $p \in P$ contained in at least $n/2d$ members of the sequence, which means

$$\sum_{J\in\mathcal{J}:\, p\in J} r_J \geq \frac{n}{2d} = \frac{1}{2d} \sum_{J\in\mathcal{J}} r_J.$$

Dividing both sides by the common denominator D and multiplying by $2d$ gives

$$2d \cdot \sum_{J\in\mathcal{J}:\, p\in J} y_J^* \geq \sum_{J\in\mathcal{J}} y_J^*.$$

Since $\mathbf{y}^*$ is a feasible solution of the dual linear program, the left-hand side is at most $2d$, and so $\sum_{J\in\mathcal{J}} y_J^* \leq 2d$. This concludes the proof of Lemma 8.6.3 as well as of Theorem 8.6.1. □

Transversal number and matching number. Let us now look at some of the concepts appearing in the proof of Theorem 8.6.1 in a general context. Let V be an arbitrary finite set, and let $\mathcal{F}$ be a system of subsets of V.

A set $X \subseteq V$ is called a *transversal* of $\mathcal{F}$ if $F \cap X \neq \emptyset$ for every $F \in \mathcal{F}$. The **transversal number** $\tau(\mathcal{F})$ is the minimum possible number of elements of a transversal of $\mathcal{F}$.

Determining or estimating the transversal number of a given set system is an important basic problem in combinatorics and combinatorial optimization, including many other problems as special cases. For example, if we consider a graph $G = (V, E)$, and view the edges as two-element subsets of V, then a transversal of E is exactly what was called a vertex cover in Section 3.3.

Another useful notion is the **matching number** of $\mathcal{F}$, denoted by $\nu(\mathcal{F})$ and defined as the maximum number of sets in a subsystem $\mathcal{M} \subseteq \mathcal{F}$ such that no two distinct sets $F_1, F_2 \in \mathcal{M}$ intersect (such an $\mathcal{M}$ is called a *matching*).

It is easily seen that $\nu(\mathcal{F}) \leq \tau(\mathcal{F})$ for every $\mathcal{F}$, since we clearly have $\tau(\mathcal{M}) = \nu(\mathcal{M})$ for every matching $\mathcal{M}$

For the graph example, $\nu(E)$ is exactly the number of edges in a maximum matching. This is also where the name "matching number" comes from.

The condition in Theorem 8.6.1 that every two d-intervals in $\mathcal{J}$ intersect can be rephrased as $\nu(\mathcal{J}) = 1$. More generally, $\nu(\mathcal{F}) \le k$ means that among every $k+1$ members of $\mathcal{F}$ we can find two that intersect. There is a more general version of Theorem 8.6.1 stating that $\tau(\mathcal{J}) \le 2d^2\nu(\mathcal{J})$ for every finite family of d-intervals. The proof is very similar to the one shown for Theorem 8.6.1, only the analog of Lemma 8.6.2 needs the well-known Turán's theorem from graph theory.

Fractional transversals and matchings. In the proof of Theorem 8.6.1 we have implicitly used another parameter of a set system, which always lies between $\nu(\mathcal{F})$ and $\tau(\mathcal{F})$ and which, unlike $\tau(\mathcal{F})$ and $\nu(\mathcal{F})$, is efficiently computable. This new parameter can be introduced in two seemingly different ways, which turn out to be equivalent by the Duality Theorem of linear programming.

A *fractional transversal* of $\mathcal{F}$ is any feasible solution $\mathbf{x}$ to the linear program

$$\begin{array}{ll} \text{minimize} & \sum_{v\in V} x_v \\ \text{subject to} & \sum_{v\in F} x_v \ge 1 \text{ for every } F\in\mathcal{F}, \\ & \mathbf{x}\ge\mathbf{0}. \end{array}$$

So in a fractional transversal one can take, say, one-third of one point and two-thirds of another, but for each set, the fractions for points in that set must add up to at least 1, one full point. The *fractional transversal number* $\tau^*(\mathcal{F})$ is the optimal value of the objective function, i.e., the minimum possible total weight of a fractional transversal.

Every transversal T corresponds to a fractional transversal, given by $x_v = 1$ if $v \in T$ and $x_v = 0$ otherwise, and thus $\tau^*(\mathcal{F}) \le \tau(\mathcal{F})$ for every $\mathcal{F}$.

A *fractional matching* for $\mathcal{F}$ is any feasible solution $\mathbf{y}$ to the linear program

$$\begin{array}{ll} \text{maximize} & \sum y_F \\ \text{subject to} & \sum_{F:\, v\in F} y_F \le 1 \text{ for every } v\in V, \\ & \mathbf{y}\ge\mathbf{0}, \end{array}$$

and the optimal value of the objective function is the *fractional matching number* $\nu^*(\mathcal{F})$.

Every matching $\mathcal{M}$ yields a fractional matching (we put $y_F = 1$ for $F \in \mathcal{M}$ and $y_F = 0$ otherwise). Thus, $\nu(\mathcal{F}) \le \nu^*(\mathcal{F})$.

Since the linear programs for τ^* and for ν^* are dual to each other, we always have $\nu^*(\mathcal{F}) = \tau^*(\mathcal{F})$, and altogether we have the promised chain of inequalities

$$\nu(\mathcal{F}) \le \nu^*(\mathcal{F}) = \tau^*(\mathcal{F}) \le \tau(\mathcal{F}).$$

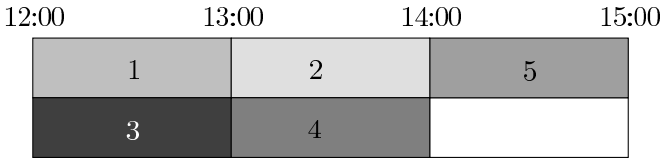
The proof of Theorem 8.6.1 can now be viewed as follows: First one proves that $\nu^*(\mathcal{J}) \le 2d$ for every family of d-intervals with $\nu(\mathcal{J}) = 1$ (Lemma 8.6.2 and a part of the argument in the proof of Lemma 8.6.3), and then one shows $\tau(\mathcal{J}) \le d\cdot\tau^*(\mathcal{J})$ (the derivation of Theorem 8.6.1 from Lemma 8.6.3). This proof scheme turned out to be very powerful and it works in many other cases as well.

Fractional concepts. Besides the fractional matching and transversal numbers, many other "fractional" quantities appear in combinatorics and combinatorial optimization. The general recipe is to take some useful integer-valued parameter Q of a graph, say, re-formulate its definition as an integer program, and introduce a "fractional Q" as the optimum value of a suitable LP relaxation of the integer program. In many cases such a fractional Q is useful for studying or approximating the original Q. The book

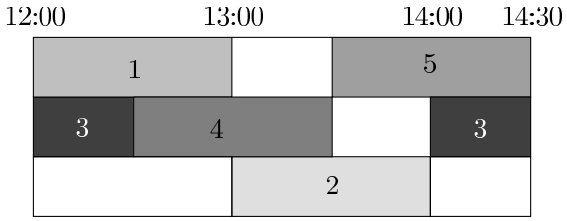
> E. R. Scheinerman and D. H. Ullman: *Fractional Graph Theory*, John Wiley & Sons, New York 1997

nicely presents such developments. Let us conclude this section by quoting an example from that book. We consider five committees, numbered 1,2,...,5, such that 1 and 2 have a common member, and so have 2 and 3, 3 and 4, 4 and 5, and 5 and 1, while any other pair of committees is disjoint. A one-hour meeting should be scheduled for each committee, and meetings of committees with a common member must not overlap. What is the length of the shortest time interval in which all the five meetings can be scheduled?

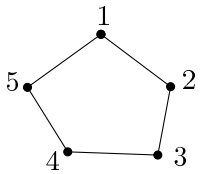
It seems that a 3-hour schedule like the one below should be optimal:



However, if one of the committees is willing to break its meeting into two half-hour parts, then a shorter schedule is possible



The first schedule corresponds to a proper coloring of the conflict graph

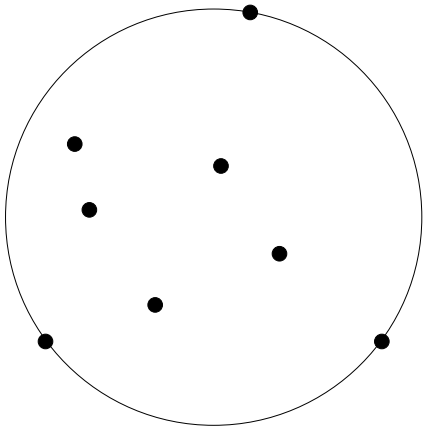


by three colors, while the second schedule corresponds to a *fractional coloring* of the same graph, with value 2.5.

8.7 Smallest Balls and Convex Programming

The smallest ball problem. We are given points $\mathbf{p}_1, \ldots, \mathbf{p}_n \in \mathbb{R}^d$, and we would like to find a ball of the smallest radius that contains all the points.[8]

[8] In the plane this is sometimes referred to as the *smallest bomb problem*, but we prefer not to elaborate this association into a real-life story.



This looks similar to some of the geometric optimization problems that we have addressed in Chapter 2, such as the problem of placing a largest possible disk inside a convex polygon. For the smallest ball problem, however, there is no simple trick that lets us write the problem as a linear program.

We will see that it can be formulated as a *convex quadratic program*, which is in many respects the next best thing to a linear program. There are efficient solvers for convex quadratic programs, based on interior-point methods or on simplex-type methods, and so this formulation can be used for computing a smallest enclosing ball in practice.

We will also derive from this formulation that the smallest enclosing ball always exists, and it is determined uniquely. (This is intuitively very plausible; think of a shrinking rubber ball.) In the course of the proof, we will establish a powerful criterion for optimality of a feasible solution of a convex program, known (in a much more general context) as the *Karush–Kuhn–Tucker conditions*. These conditions are of outstanding theoretical value, and they are the basis of efficient solution methods for many classes of optimization problems, including convex quadratic programming. The reader might still wonder, how does all of this relate to linear programming? We will use the Duality Theorem of linear programming for deriving the Karush–Kuhn–Tucker conditions.

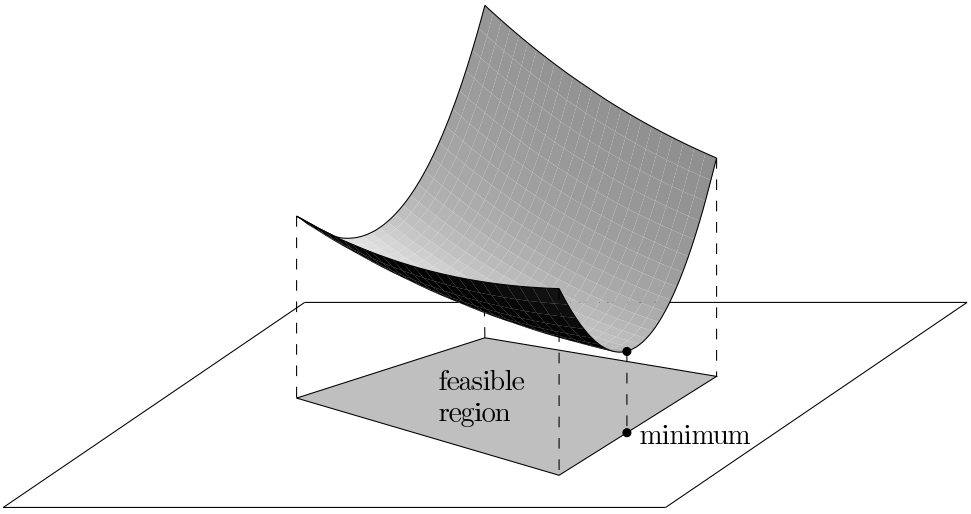
We begin by introducing convex programming, and we will return to the smallest enclosing ball problem later.

A short introduction to convex programming. Let us recall that an n-variate function $f\colon \mathbb{R}^n \to \mathbb{R}$ is *convex* if

$$f((1-t)\mathbf{x} + t\mathbf{y}) \le (1-t)f(\mathbf{x}) + tf(\mathbf{y})$$

for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and all $t \in [0,1]$. Geometrically, the segment connecting the points $(\mathbf{x}, f(\mathbf{x}))$ and $(\mathbf{y}, f(\mathbf{y}))$ in $\mathbb{R}^{n+1}$ never goes below the graph of f.

A **convex program** is the problem of minimizing a convex function in n variables subject to linear equality and inequality constraints. The following picture illustrates a 2-dimensional convex programming problem, with a planar feasible region given by four inequality constraints:



Note that the minimum need not occur at a vertex of the feasible region. We should also remark that, unlike in linear programming, we cannot change minimization to maximization, since for f convex, $-f$ is typically not convex (unless f is linear). Actually, *maximizing* a convex function subject to linear constraints is a computationally difficult (NP-hard) problem.

Here we will consider convex programs in equational form:

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq 0, \end{array} \tag{8.18}$$

where $A \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^m$, and $f\colon \mathbb{R}^n \to \mathbb{R}$ is a convex function.

In order to use calculus, we also assume that f is differentiable, with continuous partial derivatives. In this situation, the inequality

$$f(\mathbf{x}) \geq f(\mathbf{z}) + \nabla f(\mathbf{z})(\mathbf{x} - \mathbf{z}) \tag{8.19}$$

holds for all $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$, and this is an alternative characterization of convexity. Here

$$\nabla f(\mathbf{z}) = \left(\frac{\partial}{\partial x_1} f(\mathbf{x})|_{\mathbf{x}=\mathbf{z}}, \ldots, \frac{\partial}{\partial x_n} f(\mathbf{x})|_{\mathbf{x}=\mathbf{z}} \right)$$

is the *gradient* (vector of partial derivatives) of f at $\mathbf{z}$, which by convention is an n-dimensional *row* vector. Thus $\nabla f(\mathbf{z})(\mathbf{x} - \mathbf{z})$ is the scalar product of the row vector $\nabla f(\mathbf{z})$ with the column vector $\mathbf{x} - \mathbf{z}$.

Geometrically, the inequality says that the epigraph of f lies above all of its tangential hyperplanes. This has the following easy consequence.

8.7.1 Fact. *Let $C \subseteq \mathbb{R}^n$ be a convex set and $f\colon \mathbb{R}^n \to \mathbb{R}$ a differentiable convex function. A vector $\mathbf{x}^*$ minimizes $f(\mathbf{x})$ over C if and only if*

$$\nabla f(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*) \geq 0 \quad \textit{for all } \mathbf{x} \in C.$$

Proof. First we prove that the inequality implies optimality of $\mathbf{x}^*$. Using (8.19), $\nabla f(\mathbf{x}^*)(\mathbf{x}-\mathbf{x}^*) \geq 0$ implies $f(\mathbf{x}) \geq f(\mathbf{x}^*)$, so if the former holds for all $\mathbf{x} \in C$, $\mathbf{x}^*$ is a minimizer of f over C.

For the other direction, let us assume that $\mathbf{x}^*$ is a minimizer and that $\mathbf{x} \in C$. We consider the convex combination

$$\mathbf{x}(t) := \mathbf{x}^* + t(\mathbf{x}-\mathbf{x}^*) \in C, \quad t \in [0,1],$$

and we observe that

$$\frac{\partial}{\partial t} f(\mathbf{x}(t))|_{t=0} = \lim_{t\to 0} \frac{f(\mathbf{x}(t)) - f(\mathbf{x}^*)}{t} \geq 0$$

must hold, for otherwise, $f(\mathbf{x}(t)) < f(\mathbf{x}^*)$ for some small t. On the other hand, we have

$$\frac{\partial}{\partial t} f(\mathbf{x}(t))|_{t=0} = \nabla f(\mathbf{x}^*)(\mathbf{x}-\mathbf{x}^*),$$

by the chain rule. This completes the proof. □

Next we formulate and prove the promised optimality criterion for convex programming.

8.7.2 Proposition (Karush–Kuhn–Tucker conditions). *Let us consider the convex program*

$$\begin{array}{ll} \textit{minimize} & f(\mathbf{x}) \\ \textit{subject to} & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array}$$

with f convex and differentiable. A feasible solution $\mathbf{x}^* \in \mathbb{R}^n$ is optimal if and only if there is a vector $\mathbf{m} \in \mathbb{R}^m$ such that for all $j \in \{1,\dots,n\}$,

$$\nabla f(\mathbf{x}^*)_j + \mathbf{m}^T \mathbf{a}_j \begin{cases} = 0 & \text{if } x_j^* > 0 \\ \geq 0 & \text{otherwise.} \end{cases}$$

Here $\mathbf{a}_j$ is the j-th column of A.

The components of $\mathbf{m}$ are called the Karush–Kuhn–Tucker multipliers.

Proof. First we assume that there is a vector $\mathbf{m}$ with the above properties, and we let $\mathbf{x}$ be any feasible solution to the convex program. Then we get

$$\begin{aligned} \Big(\nabla f(\mathbf{x}^*) + \mathbf{m}^T A\Big)\mathbf{x}^* &= 0, \\ \Big(\nabla f(\mathbf{x}^*) + \mathbf{m}^T A\Big)\mathbf{x} &\geq 0. \end{aligned}$$

Subtracting the first equation from the second, the contributions of $\mathbf{m}^T A$ cancel (since $A\mathbf{x} = \mathbf{b} = A\mathbf{x}^*$ by the feasibility of $\mathbf{x}$ and $\mathbf{x}^*$), and we conclude that

$$\nabla f(\mathbf{x}^*)(\mathbf{x}-\mathbf{x}^*) \geq 0.$$

Since this holds for all feasible solutions $\mathbf{x}$, the solution $\mathbf{x}^*$ is optimal by Fact 8.7.1.

Conversely, let $\mathbf{x}^*$ be optimal, and let us set $\mathbf{c}^T = -\nabla f(\mathbf{x}^*)$. By Fact 8.7.1 we have $\mathbf{c}^T(\mathbf{x} - \mathbf{x}^*) \leq 0$ for all feasible solutions $\mathbf{x}$, meaning that $\mathbf{x}^*$ is an optimal solution to the linear program

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}. \end{array}$$

According to the dualization recipe (Section 6.2), its dual is the linear program

$$\begin{array}{ll} \text{minimize} & \mathbf{b}^T\mathbf{y} \\ \text{subject to} & A^T\mathbf{y} \geq \mathbf{c}, \end{array}$$

and the Duality Theorem implies that it has an optimal solution $\mathbf{y}^*$ satisfying

$$\mathbf{b}^T\mathbf{y}^* = \mathbf{c}^T\mathbf{x}^*. \tag{8.20}$$

We set $\mathbf{m} = \mathbf{y}^*$. Since $\mathbf{m}$ is a feasible solution of the dual linear program, we have $\mathbf{m}^T\mathbf{a}_j \geq c_j$ for all j, and (8.20) implies

$$(\mathbf{m}^T A - \mathbf{c}^T)\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^* - \mathbf{c}^T\mathbf{x}^* = 0.$$

So we have $\nabla f(\mathbf{x}^*)_j + \mathbf{m}^T\mathbf{a}_j = -c_j + \mathbf{m}^T\mathbf{a}_j \geq 0$, with equality whenever $x_j^* > 0$. Therefore, we have found the desired multipliers $\mathbf{m}$. □

The fact that there is dualization for everyone implies Karush-Kuhn-Tucker conditions for everyone. We encourage the reader to work out the details and we mention only one special case here: A feasible solution $\mathbf{x}^*$ to the convex programming problem

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & A\mathbf{x} = \mathbf{b} \end{array}$$

is optimal if and only if there exists a vector $\mathbf{m}$ such that

$$\nabla f(\mathbf{x}^*) + \mathbf{m}^T A = \mathbf{0}^T.$$

In this special case, the entries of $\mathbf{m}$ are called *Lagrange multipliers* and can be obtained from $\mathbf{x}^*$ through Gaussian elimination. If, in addition, f is a quadratic function, its gradient is linear, so the minimization problem itself (finding an optimal $\mathbf{x}^*$ with a matching $\mathbf{m}$) can be solved through Gaussian elimination. For example, the problem of fitting a line by the method of *least squares*, mentioned in Section 2.4, is of this easy type, because its bivariate quadratic objective function (2.1) is convex.

Smallest enclosing ball as a convex program. We are ready to state and prove a theorem characterizing the smallest enclosing ball of a point set.

8.7.3 Theorem. *Let $\mathbf{p}_1, \ldots, \mathbf{p}_n$ be points in $\mathbb{R}^d$, and let Q be the $d \times n$ matrix whose j-th column is the point $\mathbf{p}_j$. Let us consider the optimization problem*

$$\begin{array}{ll} \textit{minimize} & \mathbf{x}^T Q^T Q \mathbf{x} - \sum_{j=1}^n x_j \mathbf{p}_j^T \mathbf{p}_j \\ \textit{subject to} & \sum_{j=1}^n x_j = 1, \\ & \mathbf{x} \geq \mathbf{0} \end{array} \tag{8.21}$$

in the variables $x_1, \ldots, x_n$ (collected in the vector $\mathbf{x}$). Then the objective function $f(\mathbf{x}) := \mathbf{x}^T Q^T Q \mathbf{x} - \sum_{j=1}^n x_j \mathbf{p}_j^T \mathbf{p}_j$ is convex, and the following statements hold.

(i) *Problem (8.21) has an optimal solution $\mathbf{x}^*$.*

(ii) *There exists a point $\mathbf{p}^*$ such that $\mathbf{p}^* = Q\mathbf{x}^*$ holds for* every *optimal solution $\mathbf{x}^*$. Moreover, the ball with center $\mathbf{p}^*$ and squared radius $-f(\mathbf{x}^*)$ is the unique ball of smallest radius containing P.*

Proof of Theorem 8.7.3. The matrix $Q^T Q$ is positive semidefinite, and from this the convexity of f is easy to verify (we leave it as an exercise).

The feasibility region of program (8.21) is a compact set (actually, a *simplex*), and we are minimizing a continuous function over it. Consequently, there exists an optimal solution $\mathbf{x}^*$. The program has only one equality constraint; applying the Karush–Kuhn–Tucker conditions (Proposition 8.7.2), with $\mathbf{p}^* = Q\mathbf{x}^*$, we find that there is a 1-dimensional vector $\mathbf{m} = (\mu)$ such that

$$2\mathbf{p}_j^T \mathbf{p}^* - \mathbf{p}_j^T \mathbf{p}_j + \mu \begin{cases} = 0 & \text{if } x_j^* > 0 \\ \geq 0 & \text{otherwise.} \end{cases} \tag{8.22}$$

Subtracting $\mathbf{p}^{*T}\mathbf{p}^*$ from both sides and multiplying with -1 yields

$$\|\mathbf{p}_j - \mathbf{p}^*\|^2 \begin{cases} = \mu + \mathbf{p}^{*T}\mathbf{p}^* & \text{if } x_j^* > 0 \\ \leq \mu + \mathbf{p}^{*T}\mathbf{p}^* & \text{otherwise.} \end{cases}$$

This means that $\mathbf{p}^*$ is the center of a ball of squared radius $\rho^2 = \mu + \mathbf{p}^{*T}\mathbf{p}^*$ that contains all points from P and has the points $\mathbf{p}_j$ with $x_j^* > 0$ on the boundary. From (8.22) and $\mathbf{x} \geq \mathbf{0}$ we also get

$$\mu = \sum_{j=1}^n x_j^* \mu = \sum_{j=1}^n x_j^* \mathbf{p}_j^T \mathbf{p}_j - 2\sum_{j=1}^n x_j^* \mathbf{p}_j^T \mathbf{p}^* = \sum_{j=1}^n x_j^* \mathbf{p}_j^T \mathbf{p}_j - 2\mathbf{p}^{*T}\mathbf{p}^*,$$

and $\rho^2 = \sum_{j=1}^n x_j^* \mathbf{p}_j^T \mathbf{p}_j - \mathbf{p}^{*T}\mathbf{p}^* = -f(\mathbf{x}^*)$ follows.

It remains to prove that there can be no other ball of squared radius at most ρ^2 that contains all points from P (this also shows that $\mathbf{p}^*$ does not depend on the choice of $\mathbf{x}^*$).

Given the center $\mathbf{p} \neq \mathbf{p}^*$ of such a potential ball, we can uniquely write it in the form

$$\mathbf{p} = \mathbf{p}^* + t\mathbf{u},$$

where $\mathbf{u}$ is a vector of length 1 and $t > 0$. We define

$$F := \{j \in \{1, \ldots, n\} \mid x_j^* > 0\}.$$

This set F is nonempty since $\sum_{i=1}^n x_j^* = 1$. With $j \in F$, we get

$$\begin{aligned} (\mathbf{p}_j - \mathbf{p})^T (\mathbf{p}_j - \mathbf{p}) &= (\mathbf{p}_j - \mathbf{p}^* - t\mathbf{u})^T (\mathbf{p}_j - \mathbf{p}^* - t\mathbf{u}) \\ &= (\mathbf{p}_j - \mathbf{p}^*)^T (\mathbf{p}_j - \mathbf{p}^*) + t^2 \mathbf{u}^T \mathbf{u} - 2t\mathbf{u}^T (\mathbf{p}_j - \mathbf{p}^*) \\ &= \rho^2 + t^2 - 2t\mathbf{u}^T (\mathbf{p}_j - \mathbf{p}^*), \end{aligned}$$

where the last equality holds because of (8.22) and $x_j^* > 0$.

In order for $\mathbf{p}$ to define a ball of squared radius at most ρ^2 that contains all points, and in particular, the points $\mathbf{p}_j$, $j \in F$, we must have

$$\mathbf{u}^T(\mathbf{p}_j - \mathbf{p}^*) > 0, \quad j \in F, \tag{8.23}$$

because whenever this fails for some $j \in F$, we get $(\mathbf{p}_j - \mathbf{p})^T(\mathbf{p}_j - \mathbf{p}) > \rho^2$.

Using $x_j^* > 0, j \in F$, inequality (8.23) then yields

$$\sum_{j \in F} x_j^* \mathbf{u}^T(\mathbf{p}_j - \mathbf{p}^*) > 0.$$

On the other hand, we have $\sum_{j \in F} x_j^* = 1$ and $\mathbf{p}^* = \sum_{j=1}^n x_j^* \mathbf{p}_j = \sum_{j \in F} x_j^* \mathbf{p}_j$, which implies

$$\sum_{j \in F} x_j^* \mathbf{u}^T(\mathbf{p}_j - \mathbf{p}^*) = \mathbf{u}^T \left(\sum_{j \in F} x_j^* \mathbf{p}_j - \sum_{j \in F} x_j^* \mathbf{p}^* \right) = \mathbf{u}^T(\mathbf{p}^* - \mathbf{p}^*) = 0,$$

showing that (8.23) cannot hold. This means that $\mathbf{p} \neq \mathbf{p}^*$ cannot define an enclosing ball of squared radius at most ρ^2, so $\mathbf{p}^*$ is indeed the center of the unique smallest enclosing ball, and $-f(\mathbf{x}^*)$ is the squared radius of it. □

We have actually shown more than required: The center of the smallest enclosing ball is a convex combination of the input points that lie on its boundary: Any optimal solution $\mathbf{x}^*$ witnesses this fact.

A recent book on the topics of this section is

S. Boyd and L. Vandenberghe: *Convex optimization*, Cambridge University Press, Cambridge 2004.

9

Software and Further Reading

The most famous (and expensive) software package for solving linear programs and integer programs is called CPLEX. Freely available codes with similar functionality, although not quite as strong as CPLEX, are `lp_solve`, GLPK, and CLP. The web site

`www-neos.mcs.anl.gov/neos`

contains a guide to many other optimization software systems, and it also provides an overview of web solvers, to which one can send an input of an optimization problem and, with a bit of luck, be returned an optimum.

The web bookstore Amazon offers more books with "linear programming" in the title than with "astrology," and so it is clear that we can mention only a very narrow selection from the literature.

A slim modern textbook of linear programming is

H. Karloff: *Linear Programming*, Birkhäuser, Boston 1991.

Both linear and integer programming are treated nicely and on an excellent theoretical level in

A. Schrijver: *Theory of Linear and Integer Programming*, Wiley-Interscience, New York 1986.

The book

V. Chvátal: *Linear Programming*, W. H. Freeman, New York 1983

is considered one of the best textbooks up until the present. And those liking classical sources may appreciate

G. B. Dantzig: *Linear Programming and Extensions*, Princeton University Press, Princeton 1963.