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: Not only does it play a role only in mathematical economy, it appears frequently in computer science and in many other fields.

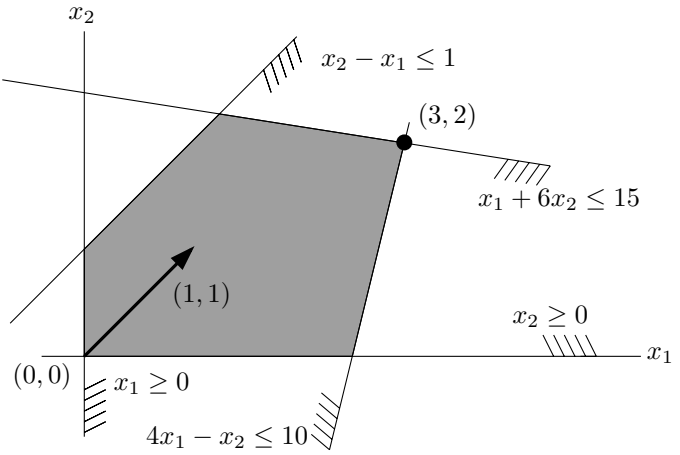
1.1 A Linear Program

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

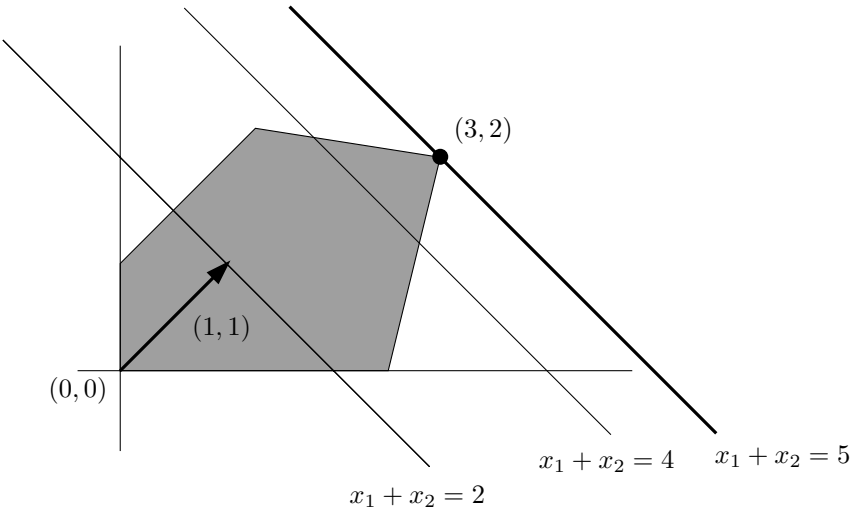
$$\begin{array}{ll} \text{Maximize the value} & x_1 + x_2 \\ \text{among all vectors } (x_1, x_2) \in \mathbb{R}^2 & \\ \text{satisfying the constraints} & x_1 \geq 0 \\ & x_2 \geq 0 \\ & x_2 - x_1 \leq 1 \\ & x_1 + 6x_2 \leq 15 \\ & 4x_1 - x_2 \leq 10. \end{array}$$

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. To make it more precise, we consider a line perpendicular to the arrow, and we think of translating it in the direction of the arrow. Then we are seeking a point where the moving line intersects our polygon for the last time. (Let us 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 $\mathbf{x} \in \mathbb{R}^n$ that satisfy 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.

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, with all variables appearing on the left-hand side. 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.

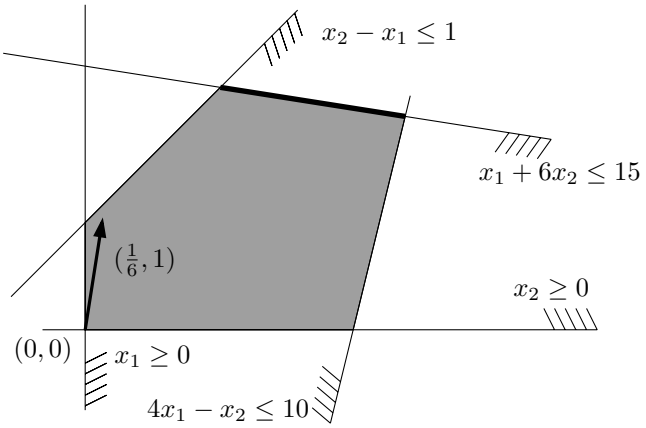
A linear program may in general have a single optimal solution, or infinitely many optimal solutions, or none at all.

We have seen a situation with a single optimal solution in the first example of a linear program. We will present examples of the other possible situations.

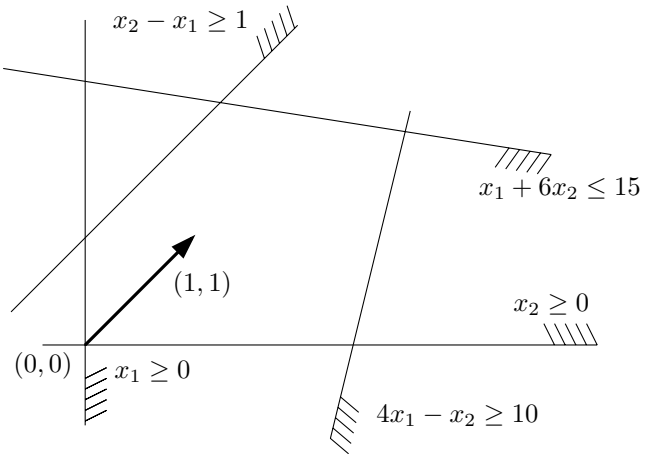
[1] Here we regard the vector $\mathbf{c}$ as an $n \times 1$ matrix, and so the expression $\mathbf{c}^T\mathbf{x}$ is a product of a $1 \times n$ matrix and an $n \times 1$ matrix. This product, formally speaking, should be a 1×1 matrix, but we regard it as a real number.

Some readers might wonder: If we consider $\mathbf{c}$ a column vector, why, in the example above, don't we write it as a column or as $(1, 1)^T$? For us, a vector is an n-tuple of numbers, and when writing an explicit vector, we separate the numbers by commas, as in $\mathbf{c} = (1, 1)$. Only if a vector appears in a context where one expects a matrix, that is, in a product of matrices, *then* it is regarded as (or "converted to") an $n \times 1$ matrix. (However, sometimes we declare a vector to be a row vector, and then it behaves as a $1 \times n$ matrix.)

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:

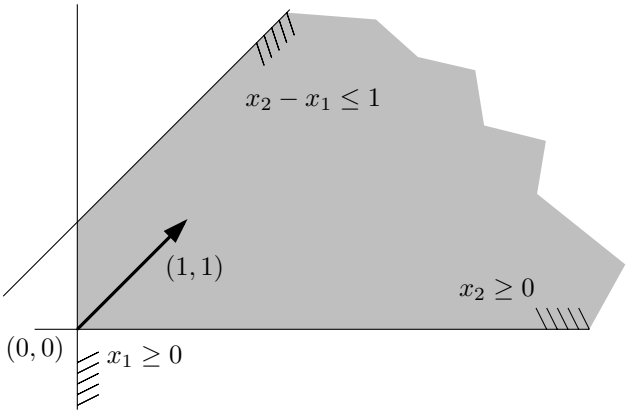


If we reverse the directions of the inequalities in the constraints $x_2 - x_1 \leq 1$ and $4x_1 - x_2 \leq 10$ in our first example, we obtain a linear program that has no feasible solution, and hence no optimal solution either:



Such a linear program is called **infeasible**.

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, as shown in the next picture:



Let us summarize: We have seen that a linear program can have one or infinitely many optimal solutions, but it may also be unbounded or infeasible. Later we will prove that no other 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 find an optimal solution graphically. A substantial linear program in practice often has several thousand 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 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 long and strenuous research, and efficient methods for linear programming are not simple.

In order that the above piece of knowledge will also make sense forever, one should not forget what a linear program is, so we repeat it once again:

A linear program is the problem of maximizing a given linear function over the set of all vectors that satisfy a given system of linear equations and inequalities. Each linear program can easily be transformed to the form

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

1.2 What Can Be Found in This Book

The rest of Chapter 1 briefly discusses the 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 must attain integer values. In this context we will see how linear programming can help in approximate solutions 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 polyhedra, are important in many other branches of mathematics and computer science as well.

Chapter 5 covers the simplex method, which is a fundamental algorithm for linear programming. In full detail it is relatively complicated, and from the contemporary point of view it is not necessarily the central topic in a first course on linear programming. In contrast, 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 omit some technical issues.

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

Chapter 9 contains remarks on software available for linear programming and on the literature.

Linear algebra is the main mathematical tool throughout the book. The required linear-algebraic notions and results are summarized in an appendix.

The book concludes with a glossary of terms that are common in linear programming but do not appear in the main text. Some of them are listed to

ensure that our index can compete with those of thicker books, and others appear as background material for the advanced reader.

Two levels of text. This book should serve mainly 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 the 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 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 of $\mathbb{R}^n$, which is an important linear-algebraic notion.[2]

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

[2] An affine subspace is a linear subspace translated by a fixed vector $\mathbf{x} \in \mathbb{R}^n$. For example, every point, every line, and $\mathbb{R}^2$ itself are the affine subspaces of $\mathbb{R}^2$.

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 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 that of Gaussian 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 tabular form:

	Basic problem	Algorithm	Solution set
Linear algebra	system of linear equations	Gaussian 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

[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.

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 earlier, in 1939, Leonid Vitalyevich Kantorovich was charged with the reorganization of the 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 happens under such regimes, his discoveries went almost unnoticed and nobody continued his work. Kantorovich together with Tjalling Koopmans received the Nobel Prize in Economics in 1975, for pioneering work in resource allocation. Somewhat ironically, Dantzig, whose contribution to linear programming is no doubt much more significant, was never awarded a Nobel Prize.

The discovery of the simplex method had a great impact on both theory and practice in economics. 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 costs were cut by 20%, say, by a mere reorganization according to some mysterious calculation. Especially when such a 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 economics.

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 a 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 of 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 faster computers. On the one hand, the implementation of the simplex method has been refined considerably, and on the other hand, new computational methods based on completely different ideas have been developed. This latter development will be described in Chapter 7.