Discrete Optimization

Guyslain Naves

Fall 2010

Contents

Introduction to linear programming

Mathematical programming is the part of applied mathematics that deals with finding solutions to mathematical problem in the most efficient way. Mathematical programming takes its origin in the second world war. At that time, the German submarines were inflicting considerable damages to the allied navy. This was greatly affecting the war effort, in particular in Great Britain which was very dependant on the supply from the other continents. The allied had to organize their logistic in a *scientific* way to counter German strategy. This leads mathematicians to develop *efficient methods* to solve pratical optimization problems. It was at that time that were founded the basis of *linear programming.* After the war, mathematical programming became a major tool for increasing the productivity of companies, and found other domains of applications, for example in economy. To illustrate the importance of mathematical programming, let's just say that one of the major companies in this domain, Ilog, has been bought recently by IBM for \$340 millions.

A *mathematical program* is given by a set of variable X, and some function $f : \mathbb{R}^X \to \mathbb{R}^+$ that we want to maximize, that is we want to give a value to each variable, giving a vector x, such that $f(x)$ is maximum. Moreover, the values for x can be submitted to additional constraints, given functions $g_1, \dots g_k$. We only accept the solution x that satisfy $g_1(x) = g_2(x) = \dots = g_k(x) = 0$. No need to say that it is not possible to give a method that solve any such problem. This is why we only look at particular cases, where f, g_1, …, g_k have some additional properties. This course is about linear programming, that is mathematical programming where the functions are linear: $f(x + y) = f(x) + f(y)$ for all x and y. There are several reasons why linear programming is interesting.

- It is sufficiently expressive to deal with most of the problems that we want to solve in practice.
- It relies on basic linear algebra, hence it has a beautiful yet accessible theory.
- Solvers are really efficient on linear programs: solving linear programs with tens of thousands of constraints and variables is considered as an easy task by now.
- It is very reliable: even when the datas are not very accurate, the solution computed by a linear solver is not too far from the correct answer.

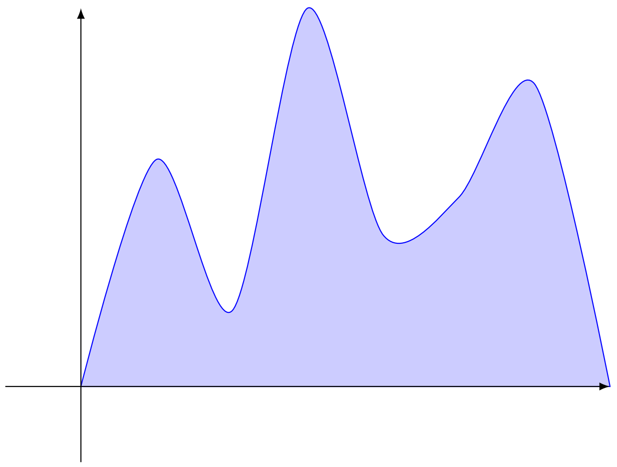


Figure 1: *A non-convex function can have local optima that are not globally optimum.*

- It supports post-analysis efficiently, allowing to make small changes to the model without recomputing a solution from scratch.

One of the most important properties of linear programs is the convexity of the constraints: given two vectors x and y satisfying the constraints of a linear program, the mean $\frac{x+y}{2}$ of these two vectors also satifies the constraints. The consequence is that if we have a solution x that is not optimal, and y is an optimal solution, we can improve our solution x by slightly moving in the direction of y. This will indeed improve the solution by the linearity of the objective function. Thus, a non-optimal solution can always be improved locally, there are no local optima that are not global. For example, in Figure 1, there are three local maximum, but only one global maximum for f. This would not happen with a convex function.

A very important result was obtained by Khachiyan in 1979. Linear programming can be solved in polynomial-time (under some technical requirements). The importance of this result is more theoretical than practical, his algorithm being inefficient in practice. But it shows that convexity is what makes mathematical programming works in many cases. As a consequence, it is theoretically possible to solve problems more general than linear programming. But linear programming happens to be more scalable and is still the major tool in operations research.

This course introduces linear programming, its method and some of its applications. The goal is to explain how practial problems can be modelized as linear programs and how to solve a linear programs efficiently. We will begin by an introduction of linear programming, and a first resolution method in Chapter 1. At that point, our algorithm will still look magical. Chapter 2 will go deeper into the mathematical theory. We will explain the relation between linear programming and geometry and find simple ways to prove the optimality of a solution. We will also see how some intriguing mathematical concepts are related to economic considerations. Then, in Chapter 3, we will basically make our algorithm faster and accurate, and show how to perform efficient post-analysis on a solution. Chapter 4 is dedicated to two classical applications of linear programming: matrix games and the cutting-stock problem. Chapter 5 studies the network simplex algorithm, a special case of our algorithm for solving transportation problems. From that, we will deduce some classical results of combinatorial optimization.

Chapter 1

The simplex method

1.1 The simplex method

Example: The iron foundry (Bradley, Hax, Magnanti)

- An iron foundry must produce 1000 pounds of casting,
- the casting must contain more than $0,45\%$ percents of manganese, and between 3.25% and 5.5% of silicone,
- the price of the casting is \$0.45 per pound,
- the price to melt down a pound of iron is \$0.5 per pound of pig iron,
- there are three different kinds of iron pigs available (the prices are given per thousand pounds):

	A	B	C
Silicon	4%	1%	0.6%
Manganese	0.45%	0.5%	0.4%
Price	\$21	\$25	\$15

- moreover, we can use pure manganese, at the price of \$8 per pound.

Question: How can the foundry maximize its profit?

To solve this problem in a mathematical way, one must first find *decision variables*. Finding good values for these variables will then be our main objective. In this example, the decision variables are the quantity of each iron pig plus the quantity of manganese used to produce the thousand pounds of casting. Hence, let x_1, x_2 and x_3 denotes respectively the number of thousands of pounds of pigs A, B, and C respectively, and x_4 the number of pounds of pure manganese used in the production.

Despite the uncertainty about the quantity of manganese and silicone in the final product, they do not constitute decision variables. We just have some freedom for the composition of the alloy, but the final product will depend only on the values taken by the four decision variable. Actually, we will introduce them latter in order to find a solution.

The next step is to find the *objective* of the program. For which quantity do we want to find an optimum (minimum or maximum)? In our problem, we want to maximize the profit made by selling this alloy. It is easily established from the price of the different pigs, of the manganese, and the price of the casting, plus the price of the melting down.

$$\max 4500 - 21.5x_1 - 25.5x_2 - 15.5x_3 - 8x_4$$

Note that the constant in the objective is useless, so we rewrite our objective as:

$$\max -21.5x_1 - 25.5x_2 - 15.5x_3 - 8x_4 = \min 21.5x_1 + 25.5x_2 + 15.5x_3 + 8x_4$$

That is, the maximal profit is reached when the cost of production is minimal.

Then, we must express the different requirements of the problem:

- on the quantity of manganese, we can write it as:

$$4.5x_1 + 5x_2 + 4x_3 + x_4 \geq 4.5$$

- on the quantity of silicone, we can decompose it in two inequalities:

$$40x_1 + 10x_2 + 6x_3 \geq 32.5$$

$$40x_1 + 10x_2 + 6x_3 \leq 55$$

- on the total quantity produced:

$$1000x_1 + 1000x_2 + 1000x_3 + x_4 = 1000$$

- finally, we must give the domain of each variable. Here, variables denote quantity of ressources, and the ressources are not limited. We can then choose any non-negative value:

$$x_1, x_2, x_3, x_4 \geq 0$$

We thus obtain a purely mathematical problem, which we state as:

$$\begin{array}{lrcrcrcrcl}
\min & 21.5x_1 & + & 25.5x_2 & + & 15.5x_3 & + & 8x_4 & \text{s.t.} & \\
& 4.5x_1 & + & 5x_2 & + & 4x_3 & + & x_4 & \geq & 4.5 \\
& 40x_1 & + & 10x_2 & + & 6x_3 & & & \geq & 32.5 \\
& 40x_1 & + & 10x_2 & + & 6x_3 & & & \leq & 55 \\
& 1000x_1 & + & 1000x_2 & + & 1000x_3 & + & x_4 & = & 1000 \\
& & & & & & & x_1, x_2, x_3, x_4 & \geq & 0
\end{array} \tag{1.1}$$

Let's give formal definitions for linear programming.

Definition 1.1.1. *A* linear constraint *over a set $\{x_1, \dots, x_n\}$ of variable is an inequation (or an equation) of the following form:*

$$\sum_{i=1}^{n} a_1 x_1 \sim b$$

where $a_1, \dots, a_n, b$ are scalars and $\sim \in \{\leq, \geq, =\}$.

Definition 1.1.2. *A* linear program *is a problem of the form: given a set of variable, maximize (or minimize) a linear form over these variables, while respecting some linear constraints. The linear form to optimize is called the* objective *of the linear program.*

Definition 1.1.3. *Given a linear program, a choice of values for the variables is a* feasible solution *if it satisfies all the constraints of the linear program. An* optimal solution *is a feasible solution that optimizes the objective.*

Our goal is, given a linear program, to find an optimal solution. Not all linear programs have an optimal solution, as is shown by the following examples.

Remark. We did not precised yet on which field we are working. All the results of this course hold for $\mathbb{Q}$ and $\mathbb{R}$, which is enough for us.

Example: The following linear program has no feasible solution, and so no optimal solution.

$$\begin{array}{lrcrcl}
\max & x_1 & - & x_2 & \text{s.t.} & \\
& -x_1 & + & x_2 & \geq & 1 \\
& x_1 & - & x_2 & \geq & 2
\end{array}$$

Indeed, by adding the two constraints, we get $0 \geq 3$. Such a system is called *infeasible.*

Example: Having a feasible solution is not even sufficient to have an optimal solution. Consider the following linear program.

$$\begin{array}{rrcrcl} \max & x_1 & & & \text{s.t.} & \\ & x_1 & + & x_2 & \geq & 1 \\ & x_1 & - & x_2 & \leq & 2 \end{array}$$

For any value $t \geq -1$, the vector $(t+2, t)$ is a feasible solution of the linear program, with objective $t+2$. Thus, there is no optimal solution. A linear program for which there are solutions of arbitrary large (or small in the case of a minimization program) is called *unbounded.*

Example: Even if we find an optimal solution, it is generally not unique.

$$\begin{array}{rrcrcrcl} \max & 3x_1 & + & x_2 & + & x_3 & \text{s.t.} & \\ & x_1 & + & x_2 & + & x_3 & \leq & 2 \\ & x_1 & & & & & \leq & 1 \\ & & & & & x_1, x_2, x_3 & \geq & 0 \end{array}$$

Here, any vector $(1, t, 1-t)$, for $0 \leq t \leq 1$, achieves a value of 4, which is the optimal solution. We thus have infinitely many solutions.

Hence, our goal is not just to find an optimal solution, but to do one of the following:

- find an optimal solution, and prove its optimality,
- or prove that there is no feasible solution,
- or prove that the linear program is unbounded, by giving a way to build arbitrarily large solutions.

We do not know yet what are the certificate for optimality. unfeasibility and unboundedness. But we can try to make an educated guess, based on the previous example. To prove infeasibility, it would be sufficient to deduce a contradiction from the system of inequation. A certificate would then describe how to combine the constraints into a new constraint like $1 \leq 0$.

For unboundedness, we would like to exhibit a solution x — $(2,0)$ in the previous example — and a a vector v with $cv > 0$ — $(1,1)$ here — such that $x + \lambda v$ is a feasible solution for all λ non-negative.

Remark. We particularly insist on the notion of certificate. As it is well-known in computer science, there is no fundamental difference between a proof and a program, hence a program giving just the answer "yes" or "no" could be considered as sufficient by many people. However, we believe that such an algorithm is very inferior to an algorithm answering "yes and this is the solution" or "no because of this". A practical reason is that such an algorithm gives a far more interesting information. A more theoretical reason is that every polynomial-time algorithm is in $NP \cap coNP$, that is, there actually are such certificate for the existence or non-existence of a solution. Finding those certificates generally explains why the problem is easy to solve, and gives the mathematical structure behind it. Understanding this structure is usually the best way to find efficient resolution algorithms.

1.1.1 Standard linear program

In order to apply the simplex method, we need to write the linear program into a special form, called the *standard form.*

Definition 1.1.4. *A linear program is in* standard form *if*

- *it is a maximisation program,*
- *all its variables are non-negative,*
- *all the other constraints are of the form* $\sum_i a_i x_i \leq b$.

Definition 1.1.5. *A variable of a linear program is* free *if there is no constraint on the positivity of the variable.*

Here are some rules to transform any linear program to an equivalent linear program in standard form.
Objective:
if the objective is to minimize a linear form, it is sufficient to multiply it by -1 to get a maximization problem.

Variables:

- A free variable x can be replaced by two new variables, $x = x^+ - x^-$. We substitute x in each constraint, and add the non-negativity constraint for the two new variable. Note that for a possible value of x, there are infinitely many possible values for x^+ and x^-.

- A non-positive variable x can be replaced by $x' = -x$, where x' is then non-negative.
- Additionally, a variable x with a constraint $x \geq b$ can be replaced by $x' = x - b$. We substitute x by $x' + b$ everywhere to get a non-negative variable.

Constraints:

- Equalities can be separated into two inequalities.
- Then we can multiply by -1 the greater-or-equal constraints to get lesser-or-equal constraints.

Non-linear objects:
Sometimes, linear constraints can be hidden behind more general constraints, as in the following examples.

- Absolute values can hide linear constraints. For instance, $|\sum_i a_i x_i| \leq b$ can be written as $-b \leq \sum_i a_i x_i \leq b$. But sometimes absolute values are not linear, like in the constraint $|x| \geq 3$. In that case the possible domain of x is not convex, so this constraint cannot be expressed by linear constraints.
- A maximum (or minimum) over linear forms may also sometimes be rewritten as a conjunction of linear constraints. Consider the constraint $\min\{3x_1+2x_2, 4x_1-x_2\}+x_3 \geq 5$. We introduce a new variable x' for the minimum, with constraints $x' \leq 3x_1+2x_2$ and $x' \leq 4x_1-x_2$. Then the original constraint can be written as $x'+x_3 \geq 5$. This works because to get the last constraint true, we want to find x' as big as possible, but less than the two original linear forms in the minimum, so we may assume that x' is the minimum.

Remark. We make a break in our process, and take some time to understand the meaning of the object we are manipulating. We are faced to the following problem (in matricial notations):

$$\min cx \text{ s.t.} Ax \leq b, x \geq 0$$

There are two different ways to read the part $Ax \leq b, \geq 0$. The first is to read it line by line, as we did so far. That is, we are looking for a vector that satisfies simultaneously m different linear inequalities. A second possibility, and perhaps more interesting, is to read it by columns: we are trying to find a positive combination of the columns of A, that is dominated by b.

For example, consider a feasible solution x_0. We have that $Ax_0 = b' \leq b$ and $cx_0 = z$ for some constant z and some vector b'. But then, any solution to $Ax = b'$, $cx = z$ is a solution with the same objective value. By seeing it as a combination of columns, there is a linear combination with at most $\operatorname{rank} A + 1$ non-zero coordinates (by linear algebra). As we are looking only for positive combination, it is not useful by itself, but it will appears in the following that looking to feasible solutions with only m non-zero entries is enough to find an optimal solution.

Let's end this section by rewriting the linear program for the iron foundry in standard form:

$$\begin{array}{lrcrcrcrcl}
\max & -21.5x_1 & - & 25.5x_2 & - & 15.5x_3 & - & 8x_4 & & \\
\text{s.t.} & -4.5x_1 & - & 5x_2 & - & 4x_3 & - & x_4 & \leq & 4.5 \\
 & -40x_1 & - & 10x_2 & - & 6x_3 & & & \leq & 32.5 \\
 & 40x_1 & + & 10x_2 & + & 6x_3 & & & \leq & 55 \\
 & -1000x_1 & - & 1000x_2 & - & 1000x_3 & - & x_4 & \leq & -1000 \\
 & 1000x_1 & + & 1000x_2 & + & 1000x_3 & + & x_4 & \leq & 1000 \\
 & & & & & & & x_1, x_2, x_3, x_4 & \geq & 0
\end{array}$$

1.1.2 Dictionaries

We begin to study how to solve a linear program, that is, find an optimal solution, or assert its unboundness or infeasibility, and give a certificate. To do this, we store the linear program in a special form that can be easily manipulated. Consider the following linear program in standard form:

$$\begin{array}{lrcrcrcl}
\max & 3x_1 & + & x_2 & + & 2x_3 & & \\
\text{s.t.} & 2x_1 & + & 3x_2 & - & x_3 & \leq & 10 \\
 & x_1 & + & 5x_2 & + & x_3 & \leq & 15 \\
 & & & & & x_1, x_2, x_3 & \geq & 0
\end{array} \tag{1.2}$$

To obtain a dictionary, we introduce first a new variable called z, which represents the value of the objective. This variable is mainly conventional, and does not play a relevant role in the simplex method. It just eases the notations.

Then, for each constraint, we add a new variable, the *slack variable*, measuring how far the constraint is from being tight[1]. Formally, for a constraint $\sum_i a_i x_i \leq b$, we introduce the variable $x := b - \sum_i a_i x_i$. Hence the constraint can now be written:

$$\sum_i a_i x_i + x = b$$

[1]an inequality is *tight* for an assignment of its variables if it is satisfied with equality

Remark. Remember the iron foundry example. We did not introduce variables for the quantities of silicone and manganese in the final product, as it was not considered as decision variables. The significance of slack variables is actually how far we are from the upper or lower bound given in the statement of the problem, so they denote those quantities.

Remark. We will usually denote the slack variable by $x_{n+1}, \ldots, x_{n+m}$, where n is the number of original variables, and m is the number of constraints. This is to emphasize that in the simplex method, we do not make a difference between original variables and slack variables; they are treated in the exact same way. Nonetheless, in some proofs, it can be useful to make the distinction. In those cases, we will sometimes switch to the notation $s_1, \ldots, s_m$ for slack variables.

If we impose that the slack variable are non-negative, adding slack variables for each constraint does not change the set of optimal solutions (by projecting the solutions of the enhanced program to the original variables). Linear program (1.2) is equivalent to:

$$\begin{array}{rrcrcrcrcrcl} \max z = & 3x_1 & + & x_2 & + & 2x_3 & & & & & & \\ \text{s.t.} & 2x_1 & + & 3x_2 & - & x_3 & + & x_4 & & & = & 10 \\ & x_1 & + & 5x_2 & + & x_3 & & & + & x_5 & = & 15 \\ & & & & & & & \multicolumn{3}{r}{x_1, x_2, x_3, x_4, x_5} & \geq & 0 \end{array} \tag{1.3}$$

Sometimes, a linear program in this form is said to be in *canonical form.* By a slight rearrangement of these equations, and by neglecting the non-negativity constraints (which are implicit in dictionaries), we get:

$$\begin{array}{rcrcrcrcr} x_4 & = & 10 & - & 2x_1 & - & 3x_2 & + & x_3 \\ x_5 & = & 15 & - & x_1 & - & 5x_2 & - & x_3 \\ \hline z & = & & & 3x_1 & + & x_2 & + & 2x_3 \end{array} \tag{1.4}$$

The equations of (1.4) is what we call a dictionary, that is a representation of a linear program in a easy-to-use format. The following definition precises what we mean.

Definition 1.1.6. *A* dictionary *is a list of linear equations whose left-hand side (LHS) is made of a single variable that does not appear elsewhere, and z must be one of the LHS (by convention, in the last equation). The variable appearing in the right-had sides (RHS) are called* non-basic*, those appearing in the left-hand sides, except z, are called* basic.

Remark. Usually, the right-hand sides are given with the same order on the variables, and beginning by the constant.

Remark. Non-negativity constraints are implicit in dictionaries. Do not forget them!

Remark. The basic variables of a dictionary correspond to a subset of columns of the original matrix forming a basis. From the point of view of finding positive combinations, it means that we try to find a positive combination of the columns associated with basic variables, while keeping the other variables at zero. We know that there exists a (unique) linear combination, as long as the columns are linearly independant. but the coefficients are not generally non-negative.

What do we earn by changing our representation of linear programs to dictionaries? The (first) answer is that dictionaries make it clear that by fixing the value of some variables, the other can be deduced immediately. Indeed, if we choose the values of non-basic variables, the values of the basic variables (and the objective) can be computed straightforwardly from each equation. This leads us to define the following.

Definition 1.1.7. *A dictionary is* feasible *if the solution obtained by fixing the non-basic variables to zero is feasible. If a feasible solution can be obtained in this way from some dictionary, this solution is said to be* basic*.*

The dictionary of our example is thus feasible. Actually, if the RHS of the linear program in standard form are non-negatives, the dictionary obtained by our construction is feasible. The basic solution obtained just set every original variable to zero, which is obviously feasible under the previous condition.

Moreover, given a feasible dictionary equivalent to the linear program, we can get a lower bound on the maximum achievable by the LP. Indeed, the constant in the objective line of the dictionary is the objective reached by the basic solution corresponding to this dictionary.

The simplex method transforms a feasible dictionary into an equivalent feasible dictionary, and try to increase the lower bound given by the dictionaries. The hope is that the lower bound can be increased to the optimum, and then the basic solution given by the last dictionary would be optimal. For now, we forget the problem of finding a first feasible dictionary, this question will be address later.

1.1.3 Pivoting

Given a feasible dictionary, we want to improve it by finding an equivalent feasible dictionary, whose basic solution is closer to the optimum. The dictionary (1.4), with feasible solution $(0,0,0,10,15)$, gives us the formula $x = 3x_1 + x_2 + 2x_3$. This suggests that we could improve our solution by increasing the values of x_1, x_2 and x_3. It would even be great just to increase x_1, as for each unit x_1 is increased, the objective is increased by 3. Hence, we are looking for a new feasible solution by increasing x_1, while keeping x_2 and x_3 to their current values. Suppose we set $x_3 = t$, then we have:

$$\begin{array}{rcrcr} x_4 & = & 10 & - & 2t \\ x_5 & = & 15 & - & t \\ x_1 & = & & & t \end{array} \tag{1.5}$$

Remember now that in a dictionary, the non-negativity constraints are implicit. we must keep our variables non-negative, so t must be less than 5. By the way, choosing $t = 5$ will give a zero value to x_4. This means that we replace the non-basic variable x_1 by x_4. That is:

- x_1 changes from non-basic to basic,
- x_4 changes from basic to non-basic.

How to find a feasible dictionary whose basic solution is our new solution $(5,0,0,0,10)$? We must express x_1, x_5 and z depending on x_2, x_3 and x_4. We can use the line containing x_4 to get an expression of x_1 depending on the new non-basic variables:

$$x_1 = 5 - 1.5x_2 + 0.5x_3 - 0.5x_4 \tag{1.6}$$

From this, we can substitute x_1 in the lines of x_5 and z by using 1.6. It gives:

$$\begin{array}{rcrcrcrcr} x_5 & = & 15 & - & (5 - 1.5x_2 + 0.5x_3 - 0.5x_4) & - & 5x_2 & - & x_3 \\ x_1 & = & 5 & - & 0.5x_4 & - & 1.5x_2 & + & 0.5x_3 \\ \hline z & = & & & 3(5 - 1.5x_2 + 0.5x_3 - 0.5x_4) & + & x_2 & + & 2x_3 \end{array} \tag{1.7}$$

which gives after simplification:

$$\begin{array}{rcrcrcrcr} x_5 & = & 10 & + & 0.5x_4 & - & 3.5x_2 & - & 1.5x_3 \\ x_1 & = & 5 & - & 0.5x_4 & - & 1.5x_2 & + & 0.5x_3 \\ \hline z & = & 15 & - & 1.5x_4 & - & 3.5x_2 & + & 3.5x_3 \end{array} \tag{1.8}$$

To obtain the dictionary (1.8), we have only rewritten the line defining x_4 in (1.4), and add to the line corresponding to x_5 and to z one time and three times respectively the line of x_4 (the substitution formula). These operations being reversible, the new dictionary (1.8) is equivalent to the previous one (1.4). As expected, its basic solution is $(5, 0, 0, 0, 10)$ with an objective of 15.

By inspecting the objective line of (1.8), we find that the objective could be again increased by increasing the value of x_3. Once again, we can increase it as long as no other variable becomes negative, and we want to make at least one variable equal to zero to get a new non-basic variable. If we increase x_3 by one unit, x_1 is increased by half a unit, so won't become negative. x_5 though would be decreased by 1.5, and its current value is 10, hence we must increase x_5 by $\frac{20}{3}$. x_5 becomes non-basic, we use the equation containing x_5 in (1.8) to get a substitution formula for the new basic variable x_3. We get:

$$\begin{array}{rcrcrcrcr}
x_3 & = & 6.67 & + & 0.333x_4 & - & 2.33x_2 & - & 0.667x_5 \\
x_1 & = & 5 & - & 0.5x_4 & - & 1.5x_2 & + & \\
 & & +\,0.5(6.67 & + & 0.333x_4 & - & 2.33x_2 & - & 0.667x_5) \\
\hline
z & = & 15 & - & 1.5x_4 & - & 3.5x_2 & + & \\
 & & +\,3.5(6.67 & + & 0.333x_4 & - & 2.33x_2 & - & 0.667x_5)
\end{array} \tag{1.9}$$

After simplification, the new dictionary is:

$$\begin{array}{rcrcrcrcr}
x_3 & = & 6.67 & + & 0.333x_4 & - & 2.33x_2 & - & 0.667x_5 \\
x_1 & = & 8.33 & - & 0.333x_4 & - & 2.67x_2 & - & 0.333x_5 \\
\hline
z & = & 38.3 & - & 0.333x_4 & - & 11.7x_2 & - & 2.33x_5
\end{array} \tag{1.10}$$

Again, all the operations that we did are reversible, thus the new dictionary (1.10) is equivalent the the previous ones. Moreover, the objective line has a very special form: all the variables appears with a negative factor. It means that 38.3 (more exactly $\frac{115}{3}$) is an upper bound on the optimal solution, as each solution must satisfy the equation of dictionary (1.10). But the basic solution determined by (1.10) attains this value. So an optimal solution for our linear program is given by $(\frac{25}{3}, 0, \frac{20}{3}, 0, 0)$.

Remark. In this example, we may even conclude that this is the best solution, as it is the only possible solution with variables x_2, x_4 and x_5 having value 0. There could have been other optimal solutions if the coeffcients of some non-basic variables in the objective had been zero.

Remark. We gave the dictionaries with only three significant decimals. Algorithms used in practice work with floating point arithmetic, thus have also a limited number of decimals. This can cause errors in the accuracy of the result. We postpone the treatment of accuracy to a latter section.

We summarize what we did to improve our dictionaries. Each step, we did the following:

1. find a non-basic variable x_n with positive coefficient in the objective,
2. find a basic variable x_b which constraints the most the increase of the non-basic variable.
3. use the line containing x_b as a substitution formula for x_n,
4. substitute x_n in each other line of the dictionary.

These operations are refered as *pivoting* a dictionary. The simplex method is just successive pivoting until reaching the optimal solution (or possibly failing somewhere during the process).

Remark. There can be many variables with positive coefficients in the objective line. The simplex method does not specify which of them must be chosen. Similarly, there can also be many basic variables in the second step, and again we do not precise which one is chosen. The simplex method is not an algorithm, and we have some freedom in the choice of pivoting variables. A way to chose a variable deterministically is called a *rule.*

We give a general description of the simplex method. First, we distinguish the sets $\boldsymbol{x_N}$ and $\boldsymbol{x_B}$ of non-basic and basic variables, respectively. Then our original dictionary can be written as (in matricial notations):

$$\begin{array}{rcccc} \boldsymbol{x_B} & = & b & - & A\boldsymbol{x_N} \\ \hline z & = & z^* & + & c\boldsymbol{x_N} \end{array} \tag{1.11}$$

Let $x_j \in \boldsymbol{x_N}$ be a variable whose associated coefficient in is positive. If there is none, we claim that the solution which sets variables of $\boldsymbol{x_N}$ to 0 and $\boldsymbol{x_B} = b$ is optimal, which is clear from the fact that variables must be positive. Unfortunately, it is not clear at this point to prove that the dictionary is equivalent to the initial linear program (in the case of the dictionary has been obtained after many steps of computation). We will see later of to give a certificate of optimality from the optimal dictionary.

Definition 1.1.8. *If x_j is the non-basic variable chosen in the first step of pivoting, x_i is called the* entering variable *of the pivoting process.*

Entering refers to the fact that this variable will enter the basis.

Once an entering variable has been chosen, we must determine which variable will leave the basis. Let a_j be the column of A corresponding to the entering variable x_j. Each unit added to x_i changes the values of $\boldsymbol{x_B}$ by $-a$, but the change is limited by the values in b. Let I be the indices of the variables in $\boldsymbol{x_B}$. Hence a basic variable x_i, $i \in I$ imposes a constraint on the change of at most $\frac{b_i}{a_{ij}}$ if a_{ij} is positive (if a_{ij} is negative, there is no constraint as the value of the basic variable would be increased by the change of x_j). The maximal possible change for x_j is thus given by the minimum of the $\frac{b_i}{a_{ij}}$, $i \in I$ with a_{ij} positive. Let $i \in I$ attaining this minimum.

Definition 1.1.9. *If x_i is the basic variable chosen in the second step of pivoting, x_i is called the* leaving variable *of the pivoting process. The equation $x_i = b_i - a_i \boldsymbol{x_N}$ is known as the* pivot row*.*

Let A' be obtained from matrix A by removing row i and column j. The original dictionary can be rewritten with the good definitions:

$$\begin{array}{rcrcrcr} \boldsymbol{x'_B} & = & b' & - & A'\boldsymbol{x'_N} & - & a_{\bullet j}x_j \\ x_i & = & b_i & - & a_{i\bullet}\boldsymbol{x'_N} & - & a_{ij}x_j \\ \hline z & = & z^* & + & c'\boldsymbol{x'_N} & + & c_jx_j \end{array} \tag{1.12}$$

And the new dictionary is then obtained by substituting x_i by x_j everywhere except in the pivot row, which is just transform to have x_j isolated on the LHS. It means that the kth line, $k \neq i$, is obtained from the original kth line by adding $\frac{a_{kj}}{a_{ij}}$ times the line containing x_i, and the objective by adding $\frac{c_j}{a_{ij}}$. Note that by the choice of i, a_{ij} must be positive, and thus non-zero. We obtain:

$$\begin{array}{rcrcrcr} \boldsymbol{x'_B} & = & b' & - & A'\boldsymbol{x'_N} & & \\ & & -\,a_{ij}^{-1}a_{\bullet j}(b_i & - & a_{i\bullet}\boldsymbol{x'_N} & - & x_i) \\ x_j & = & a_{ij}^{-1}b_i & - & a_{ij}^{-1}a_{i\bullet}\boldsymbol{x'_N} & - & a_{ij}^{-1}x_i \\ \hline z & = & z^* & + & c'\boldsymbol{x'_N} & & \\ & & +\,a_{ij}^{-1}c_j(b_i & - & a_{i\bullet}\boldsymbol{x'_N} & - & x_i) \end{array} \tag{1.13}$$

which then can be simplified to:

$$\begin{array}{rcrcrcr} \boldsymbol{x'_B} & = & b' - \frac{b_i}{a_{ij}}a_{\bullet j} & - & (A' - a_{ij}^{-1}a_{\bullet j}a_{i\bullet})\boldsymbol{x'_N} & + & a_{ij}^{-1}a_{\bullet j}x_i \\ x_j & = & \frac{b_j}{a_{ij}} & - & a_{ij}^{-1}a_{i\bullet}\boldsymbol{x'_N} & - & a_{ij}^{-1}x_i \\ \hline z & = & z^* + \frac{c_jb_i}{a_{ij}} & + & (c' - \frac{c_j}{a_{ij}}a_{i\bullet})\boldsymbol{x'_N} & - & \frac{c_j}{a_{ij}}x_i \end{array} \tag{1.14}$$

This is clearly a dictionary with basic variables $(\boldsymbol{x_B} \setminus \{x_i\}) \cup \{x_j\}$. If the original dictionary was feasible, that is $b' \geq 0$ and $b_i \geq 0$, then the new dictionary is also feasible. To see this, we need to prove that $\frac{b_i}{a_{ij}} \geq 0$, and that:

$$b' - \frac{b_i}{a_{ij}}a_{\bullet j} \geq 0$$

The former follows from the non-negativity of b_i and a_{ij}. The latter follows from the choice of i; we chose i minimizing $\frac{b_i}{a_{ij}}$, which means that

$$b_k - \frac{b_i}{a_{ij}} a_{kj} \geq b_k - \frac{b_k}{a_{kj}} a_{kj} = 0$$

and so the constants of each line of the new dictionary are non-negative. Last, the objective of the basic solution has not been decreased. Indeed $\frac{c_j b_i}{a_{ij}}$ is non-negative, as c_j is non-negative and the two other factors are positive.

During the tranformation, all the operations that we do are replacing some line by its sum with a multiple of another line. As a consequence, these operations are reversible, and thus the set of feasible solutions is not altered. It also means that any line in a dictionary obtained after many pivoting step, can be obtain by a linear combination of lines of the original dictionary (or of any equivalent dictionary by the way). It is even quite easy to find those combinations: if $\sum_{i \in I} a_i x_i + \sum_{j \in J} a_j x_j = b$ is a line of the current dictionary, with I the indices of the basic variables of the original dictionary and $J = [\![1, n]\!] \setminus I$, then this line must have been obtained by adding a_i times the line of the original dictionary containing x_i, for each $i \in I$, as this is the only line containing x_i in the original dictionary.

What is perhaps more surprising is that every choice of basic variables leads to a unique dictionary. Indeed, suppose that we express the basic variable x_b as an affine combination of non-basic variables x_j, $j \in J$, in two different ways: $x_b = b - \sum_{j \in J} a_j x_j$ and $x_b = b' - \sum_{j \in J} a'_j x_j$. As the two systems of linear equations are equivalent, x_b must have the same value for every choice of $x_j, j \in J$, that is:

$$\sum_{j \in J} (a_j - a'_j) x_j = 0, \quad \text{for all choices of } x_j\text{'s}$$

and so $a_j = a'_j$ for each $j \in J$. We should state this as a lemma, saying that dictionaries are determined by their basis:

Lemma 1.1.10. *Two equivalent dictionaries with the same set of basic variables are equal.*

We conclude this section by giving a formal statement of the pivoting process.

Let $\boldsymbol{x_B} = b - A\boldsymbol{x_N}$, $z = z^* + \boldsymbol{c}\boldsymbol{x_N}$ be a feasible dictionary $\mathcal{D}$.

1. Choose a non-basic variable x_j whose coefficient in the objective line c_j is positive.
2. Choose a basic variable x_j, with $a_{ij} > 0$, minimizing $\frac{b_i}{a_{ij}}$.
3. Return the dictionary composed of the following lines:
    - for each line $k \neq i$ of $\mathcal{D}$, the sum of that line and $\frac{a_{kj}}{a_{ij}}$ times the line i,
    - the line i of $\mathcal{D}$ divided by a_{ij},
    - the sum of the objective line and $\frac{c_j}{a_{ij}}$ times the line i.

The new dictionary $\mathcal{D}'$ is a feasible dictionary equivalent to $\mathcal{D}$, with a constant in the objective line greater than or equal to the one of $\mathcal{D}$.

1.2 Termination of the simplex method

Once pivoting is defined, the simplex method is just the successive application of pivoting, until the objective has only negative terms (that is, the basic solution is optimal). But as we already know, some LPs do not have optimal solutions. What happens then? And will we ever reach the optimal solution if there is one? How can we build a first dictionary? We will address the question of optimality first, then the complexity, and then the initialization.

1.2.1 Optimality

There is obviously two ways how the pivoting may fail. The first is if there is no candidate variable for entering the basis, and we have already proved that this shows the optimality of the basic solution. By the way, we also have a not-so-complicated certificate for optimality (as long as we have a starting dictionary): by giving the optimal dictionary, it is easy to check the optimality of the solution, and we can also easily check the equivalence between this dictionary and the original dictionary (following a previous remark stating how the lines of a dictionary obtained after many pivoting relates to the original dictionary).

The second way the pivoting may fail is when no candidate is found for leaving the basis. It means that for each basic variable x_i, if x_j is the entering variable, a_{ij} is non-positive. Let $\boldsymbol{x}$ be the basic solution encoded by the dictionary. Let u be the vector obtained by setting every non-basic variable to 0, except x_j which is set to 1, and every basic variable x_i to $-a_{ij}$. Because the a_{ij} are non-positive, u has only positive coordinates. Then, every vector $\boldsymbol{x} + tu$ for $t \in \mathbb{R}^+$ is a feasible solution, as it is valid for the dictionary, and all its coordinates are non-negative. Moreover, the objective value for $\boldsymbol{x} + tu$ is $z^* + tc_j$, where c_j is positive. This proves the unboundedness of the linear program.

So when the pivoting fails, either we have a solution, or the program is unbounded and we have a proof of this fact.

Remark. Our certificate for unboundedness here is a feasible solution $\boldsymbol{x}$ plus a direction u. What can be the meaning of this direction in the original linear program? Each constraint stays satisfied by adding any positive factor of this direction. We deduce that:

- for an equality constraint $ax = b$, it implies $au = 0$,
- for a less-than constraint $ax \leq b$, we have $au \leq 0$,
- for a more-than constraint $ax \geq b$, $au \geq 0$.

Moreover, this direction must improve the objective, thus we need also to have $cu > 0$. If such an u exists, it is easy to see that if there is a feasible solution, the problem is unbounded.

1.2.2 Degeneracy

We turn to the question of the termination and the complexity of the simplex method. We already have two valuable tools to prove that the simplex method terminates. First, we know that the objective value achieved by the basic solutions is non-decreasing. Second, the Lemma 1.1.10 tells us that each dictionary depends only of its basis, which means that the simplex method has at most $\binom{n+m}{m}$ different states. To conclude, we would only need to prove that the objective value is increasing, that is, it is increased by a small but non-zero amount at each step.

Unfortunately, it is not the case. The objective may stay unchanged during a pivoting, and this phenomenon is called *degeneracy*. How can it happen? Remember that the idea of pivoting is to increase the objective value by increasing a non-basic variable. The leaving variable gives us the maximum possible increase that does not violate non-negativity constraints. So as long as this value is positive, the objective is also increased. The bad point is that this maximum increase $\frac{b_i}{a_{ij}}$ can be null, if b_i is zero. This happens if the basic solution has some basic variables with value 0. Such a dictionary is called *degenerate*. Let's see an example.

$$\begin{array}{rrcrcl}
\max & 10x_1 & + & x_2 & \text{s.t.} & \\
 & x_1 & - & x_2 & \leq & 2 \\
 & 4x_1 & + & x_2 & \leq & 10 \\
 & x_1 & + & 2x_2 & \leq & 6 \\
 & 2x_1 & + & x_2 & \leq & 6 \\
 & & & x_1, x_2 & \geq & 0
\end{array} \tag{1.15}$$

We build a feasible dictionary (as the RHS are all positive, it is easy):

$$\begin{array}{rcrcrcr}
x_4 & = & 2 & - & x_1 & + & x_2 \\
x_5 & = & 10 & - & 4x_1 & - & x_2 \\
x_6 & = & 6 & - & x_1 & - & 2x_2 \\
x_7 & = & 6 & - & 2x_1 & - & x_2 \\
\hline
z & = & & & 10x_1 & + & x_2
\end{array} \tag{1.16}$$

We can choose x_1 or x_2 as entering variable. Let's take x_2. Then, x_6 must leave the basis. It gives:

$$\begin{array}{rcrcrcr}
x_2 & = & 3 & - & 0.5x_1 & - & 0.5x_6 \\
x_4 & = & 5 & - & 1.5x_1 & - & 0.5x_6 \\
x_5 & = & 7 & - & 3.5x_1 & + & 0.5x_6 \\
x_7 & = & 3 & - & 1.5x_1 & + & 0.5x_6 \\
\hline
z & = & 3 & + & 9.5x_1 & - & 0.5x_6
\end{array} \tag{1.17}$$

Then x_1 enters. It happens now that the non-negativity of both x_5 and x_7 impose the most stringent bound on the increase of x_1. We choose to remove x_7. We get:

$$\begin{array}{rcrcrcr}
x_1 & = & 2 & - & 2.33x_7 & + & 0.33x_6 \\
x_2 & = & 2 & + & 0.33x_7 & - & 0.67x_6 \\
x_4 & = & 2 & + & x_7 & - & x_6 \\
x_5 & = & & & 2.33x_7 & - & 0.67x_6 \\
\hline
z & = & 22 & - & 6.33x_7 & + & 2.67x_6
\end{array} \tag{1.18}$$

The immediate consequence of having many choices for the leaving variable is that all the not-chosen possibilities have value 0 in the basic solution of the next dictionary. This leads to a bad effect in the following step. Here we must choose x_6 to enter and x_5 to leave, as x_5 is already to zero and x_6 appears negatively on its line.

Definition 1.2.1. *Basic solutions with a basic variable having a value* 0 *are called* degenerate. *Feasible dictionaries whose basic solution is degenerate, and simplex steps that do not change the basic solution are also called* degenerate.

The next dictionary is then:

$$\begin{array}{rcrcrcr} x_6 & = & & & 3.5x_7 & - & 1.5x_5 \\ x_1 & = & 2 & + & 0.5x_7 & - & 0.5x_5 \\ x_2 & = & 2 & - & 2x_7 & + & x_5 \\ x_4 & = & 2 & - & 2.5x_7 & + & 1.5x_5 \\ \hline z & = & 22 & + & 3x_7 & - & 4x_5 \end{array} \tag{1.19}$$

Thus, the objective value has not been increased during this step. Fortunately, in the next step, x_7 leaves, and x_7 appears positively in the line containing x_6, hence the objective will increase again. And we get the following dictionary, which is optimal:

$$\begin{array}{rcrcrcr} x_7 & = & 0.8 & - & 0.4x_4 & + & 0.6x_5 \\ x_6 & = & 4.8 & - & 1.4x_4 & + & 1.6x_5 \\ x_1 & = & 2.4 & - & 0.2x_4 & + & 0.7x_5 \\ x_2 & = & 0.4 & + & 0.8x_4 & + & 0.3x_5 \\ \hline z & = & 24.4 & - & 1.2x_4 & - & 2.2x_5 \end{array} \tag{1.20}$$

As we have seen, degeneracy occurs when there are more than one candidate variables for leaving the basis. In the example, we could have avoided degeneracy, by choosing x_1 as entering variable in the first step, or by choosing x_5 instead of x_7 as leaving variable during the third step (exercise: check that this would not have led to a degenerate step). This shows that degeneracy is dependant on the rules for choosing entering and leaving variable. However, sometimes, degenerate steps cannot be avoided, even by carefully choosing the entering and leaving variables. Here is an example:

$$\begin{array}{rcrcrcrcr} x_4 & = & 1 & - & x_1 & - & x_2 & + & x_3 \\ x_5 & = & 1 & - & x_1 & + & x_2 & - & x_3 \\ x_6 & = & 1 & + & x_1 & - & x_2 & - & x_3 \\ \hline z & = & & & x_1 & + & x_2 & + & x_3 \end{array} \tag{1.21}$$

By symmetry, it is equivalent to choose any entering variable, so let's take x_1. Similarly, among x_4 and x_5, we choose x_4 to leave the basis. We get the following dictionary:

$$\begin{array}{rcrcrcrcr} x_1 & = & 1 & - & x_4 & - & x_2 & + & x_3 \\ x_5 & = & & + & x_4 & + & 2x_2 & - & 2x_3 \\ x_6 & = & 2 & - & x_4 & - & 2x_2 & & \\ \hline z & = & 1 & - & x_4 & & & + & 2x_3 \end{array} \tag{1.22}$$

And then, the next step is degenerate.

Degeneracy is not a problem by itself, but it means that we cannot deduce the termination of the algorithm by the fact that there is only a finite number of possible state. Indeed, if the objective value does not change, it might happens that the simplex method cycles between them. And actually, in some cases, it does cycle.

1.2.3 Cycling

We mention an example of cycling, just to prove that it can happen, and then we will see how to avoid cycling. We start with the following dictionary, and then, apply the simplex method with the rules:
entering: we break ties by choosing the smallest subscript variable,
leaving: we choose the bottom-most possible line of the dictionary.

$$\begin{array}{rclcrcrcrcr} x_2 & = & 7x_3 & + & 3x_4 & - & 7x_5 & - & 2x_6 \\ x_1 & = & -2x_3 & - & x_4 & + & 3x_5 & + & x_6 \\ \hline z & = & 2x_3 & + & 2x_4 & - & 8x_5 & - & 2x_6 \end{array} \tag{1.23}$$

After the first iteration, choosing x_3 to enter the basis:

$$\begin{array}{rclcrcrcrcr} x_3 & = & -\frac{1}{2}x_1 & - & \frac{1}{2}x_4 & + & \frac{3}{2}x_5 & + & \frac{1}{2}x_6 \\ x_2 & = & -\frac{7}{2}x_1 & - & \frac{1}{2}x_4 & + & \frac{7}{2}x_5 & + & \frac{3}{2}x_6 \\ \hline z & = & -x_1 & + & x_4 & - & 5x_5 & - & x_6 \end{array} \tag{1.24}$$

After the second iteration, choosing x_2 to leave:

$$\begin{array}{rclcrcrcrcr} x_4 & = & -7x_1 & - & 2x_2 & + & 7x_5 & + & 3x_6 \\ x_3 & = & 3x_1 & + & x_2 & - & 2x_5 & - & x_6 \\ \hline z & = & -8x_1 & - & 2x_2 & + & 2x_5 & + & 2x_6 \end{array} \tag{1.25}$$

After the third iteration:

$$\begin{array}{rclcrcrcrcr} x_5 & = & \frac{3}{2}x_1 & + & \frac{1}{2}x_2 & - & \frac{1}{2}x_3 & - & \frac{1}{2}x_6 \\ x_4 & = & \frac{7}{2}x_1 & + & \frac{3}{2}x_2 & - & \frac{7}{2}x_3 & - & \frac{1}{2}x_6 \\ \hline z & = & -5x_1 & - & x_2 & - & x_3 & + & x_6 \end{array} \tag{1.26}$$

After the fourth iteration:

$$\begin{array}{rclcrcrcrcr} x_6 & = & 7x_1 & + & 3x_2 & - & 7x_3 & - & 2x_4 \\ x_5 & = & -2x_1 & - & x_2 & + & 3x_3 & + & x_4 \\ \hline z & = & 2x_1 & + & 2x_2 & - & 8x_3 & - & 2x_4 \end{array} \tag{1.27}$$

After the fifth iteration:

$$\begin{array}{rcrcrcrcr}
x_1 & = & -\frac{1}{2}x_2 & + & \frac{3}{2}x_3 & + & \frac{1}{2}x_4 & - & \frac{1}{2}x_5 \\
x_6 & = & -\frac{1}{2}x_2 & + & \frac{7}{2}x_3 & + & \frac{3}{2}x_4 & - & \frac{7}{2}x_5 \\
\hline
z & = & x_2 & - & 5x_3 & - & x_4 & - & x_5
\end{array} \tag{1.28}$$

After the sixth iteration:

$$\begin{array}{rcrcrcrcr}
x_2 & = & 7x_3 & + & 3x_4 & - & 7x_5 & - & 2x_6 \\
x_1 & = & -2x_3 & - & x_4 & + & 3x_5 & + & x_6 \\
\hline
z & = & 2x_3 & + & 2x_4 & - & 8x_5 & - & 2x_6
\end{array} \tag{1.29}$$

which is just the original dictionary. We can then go on applying the simplex method with these rules without ever reaching the optimal. But the optimal could have been reached in only one iteration, by choosing x_4 as entering variable. We would have obtained:

$$\begin{array}{rcrcrcrcr}
x_4 & = & -x_1 & - & 2x_3 & + & 3x_5 & + & x_6 \\
x_2 & = & -3x_1 & + & x_3 & + & 2x_5 & + & x_6 \\
\hline
z & = & -2x_1 & - & 2x_3 & - & 2x_5 & &
\end{array} \tag{1.30}$$

Actually, in this example, the third dictionary is symmetric to the first one, so we can see that it will cycle just by checking the two first iterations.

1.2.4 Bland's rule

Fortunately, cycling is not a fatality, there are rules that do not cycle. One of the simplest rule is Bland's rule:

entering: choose the smallest subscript variable with a positive coefficient in the objective lines.

leaving: choose among the candidate the variable with smallest subscript.

So Bland's rule is familiarly known as the smallest subscript rule.

Theorem 1.2.2 (Bland, 1977)**.** *The simplex method with Bland's rule always terminates.*

Proof. By contradiction, suppose that starting from such dictionary D_1, we obtain the dictionaries $D_2, \dots, D_k$, where D_{i+1} is the image of D_1 by a simplex step with Bland's rule, and $D_k = D_1$.

Among the variables, consider one, x_i, which is basic at least once, and non-basic at least once, in $D_1, \dots, D_k$, and choose it such that i is maximal. That is, every variable x_j with $j > i$ is either always basic, or always non-basic. We may assume that x_i leaves the basis between D_s and D_{s+1}, and enters the basis between D_t and D_{t+1} with $s < t$. We denote B_s the basis of D_s and B_t the basis of D_t, and we write D_s as:

$$\begin{array}{rcrcr} \boldsymbol{x_{B_s}} & = & b' & - & A'\boldsymbol{x_{N_s}} \\ z & = & z^* & + & c'\boldsymbol{x_{N_s}} \end{array} \tag{1.31}$$

and D_t as :

$$\begin{array}{rcrcr} \boldsymbol{x_{B_t}} & = & b'' & - & A''\boldsymbol{x_{N_t}} \\ z & = & z^* & + & c''\boldsymbol{x_{N_t}} \end{array} \tag{1.32}$$

let u be the vector obtained by increasing by one unit the entering variable x_j for the sth iteration, when x_i leaves the basis. That is:

$$u_k = \begin{cases} b'_k - a'_{kj} & \text{if } k \in B_s, \\ 1 & \text{if } k = j, \\ 0 & \text{otherwise.} \end{cases} \tag{1.33}$$

The objective value of this (unfeasible) solution is $z^* + c'_j$, and so by evaluating it with the objective line of D_t we have:

$$\sum_{k \in N_t} c''_k u_k = c'_j \geq 0 \tag{1.34}$$

which we unfold and rewrite as:

$$\sum_{k \in B_s - i} c''_k (b'_k - a'_{kj}) + c''_i (b'_i - a'_{ij}) \geq 0 \tag{1.35}$$

and then, because for a given k, $c''_k \neq 0$ and $b'_k \neq 0$ means that $k \in B_s \cap N_t$, implying that x_k is a leaving variable in some iteration of the cycle and so the kth coordinate of the basic solution of these dictionary is zero by degeneracy:

$$\sum_{k \in B_s \cap N_t - i} c''_k a'_{kj} + c''_i a'_{ij} \leq 0 \tag{1.36}$$

But then, by the choice of entering and leaving variables, c''_i and a'_{ij} are positive, and c''_k is non-positive for each valid k. And a'_{kj} is negative, because b'_k is zero and k was not chosen as a leaving variable for the sth iteration. Thus the previous inequality is a contradiction as the LHS is positive. $\square$

1.2.5 The perturbation method

We present here an alternative to Bland's rule. The idea is to completely avoid degeneracy, As we have seen in Section 1.2.2, this is not possible while keeping unchanged the linear program. But actually, degeneracy can only happen in some very special cases (even if they can be quite frequent in practice, due to the special structure of practical problems).

Degeneracy is by definition the appearance of a zero in the constant column of a dictionary. This happens when by summing lines of the initial dictionary, the constants cancel each other. The idea of the perturbation method is to slightly change the value of the RHS of the original linear program, hoping that in this case, the constants will not cancel each other anymore. If moreover we add only very very small coefficient, there is some chance that the optimal solutions will only be very very slightly different.

How to implement this idea? Remember Lemma 1.1.10 and the discussion preceding it: Given a line of a dictionary D' obtained after any number of pivoting from an initial dictionary, this line can be describe quite easily as a sum of the line of the original dictionary, just by looking at the coefficients of the original basic variables in our line. And so the constant is obtained by taking the sum of the constants of the first dictionary, with the same coefficient.

First, we can scale the linear program by a sufficently large constant (the gcd of all the denominators of all feasible dictionaries), such that every coefficient of a feasible dictionary is integral. Then let M be the largest possible coefficient that can occur in the description of a line as a linear combination of lines of the original dictionary. Define $\varepsilon = \min\{\frac{1}{2M}, 10^{-9}\}$. Let $\varepsilon_i = \varepsilon^i$ for $1 \leq i \leq m$, and add ε_i to the ith line of the first dictionary. Then, we prove that there are no more degeneracy, because for any linear combination of lines $y_1, \ldots, y_m$, we have that the constant obtained is:

$$b'_j = \sum_i y_i b_i + \sum_i y_i \varepsilon^i \tag{1.37}$$

By the choice of ε, we have that $|\sum_i y_i \varepsilon^i| < 1$ and $\sum_i y_i b_i$ is integral, thus $b'_j = 0$ implies that $\sum_i y_i b_i = 0$. Then, suppose that j the smallest coefficient with y_j non-null, we would have:

$$y_j = \sum_{i>j} y_i \epsilon^{i-j} \tag{1.38}$$

The right-hand side has absolute value less than one, while the left-hand side is integral, contradicting the definition of j. All the y_i's must be zero, which is never the case for a line of a dictionary.

Moreover, the effect on the objective is also quite small: for the same reason, the slack between the original optimal solution and the optimal solution obtained by perturbation is strictly less that $\frac{1}{2}$, so by rounding the solution we would get the optimal value of the original problem.

Now the trick is that we do not need to scale the system or to find ε, as we can just keep the ε_i's as symbols in the lines, because by their choice, they will never cancel each other. With symbols, this method is known as the lexicographic method. Actually, we do not even have to keep them as symbols, because as we mentionned, they are quite easy to find from any dictionary: the factor of ε_i in a line is the same as the factor of the ith basic variable of the original dictionary in the same line.

1.2.6 Complexity

At this point, we know that the simplex terminates, and that it can do at most $\binom{n+m}{m}$ iterations from a dictionary with m rows and n columns. But this value can be quite large: for a linear program with 100 variables and 100 constraints (which is considered as very small by current standards), this represents more than $9 \cdot 10^{58}$.

In practical applications, the number of iterations seems to depend linearly on m, which is quite better (see Chvátal's book for more details). When dealing with implementation questions, we will assume that the number of iterations is $O(m \log n)$, as this is what is empirically observed. But unfortunately, this bound is no true in theory. Actually, we can even find examples where the number of iteration is closer to the $\binom{n+m}{n}$ qualitatively, than to $m \log n$. These examples are dependant on the tie-breaking rules of the simple method. so we will only study an example for the *largest-coefficient* rule:

entering Choose the variable with largest coefficient in the objective line. (we will not need a rule for leaving variable as we will not have degeneracy)

The following example is due to Klee and Minty (1972). They proved that we need $2^n - 1$ iterations with the largest-coefficient rule to solve the following linear program:

$$\begin{array}{lrcll} \max & \sum_{j=1}^{n} 10^{n-j} x_j & & \text{subject to} & \\ & 2\left(\sum_{j=1}^{i-1} 10^{i-j} x_j\right) + x_i & \leq & 100^{i-1} & \text{for all } i \in [\![1, n]\!] \\ & x_i & \geq & 0 & \text{for all } i \in [\![1, n]\!] \end{array} \tag{1.39}$$

Theorem 1.2.3 (Klee, Minty)**.** *The simplex method with largest coefficient rule takes 2^{n-1} operations to solve the linear program* (1.39)*.*

Proof. We denote $s_1, \dots, x_n$ the slack variables. First, we prove that in any feasible dictionary, exactly one of s_i, x_i is basic, for each $i \in [\![1, n]\!]$. Indeed, by summing the ith equation with -10 times the $i-1$th equation we get:

$$\begin{aligned}100^{i-1} - 100^{i-2} &= x_i + s_i + 2\sum_{j<i} 10^{i-j} x_j \\ &\quad -10x_{i-1} - 10s_{i-1} - 20 \sum_{j<i-1} 10^{i-1-j} x_j \qquad (1.40)\\ &= x_i + s_i + 10x_{i-1} - 10s_{i-1} \qquad (1.41)\end{aligned}$$

As s_{i-1} is non-negative, and x_{i-1} is at most 100^{i-2} by the $i-1$th inequality, $x_i + s_i >= 100^{i-1} - 2 \cdot 100^{i-2}$ so one of them must be basic. As there are exactly n basic variables, the other one must be non-basic.

For a feasible dictionary, the basic variable are $\{x_i, i \in I\} \cup \{s_i, i \notin I\}$ for some set $I \subseteq [\![1, n]\!]$. Let's find the objective line of this dictionary. We know that it is a sum of the original objective with the lines of the original dictionary. As s_i is basic if $i \notin I$, it does not occur in the objective, so we must take a linear combination of the lines with indexes in I. So the objective can be written :

$$z = \sum_{j=1}^{n} 10^{n-j} x_j + \sum_{i \in I} \lambda_i \left(2\sum_{j=1}^{i-1} 10^{i-j} x_j + x_i + s_i - 100^{i-1} \right) \qquad (1.42)$$

Because x_i is basic when $i \in I$, its coefficient must be zero in (1.42), that is:

$$10^{n-i} + \lambda_i - 2\sum_{j>i} \lambda_j 10^{j-i} = 0 \qquad (1.43)$$

which we rewrite as:

$$\lambda_i = -10^{n-i} + 2\sum_{j>i} \lambda_j 10^{j-i} \qquad (1.44)$$

This is an upper triangular system, thus we can compute easily the values for the λ_i. Denoting $I = \{i_1 < i_2 \ldots < i_k\}$, we prove iteratively that:

$$\lambda_{i_j} = (-1)^{k+1-j} 10^{n-i_j} \qquad (1.45)$$

Indeed:

$$\begin{aligned}\lambda_{i_j} &= -10^{n-i_j} + 2\sum_{l=j+1}^{l} (-1)^{k+1-j} 10^{n-i_l} 10^{i_l - i_j} \\ &= -10^{n-i_j} + 2\sum_{l=j+1}^{l} (-1)^{k+1-j} 10^{n-i_j} \\ &= (-1)^{k+1-j} 10^{n-i_j}\end{aligned}$$

Note that λ_i is the coefficient of s_i in the objective line (1.42). We want to know the other coefficient. Let $i \notin I$, the coefficient of x_i is then:

$$\begin{aligned}
c_i &= 10^{n-i} + \sum_{j\in I, j>i} \lambda_j \\
&= 10^{n-i} + \sum_{l=k'}^{k} (-1)^{k+1-l} 10^{n-i_l} \cdot 2 \cdot 10^{i_l - j} \\
&= 10^{n-i} + \sum_{l=k'}^{k} (-1)^{k+1-l} 2 \cdot 10^{n-i} \\
&= \begin{cases} -10^{n-i} & \text{if } k-k'+1 \text{ is odd,} \\ 10^{n-1} & \text{otherwise.} \end{cases}
\end{aligned} \tag{1.46}$$

where k' is the smallest index such that $i_{k'}$ is bigger than i.

We remark the following:

- the absolute value of the coefficient for x_i or s_i is always 10^{n-i}, so we always choose the variable with positive coefficient and smallest index to enter,
- the sign of the ith variable depends on the parity of the number of non-basic slack variables with bigger indices only,
- if x_i enters, s_i leaves, and inversely if s_i enters, x_i leaves the basis, as exactly one of them is in the basis,
- thus if the non-basic variable with index i enters the basis, then the signs of all the coefficients of variables of indices $j \leq i$ is changed to their respective opposite, and the others are conserved.

Now, it is an easy exercise to prove that to get all variables appearing negatively from a dictionary where they all appears positively, we need exactly $2^n - 1$ iterations. $\square$

$2^n - 1$ is qualitatively not better than $\binom{n+m}{m}$: for $n = 100$, this is more than 10^{30}. So this Klee-Minty example is a bad news. One could hope that by choosing another rule, we could be able to prove a better bound on the maximal number of iterations. Unfortunately, it seems empirically that for every simple rule there is a bad example, like the Klee-Minty one. It is not known at present if there is a rule for the simplex method that only uses a polynomial number of iterations in the worst case.

Remark. In computer science, complexity of algorithms is measured as a function of the size of the problem. Informally, the size of the problem is the number of symbols in a written description of this problem. For a linear program with n variables, m constraints, this length is $O(nm)$ (as we usually use floating point values in computations, the size of a constant is the size of a float, so it is a constant and we do not mention it, but if we want to solve an LP with exact values, we should add the size of the constants). The complexity of an algorithm is the function which associates to each integer n the maximum running time of the algorithm over an instance of size n. This is refered as the worst-case complexity. Determining the complexity of an algorithm usually consists in giving an asymptotic upper bound for its complexity. Moreover, to be considered practical, an algorithm must have a complexity bounded by a polynomial in its size. With that point of view, the simplex method should be considered useless.

However, this classical definition of complexity only measures the worst-case scenario, while it may happen that most of the instances are significantly easier to solve than these hardest cases. In particular, problems from industrial applications have generally very special structures, that could make them easier to solve (we will study how to use one of the possible structures with the network simplex method). Moreover, the complexity is usually determined asymptotically which means that it is correct for *sufficiently large* instances, but it could be completely different for practical instances.

It happens that in the case of the simplex method, even if the theoretical complexity is exponential, it works very well in practice. In particular, it works significantly better than the equivalent polynomial algorithms.

Another teaching from complexity theory is that to be *easy* to solve, a problem must be easy to certify. That is, we can be able to give short proofs (or certificates) for the solution (in complexity terms, P $\subseteq$ NP $\cap$ coNP), whether it is positive or negative. In our case, it means that we would like to be able to give short proofs of optimality, unboundedness or infeasibility. We have already seen how to certify optimality (even if we will see a better way for that latter) and unboundedness (by giving a feasible solution and a direction of improvement).

1.3 Initialisation

Given a feasible dictionary, we can use the simplex method to give an optimal solution, or prove that the problem is unbounded. But how can we get a feasible dictionary, or prove that the problem is infeasible? The answer is easy: by applying the simplex method on a feasible dictionary! This is the two-phases simplex method

1.3.1 Two-phase simplex

Let $\max cx$ s.t. $Ax \leq b$ be a linear program, where b has negative coefficient. The idea is to try to find a solution that violates as least as possible the constraints of this program. One way to loosen the constraints is to add a positive value to the RHS of the program. So we want to find a solution for which we add the minimum value to the RHS, that is we must solve the following problem:

$$\begin{array}{rccc} \max & -x_0 & & \text{subject to} \\ & Ax & \leq & b + x_0 \end{array} \tag{1.47}$$

This problem has clearly a feasible solution, which consists in setting all variables to zero, save x_0 which is set to the absolute value of the minimum RHS constant.

Then we can write the corresponding dictionary:

$$\begin{array}{ccccccc} s & = & b & - & Ax & + & x_0 \\ \hline z & = & & & & - & x_0 \end{array} \tag{1.48}$$

This is still an infeasible dictionary. But we know that we can get a feasible dictionary by setting x_0 basic, while keeping all the other original variables non-basic. So we only have to pivot x_0 to get a feasible dictionary. If s_i is the leaving slack variable, that is b_i is minimum, we get:

$$\begin{array}{ccccccc} x_0 & = & -b_i & + & a_{i\bullet}x & - & s_i \\ s' & = & (b' - b_i) & - & (A' - \mathbf{1}a_{i\bullet})x & - & s_i \\ \hline z & = & b_i & - & a_{i\bullet}x & + & s_i \end{array} \tag{1.49}$$

As b_i is the minimal negative constant in the LHS of the linear program, this dictionay is indeed feasible. Now we use the simplex method to find the optimal solution. If it is zero, then we should get a feasible solution for the original linear program, while if it is negative, it was infeasible.

First, suppose we get a solution with value zero. Then x_0 has value 0. Hence x_0 should be non-basic, if we select x_0 as a leaving variable as soon as it is possible. We can now just remove the x_0 column to get the constraint lines of a dictionary for the original problem. All we have to do now is to find what is the objective line. For that, we substitute the basic variables of the original objective by using the constraint lines of the dictionary.

Next, suppose that we reach an optimal solution with value less than zero. The problem is then infeasible, and the final dictionary is a certificate of unfeasibility, as it is easy to show the equivalence between it and (1.48), and the equivalence between the fact that the original problem is feasible and the optimum of (1.47) is zero. Anyway, we will see a even simpler certificate latter.

This method consisting in a first phase where we find a feasible solution, then a second phase to compute the optimal solution, is called the *two-phase simplex.* To conclude this part, we present an example of computation of the two-phase simplex. We solve the following LP:

$$\begin{array}{lrcrcrcrcr} \max & 4x_1 & + & 3x_2 & - & x_3 & + & 5x_4 & & \text{s.t.} \\ & x_1 & & & + & 3x_3 & + & 2x_4 & \leq & 12 \\ & 2x_1 & + & x_2 & + & x_3 & + & 2x_4 & \leq & 10 \\ & -2x_1 & - & x_2 & - & x_3 & - & 2x_4 & \leq & -10 \\ & & & & & \multicolumn{3}{r}{x_1, x_2, x_3, x_4} & \geq & 0 \end{array} \tag{1.50}$$

The first dictionary for the first phase is:

$$\begin{array}{rcrcrcrcrcrcr} x_5 & = & -10 & + & 2x_1 & + & x_2 & + & x_3 & + & 2x_4 & + & x_0 \\ x_6 & = & 10 & - & 2x_1 & - & x_2 & - & x_3 & - & 2x_4 & + & x_0 \\ x_7 & = & 12 & - & x_1 & & & - & 3x_3 & - & 2x_4 & + & x_0 \\ \hline z & = & & & & & & & & & & - & x_0 \end{array} \tag{1.51}$$

As expected, it is infeasible, but we can make x_0 enter the basis. This corresponds to increasing the value of x_0 until a basic variable becomes null. It happens when x_0 reaches 10, and x_5 becomes non-basic. this gives:

$$\begin{array}{rcrcrcrcrcrcr} x_0 & = & 10 & + & x_5 & - & 2x_1 & - & x_2 & - & x_3 & - & 2x_4 \\ x_6 & = & 20 & + & x_5 & - & 4x_1 & - & 2x_2 & - & 2x_3 & - & 4x_4 \\ x_7 & = & 22 & + & x_5 & - & 3x_1 & - & x_2 & - & 4x_3 & - & 4x_4 \\ \hline z & = & -10 & - & x_5 & + & 2x_1 & + & x_2 & + & x_3 & + & 2x_4 \end{array} \tag{1.52}$$

If we choose x_2 as entering variable, we will be able to make x_0 leaving the basis, which means that we will reach the optimum for the auxiliary program:

$$\begin{array}{rcrcrcrcrcrcr} x_2 & = & 10 & - & x_0 & + & x_5 & - & 2x_1 & - & x_3 & - & 2x_4 \\ x_6 & = & & + & 2x_0 & - & x_5 & & & & & & \\ x_7 & = & 12 & + & x_0 & & & - & x_1 & - & 3x_3 & - & 2x_4 \\ \hline z & = & & - & x_0 & & & & & & & & \end{array} \tag{1.53}$$

This gives us a feasible solution for the original program: $(0, 10, 0, 0)$, and a feasible dictionary with the following lines;

$$\begin{array}{rcrcrcrcrcr}
x_2 & = & 10 & + & x_5 & - & 2x_1 & - & x_3 & - & 2x_4 \\
x_6 & = & & - & x_5 & & & & & & \\
x_7 & = & 12 & & & - & x_1 & - & 3x_3 & - & 2x_4 \\
\hline
z & = & 30 & + & 3x_5 & - & 2x_1 & - & 4x_3 & - & x_4
\end{array} \qquad (1.54)$$

The last line is obtained by taking the objective of (1.50), $4x_1+3x_2-x_3+5x_4$, and by replacing x_2 using the line of the dictionary containing x_2, which is $x_2 = 10 + x_5 - 2x_1 - x_3 - 2x_4$.

We may then apply the second phase of the simplex method. x_5 is clearly the only possibility for entering variable, and then x_6 must leave, and we get:

$$\begin{array}{rcrcrcrcrcr}
x_5 & = & & - & x_6 & & & & & & \\
x_2 & = & 10 & - & x_6 & - & 2x_1 & - & x_3 & - & 2x_4 \\
x_7 & = & 12 & & & - & x_1 & - & 3x_3 & - & 2x_4 \\
\hline
z & = & 30 & - & 3x_6 & - & 2x_1 & - & 4x_3 & - & x_4
\end{array} \qquad (1.55)$$

We can conclude this chapter with the following theorem:

Theorem 1.3.1. *For a linear program* $\max cx$ *s.t.* $ax \le b, x \ge 0$*, exactly one of these propositions is true:*

- *there is an optimal solution* x^**,*
- *it is infeasible,*
- *it is unbounded: there is a feasible solution* x_0 *and a vector* $u \ge 0$ *with* $cu > 0$*,* $Au \le 0$*.*

This is just a direct consequence of the two-phase simplex method. We will have to strengthen the two first cases to have nice certificates (we already have short certificates, but we can do better, this is the subject of the next chapter).

Chapter 2

Duality

2.1 Geometry

2.1.1 Why linear programming is a geometric problem

Until now, we have only manipulated linear equations and inequations in a purely algebraic way. But linear algebra is intimately related to Euclidian geometry. We develop in this chapter the geometric interpretation of the simplex method, and we will deduce from this point of view new certificates of optimality and infeasibility.

Let's start with some basic observations. Given a set of inequalities $Ax \leq b$ with n variables, we can represent solutions as vectors of a n dimensional vector space. Given a orthonormal basis $(e_1, \ldots, e_n)$, x is associated with the vector $\sum_{i=1}^{n} x_i e_i$ (we denote it x as well).

What is the significance of a constraint $ax \leq b$ (where a is a row and b a scalar)? First, recall that the set $H = \{x \mid ax = b\}$ is by definition an affine hyperplane of the vectorial space. This hyperplane separates the space in two parts: $\{x \mid ax > b\}$ and $\{x \mid ax < b\}$. Thus $ax \leq b$ is one of the two half-spaces defined by the hyperplane H. Hence a system $Ax \leq b$, with A an $m \times n$ matrix, and b a column vector of size m, represents the intersection of m half-spaces.

Definition 2.1.1. *A polyhedron is the intersection of finitely many closed affine half-spaces of a vectorial space. That is, a set $P := \{x \mid Ax \leq b\}$ for some matrix A and vector b.*

Solving a linear program is just finding a particular point in a polyhedron P. But which one? We want to maximize a linear form given by a row vector c of dimension n. It means that we want to be as far as possible from the origin, in the direction of c. One way of representing this is to take the hyperplane whose normal vector is c, that is, the set of point $cx = 0$, and to translate it as far as possible in the direction of c, while intersecting the polyhedron P.

Let's see this with pictures in dimension 2. In that case, a hyperplane is a line, a half-space is the set of point in one side of the line. Take the following linear program:

$$\begin{array}{rcrcl} \max & x & + & y & & \text{s.t.} \\ & 3x & - & y & \leq & 3 \\ & -x & + & 2y & \leq & 5 \\ & & & x, y & \geq & 0 \end{array} \tag{2.1}$$

The set of feasible solution can be represented as in Figure 2.1. It is the intersection of the two non-negativity constraint (the first quadrant), with the two half-spaces defined by the blue and red lines. That gives the polygonal purple surface.

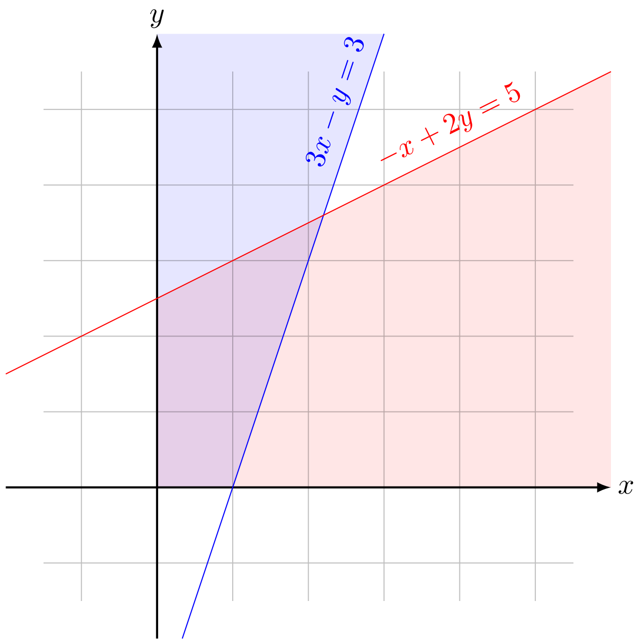


Figure 2.1: *The polyhedron defined by the inequalities of linear program* (2.1). *It is the intersection of four half-spaces, one by inequality.*

Among all the points of this polyhedron, we want to find one maximizing

$x + y$. Figure 2.2 shows the hyperplanes $x + y = d$, for $d = 0, 1, 3$.

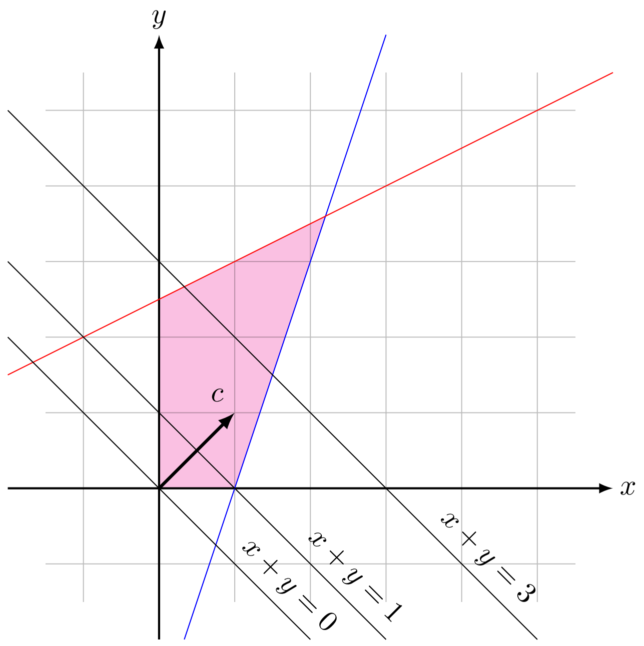


Figure 2.2: *Continuing Figure* (2.1)*, the three parallel black lines represents constant value lines for the objective function.*

From the picture, it is clear that the optimum will be reached by the intersection of the blue and the red lines. By solving the linear system of inequalities, we get the coordinates $(2.2, 3.6)$ of this point, and we deduce the optimum value 5.8. If we draw the line $x + y = 5.8$, the polyhedron of feasible solutions is completely on one side of this line, see Figure 2.3, proving that it is indeed the only optimal solution.

What is the significance of the simplex method geometrically? For that, we must find a geometrical object equivalent to dictionaries. Dictionaries are determined by a set of basic variables, and defined a basic solution. The basic solution is clearly a point of the feasible polyhedron. Non-basic variables define the inequalities that are tight: if x is a non-basic slack variable corresponding to inequality $ax \leq b$, then the basic solution is on the hyperplane $ax = b$. And if x is a non-slack non-basic variable, the basic solution is on the hyperplane $x = 0$. Hence, each of the n non-basic variable defines a hyperplane on which the basic solution lies. As we are

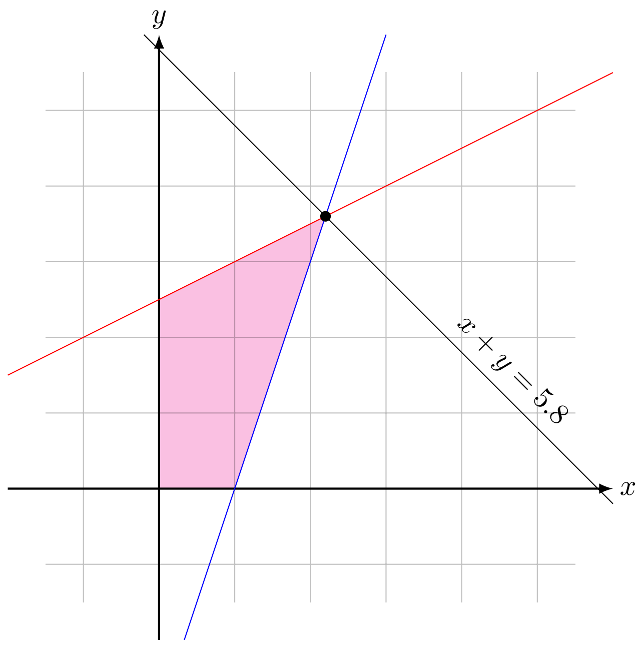


Figure 2.3: *The feasible solutions are all on the same side of the black line, hence any feasible solution has value at most* 5.8*. The intersection of the black line and the polyhedron is thus the only optimal solution.*

in a vector space of dimension n (when we consider the non-slack variables only), the intersection of n independant affine hyperplane contains at most one point, which must be the basic solution (and reciprocally, the existence of the basic solution proves that the affine hyperplanes are independant). And the points defined by the intersection of n constraints are the vertices of the polyhedron. This is the main characteristic of the simplex method: *the simplex method deals with the vertices of the feasible polyhedron of a linear program.*

What does a pivot operation mean? We change from one vertex to another. But more precisely what is the relation between these two vertices? A pivot makes one non-basic variable entering the basis, which means that we had a point on some hyperplane, and we leave that hyperplane. And there is a basic variable becoming non-basic, which means on the contrary that we are going to an other hyperplane. Finally, all the other non-basic variables stay non-basic, hence we stay on all the other hyperplanes in which is our first vertices. This means that we are moving along the edge incident to the first vertex, until reaching a new constraint defining a new vertices. *We are moving along the edges of the polyhedron.*

This is true as long as there is no degeneracy. Degeneracy happens when a vertex of the polyhedron intersects more than n supporting[1] hyperplane. As we only need n hyperplane to describe the solution, we have many different possibilities, and a degenerate step is just changing the representation to another one.

We illustrate these ideas by continuing the graphical example of linear program (2.1). We start with the solution $(0,0)$, the intersection of the lines corresponding to the non-negativity constraints. Then, we must replace one of the two constraints by another one. We can replace the non-negativity of y, it corresponds to increasing the value of y in the dictionary, and a displacement along the y-axis, until another constraint becomes tight, here $-x + 2y \leq 5$. The second slack variable, associated with this inequality, becomes non-basic. We obtain from the dictionary:

$$\begin{array}{rcrcrcr} s_1 & = & 3 & - & 3x & + & y \\ s_2 & = & 5 & + & x & - & 2y \\ \hline z & = & & & x & + & y \end{array}$$

[1]An affine hyperplane supports a polyhedron if the polyhedron is contained in one of the two closed half-spaces defined by the hyperplane, and intersects the hyperplane.

to the dictionary:

$$\begin{array}{rcrcrcr} y & = & 2.5 & + & 0.5x & - & 0.5s_2 \\ s_1 & = & 5.5 & - & 2.5x & - & 0.5s_2 \\ \hline z & = & 2.5 & + & 1.5x & - & 0.5s_2 \end{array}$$

whose basic solution is $A = (0, 2.5)$. We can then replace the non-negativity of x by the first constraint of the LP. It means that we increase the value of x in the dictionary. We can do this until we reach the line corresponding to first constraint, reaching the basic solution $Z = (2.2, 3.6)$ of the following dictionary:

$$\begin{array}{rcrcrcr} x & = & 2.2 & - & 0.4s_1 & - & 0.2s_2 \\ y & = & 3.6 & - & 0.2s_1 & - & 0.6s_2 \\ \hline z & = & 5.8 & - & 0.6s_1 & - & 0.8s_2 \end{array}$$

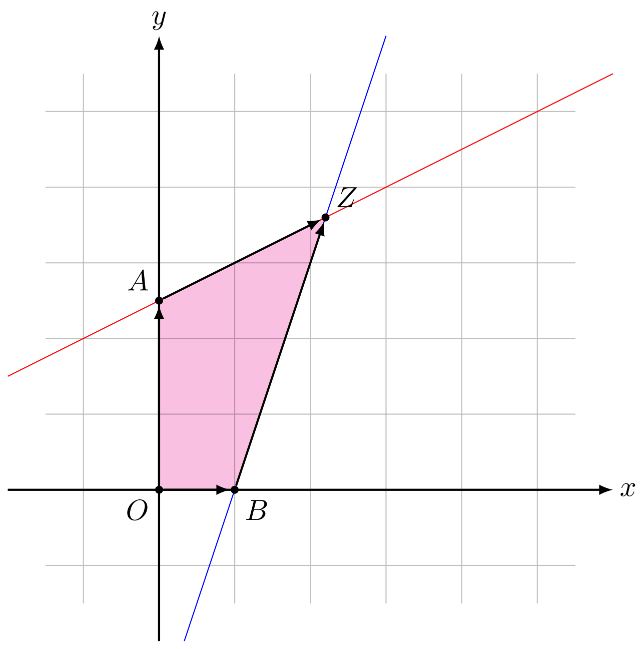


Figure 2.4: *The simplex method consists in travelling from vertices to vertices along the edges of a polyhedron, in the direction given by the objective function.*

During the first iteration, we could have follow the x-axis instead of the

y-axis, this would have given the dictionary, with basic solution $B = (1, 0)$:

$$\begin{array}{rcrcrcr}
x & = & 1 & - & 0.33s_1 & + & 0.33y \\
s_2 & = & 6 & - & 0.33s_1 & - & 1.67y \\
\hline
z & = & 1 & - & 0.33s_1 & + & 1.33y
\end{array}$$

We summarize these considerations in Figure 2.4.

Degeneracy is the fact of having a vertex of a polyhedron that can be described in different ways as the intersection of tight constraints. Recall the example (1.15):

$$\begin{array}{rrcrcrl}
\max & 10x & + & y & \text{s.t.} & & \\
 & x & - & y & \leq & 2 & (a) \\
 & 4x & + & y & \leq & 10 & (b) \\
 & x & + & 2y & \leq & 6 & (c) \\
 & 2x & + & y & \leq & 6 & (d) \\
 & & & x_1, x_2 & \geq & 0 &
\end{array}$$

After the third iteration, we got a basic solution described by the intersection of the tight constraints corresponding to the third and fourth original inequalities:

$$\begin{array}{rcrcrcr}
x & = & 2 & - & 2.33s_d & + & 0.33s_c \\
y & = & 2 & + & 0.33s_d & - & 0.67s_c \\
s_a & = & 2 & + & s_d & - & s_c \\
s_b & = & & & 2.33s_d & - & 0.67s_c \\
\hline
z & = & 22 & - & 6.33s_d & + & 2.67s_c
\end{array}$$

Let's look at a picture of the situation, in Figure 2.5. The degenerate dictionary happens when we are at point A, as three different constraints have their supporting hyperplane containing this point. It means that A can be described as the intersection of (b) and (c), of (b) and (d), or of (c) and (d). Here we have a description in terms of the (c) and (d), as the non-basic variable are x_c and x_d. But the next point is the intersection of (a) and (b). We can only exchange one constraint for another one during a pivot, hence we need two pivots here to get to the new point: the first one just changes the representation of A to one having x_b non-basic. The second goes to the optimal point Z.

In that special example, the degeneracy is due to a useless constraint: (d). We could remove (d) without changing the set of feasible solutions.

Definition 2.1.2. *A constraint of a system of inequalities is* redundant *if for every vector satisfying all the other inequalities, this constraint is also satisfied.*

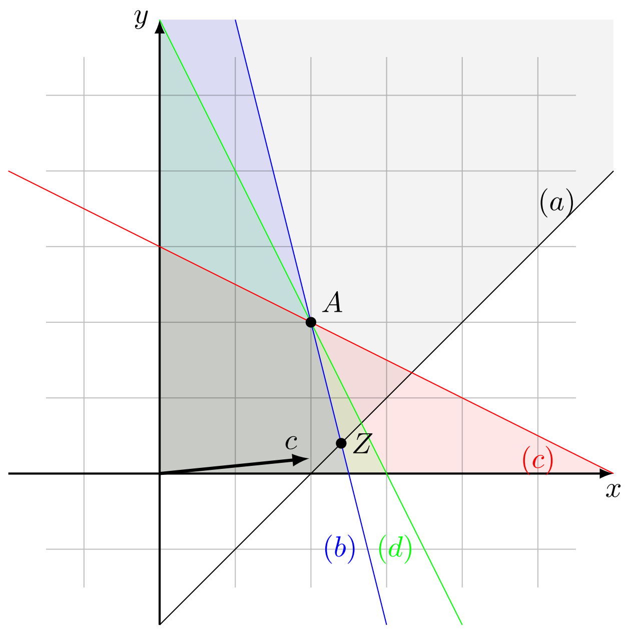


Figure 2.5: *An illustration of degeneracy.*

In dimension 2, degeneracy happens only when there are redundant constraints. This is not true in higher dimension, for example, the dictionary (1.21) for which we showed that degeneracy must happen during the resolution, correspond to the polyhedron of Figure 2.6, which has no redundant constraints.

$$\begin{array}{lrcrcrcl}
\max & x_1 & + & x_2 & + & x_3 & & \text{s.t.} \\
 & x_1 & + & x_2 & - & x_3 & \leq & 2 \\
 & x_1 & - & x_2 & + & x_3 & \leq & 2 \\
 & -x_1 & + & x_2 & + & x_3 & \leq & 2 \\
 & & & & & x_1, x_2, x_3 & \geq & 0
\end{array}$$

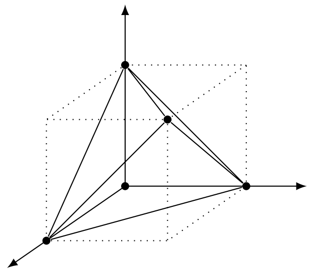


Figure 2.6: *Degeneracy can be a consequence of having too many facets containing a single vertex of the polyhedron: here some vertices are the intersection of* 4 *facets.*

$$\begin{array}{lrcrcr}
\max & 2x & + & 2y & \text{s.t.} & \\
 & -x & + & 2y & \leq & 0 \\
 & -2x & + & y & \leq & -3 \\
 & 3x & - & y & \leq & -2 \\
 & & & x, y & \geq & 0
\end{array}$$

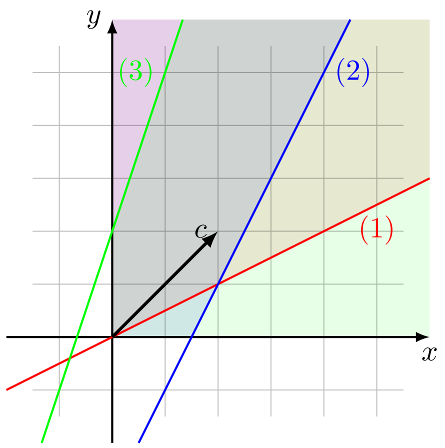


Figure 2.7: *This linear program is clearly unbounded. If we apply the simplex method, we will find the feasible solution* $(0, 2)$ *plus the direction* $(1, 3)$*, or the feasible solution* $(2, 1)$ *and the direction* $(1, 2)$*.*

To conclude this part, we mention an example where the linear program is unbounded, in Figure 2.7.

2.1.2 Fundamental theorem

The reader should now be convinced that the simplex method, and more generally linear programming, are strongly related to geometry. The following theorem will be our main tool for the geometrical study of polyhedron and linear programming:

Theorem 2.1.3 (Farkas 1894, Minkowsky 1896, Carathéodory 1911, Weyl 1935)**.** *Let $a_1, a_2, \ldots, a_n$ be column vectors of dimension m, and b a column vector of dimension m. Then, exactly one of the following is true:*

(i) *either b is a non-negative combination of linearly independant vectors of $a_1, \ldots, a_n$,*

(ii) *or there is a vector c with $ca_i \leq 0$ for all $1 \leq i \leq n$, and $cb > 0$. Moreover, c can be chosen such that $ca_i = 0$ for $d-1$ different values of i, where $d = \text{rank}\{a_1, \ldots, a_n, b\}$.*

Example: Condition (i) states that b is a conic combination[2] of independant vectors $a_1, \ldots, a_n$. The important part is that we do not need more vectors than d. Condition (ii) says that there is a hyperplane, with normal vector c, such that b is in one side of the hyperplane, while $a_1, \ldots, a_n$ are on the other side.

Figure 2.8 gives an example of both conditions.

Proof. First, the two conditions are not compatible, as the existence of c implies that if $\lambda_i \geq 0$ for all $i \in [\![1, n]\!]$, then $c \cdot (\sum_{i=1}^{n} \lambda_i a_i) \geq 0$ while $cb < 0$, discriminating b from $\sum_{i=1}^{n} \lambda_i a_i$.

If b is not a linear combination of $a_1, \ldots, a_n$, then any hyperplane containing the vectorial space generated by $a_1, \ldots, a_n$ would provide a normal vector c satifying statement (ii). We consider now that b is in the vectorial space generated by $a_1, \ldots a_n$, hence $b = \sum_{i \in I} \lambda_i a_i = b$, with $|I| = d = \text{rank}\{a_1, \ldots, a_n\}$.

We describe the following procedure:

[2]By definition, a *conic combination* is a non-negative combination.

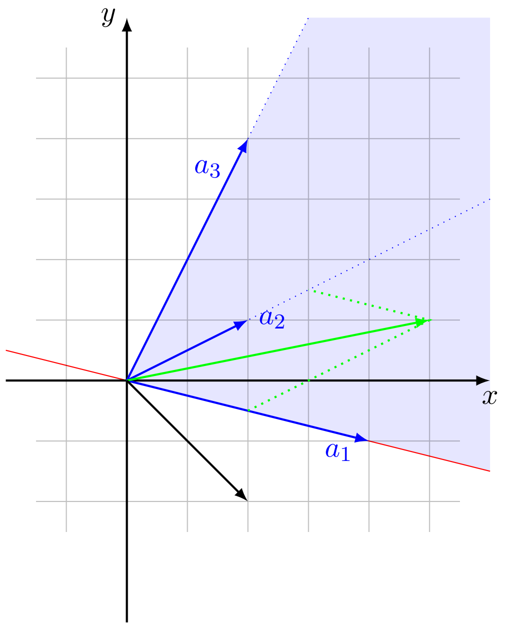


Figure 2.8: *The vectors* a_1, a_2, a_3 *generate the cone in blue. If a vector is inside the cone (like the green one), it is a non-negative combination of* linearly independant *vectors (here,* $0.5a_1 + 1.5a_2$*). If a vector is outside the cone, there is a hyperspace separating it from the cone. Here, the black vector is separated by the red line, containg* $2-1$ *vectors among* a_1, a_2, a_3.

If $\lambda_i \geq 0$ for all $i \in I$, then statement (i) is checked. Otherwise, we choose a minimal $j \in I$ with $\lambda_i < 0$. Let $c \neq 0$ be the projection of a_j on the orthogonal space of $\text{vect}\{a_i, i \in I - j\}$. That is $c \cdot a_i = 0$ for all $i \in I - j$, and $c \cdot a_j > 0$, and then $c \cdot b = \lambda_j < 0$.

Then, if $c \cdot a_k \geq 0$ for all $k \in [\![1, n]\!]$, statement (ii) is satisfied. Else, there is a vector a_i, $i \in [\![1, n]\!]$ with $c \cdot a_i < 0$. a_i is not in $\text{vect}\{a_k, k \in I - j\}$. We choose i minimal. Hence b is a linear combination $\sum_{k \in I-j+i} \gamma_k a_k$. We start again the procedure with this new decomposition of b.

To conclude the proof, it is sufficient to prove the termination of the previous procedure. As there are finitely many choices for the set I, if the procedure fails to terminate, there is a cycle $I_0, I_1, \ldots, I_k = I_0$, such that I_{l+1} is obtained from I_l by one step of the procedure. Let l be the biggest index that is in some I_s, but not in some I_t, choose s and t such that i is in I_{t+1} but not in I_{s+1}. Denote λ_k, i, j, c the values taken by the variables during the procedure applied to I_s, and λ'_k, i', j', c' the values during the procedure applied to I_t. Thus we have $j = l$ and $i' = l$. Then:

$$0 > c' \cdot b = c' \left(\sum_{k \in I_s} \lambda_k a_k \right) = \sum_{k \in I_s, k < l} \lambda_k c` a_k \geq 0 \tag{2.2}$$

where the first inequation comes from the choice of c. The second equality is a consequence of the fact that if $l < k \in I_s$, then $k \in I_t$ by the choice of l, and then $c' a_k = 0$ by the choice of c'. Then the last inequality of (2.2) follows as l is the smallest index with $c' a_k < 0$ when applying the procedure on I_s. This is a contradiction, hence the procedure terminates. $\square$

Remark. The procedure in the proof is just a disguised form of the simplex method with Bland's rule. It is not completely constructive, as we need to find an initial decomposition for b, but it can easily be obtained using basic linear algebra.

Many beautiful geometric results follow from this theorem. We state some of them.

Definition 2.1.4. *A* polytope *is the convex hull of a finite set of points in $\mathbb{R}^n$. That is,*

$$P = \{\sum_{i=1}^{k} \lambda_i x_i \mid \forall i \in [\![1, k]\!], \lambda_i \geq 0, \sum_{i=1}^{k} \lambda_i = 1\}$$

It happens that polytopes are a special forms of polyhedron: they are bounded polyhedron. The converse is also true.

Theorem 2.1.5 (Minkowsky 1896)**.** *P is a polytope iff P is a bounded polyhedron.*

The proof of this theorem is not easy, and we need several other results to get it. We introduce some definitions about cones.

Definition 2.1.6. *A set C is a* cone *if for each pair of vectors $x, y \in C$ and any non-negative value t, $x + ty \in C$.*

Definition 2.1.7. *A cone C is* finitely generated *if there are finitely many vectors $x_1, \ldots, x_n$ such that $C = \{\sum_{i=1}^{n} \lambda_i x_i \mid \lambda \geq 0\}$. A set C is a* polyhedral cone *if it is a cone and it is the intersection of finitely many linear half-spaces $C = \{x \mid Ax \leq 0\}$.*

Theorem 2.1.8 (Farkas, Minkowsky, Weyl)**.** *A cone is finitely generated iff it is polyhedral.*

Proof. We prove it for full-rank cones only. Let C be a finitely generated cone, generated by vectors $a_1, \ldots, a_m$. For any subset $\{a_i, i \in I\}$ of $n-1$ linearly independant vectors, there is exactly one hyperplane $H = \{x, cx = 0\}$ containing these vectors. Consider these hyperplanes for which the vectors $a_1, \ldots, a_m$ are on the same side $\{x, cx \leq 0\}$, this defines $c_1, c_2, \ldots, c_k$. Note that k is bounded as there are only finitely many ways to choose $n-1$ linearly independant vectors. Then, Theorem 2.1.3 states that $x \notin C$ implies $c_i x < 0$ for some i, which means that $P := \{x \mid c_i x \leq 0, 1 \leq i \leq k\}$ is contained in C. Conversely, any vector v in C can be written as $v = \sum_{i=1}^{n} \lambda_i a_i$, with $\lambda \geq 0$. As $c_j a_i \leq 0$ for all j, $c_j v \leq 0$.

Now, let C be a polyhedral cone, $C = \{x \mid c_i x \leq 0, 1 \leq i \leq n\}$. Then consider the cone generated by $c_1^T, \ldots, c_n^T$, by the first part of the proof, it is a polyhedral cone $\{x \mid a_i x \leq 0, 1 \leq i \leq k\}$. We prove that C is generated by $a_1^T, \ldots, a_k^T$. First note that a_j^T is in C, as $c_i a_j^T = a_j c_i^T \leq 0$ for all i. Suppose there is a vector x in C, but not generated by $a_1^T, \ldots, a_k^T$. Then by Theorem 2.1.3, there is a vector b such that $b a_i^T \leq 0$ for all i, but $bx > 0$. Then b^T is in the cone generated by $c_1^T, \ldots, c_n^T$, that is $b^T = \sum_i \gamma_i c_i^T$, and then $bx = \sum_i \gamma_i c_i x \leq 0$, contradiction. Thus, C is finitely generated. □

Transforming a result about cones into a result about polyhedron is quite easy: Let $Ax \leq b$ be a system of inequalities defining a polyhedron P. We add a new non-negative variable λ, and study the system $Ax - \lambda b \leq 0, \lambda \geq 0$.

This is clearly a polyhedral cone, thus a finitely generated cone, generated by some vectors $u_1, \ldots, u_n, v_1, \ldots, v_p$, where the u_i's have λ-value 0, and the v_i's have λ-value 1 (by scaling). Then, the original polyhedron is just the intersection with the hyperplane $\lambda = 1$. Thus, any vector of P can be decomposed as:

$$\sum \lambda_i u_i + \sum \gamma_i v_i, \quad \text{with } \sum_i \gamma_i = 1, \gamma \geq 0, \lambda \geq 0$$

This proves half of the following famous theorem, that implies Theorem 2.1.5:

Theorem 2.1.9 (Minkowsky-Weyl)**.** *A set P is a polyhedron iff $P = C + Q$ where C is a polyhedral cone and Q is a polytope.*

Proof. It remains to prove that the sum of a polytope and a cone is a polyhedron. We use the same idea: add a new variable. There are vectors $u_1, \ldots, u_n, v_1, \ldots, v_m$ such that $x \in C + Q$ iff there are $\lambda, \gamma \geq 0$, $\sum \gamma_i = 1$ and

$$x = \sum_i \lambda_i u_i + \sum_j \gamma_j v_j$$

We denote $u_i' = \binom{u_i}{0}$ and $v_i' = \binom{v_i}{1}$. Then $x \in P + C$ iff

$$\begin{pmatrix} x \\ 1 \end{pmatrix} = \sum_i \lambda_i u_i' + \sum_i \gamma_i v_i'$$

The cone generated by the u_i''s and v_i''s can be written as $\{\binom{x}{\alpha} \mid Ax - \alpha b \leq 0\}$ by Theorem 2.1.8, and so $C + Q = \{x \mid Ax = b\}$ is a polyhedron. $\square$

We will not use directly the Minkowsky-Weyl theorem, but it gives us many ideas about what we are doing when we solve linear programs, as can be seen in Figure 2.9. The next section develops the nature of polyhedra.

Another example where a result on cone leads to a result on polytope is the following:

Theorem 2.1.10 (Carathéodory)**.** *If v is in the cone generated by $a_1, \ldots, a_m$, then v is a non-negative combination of* $\operatorname{rank}\{a_1, \ldots, a_m\}$ *vectors among $a_1, \ldots, a_m$. If v is in a polytope of dimension n generated by k points, then v is a convex combination of only $n + 1$ of these points.*

The proof is left as an exercise.

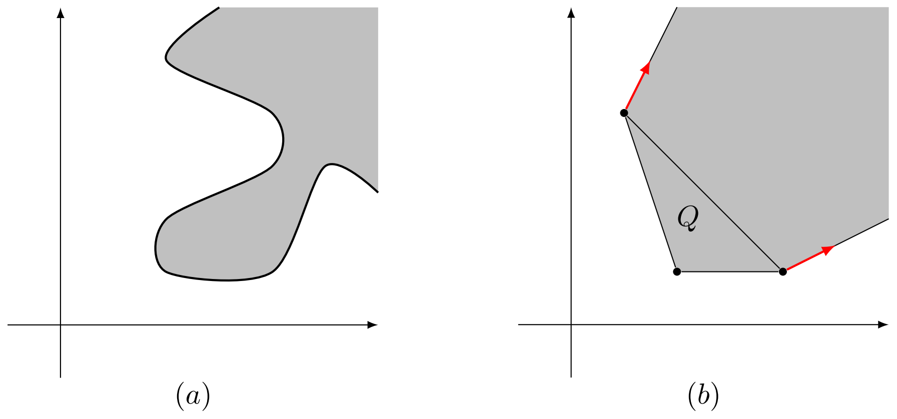


Figure 2.9: *Feasible set of a linear program before Minkowsky-Weyl* (a)*, and after Minkowsky-Weyl* (b)*. In the latter, we have vertices, facets, extremal rays for the cone,... The black polyhedron is the sum of the inner triangle, convex hull of three vertices, and the cone generated by the red vectors.*

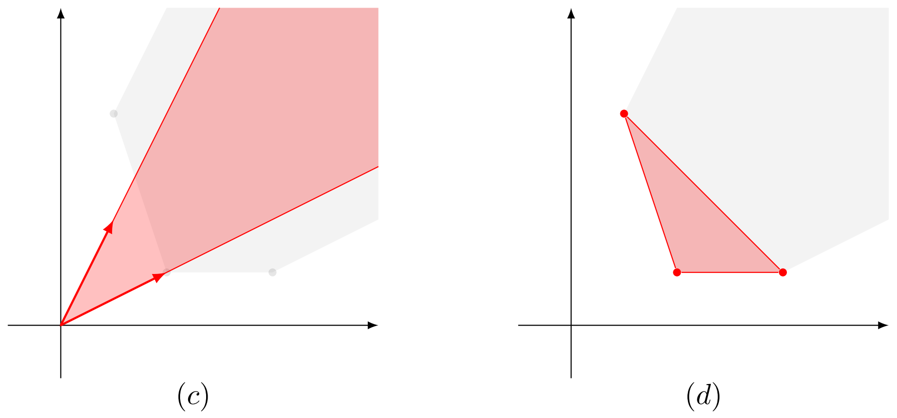


Figure 2.10: *The Minkowsky decomposition of the polyhedron of Figure 2.9,* (a)*. In red, its characteristic cone* C *in the left, and the polytope* Q *in the right.*

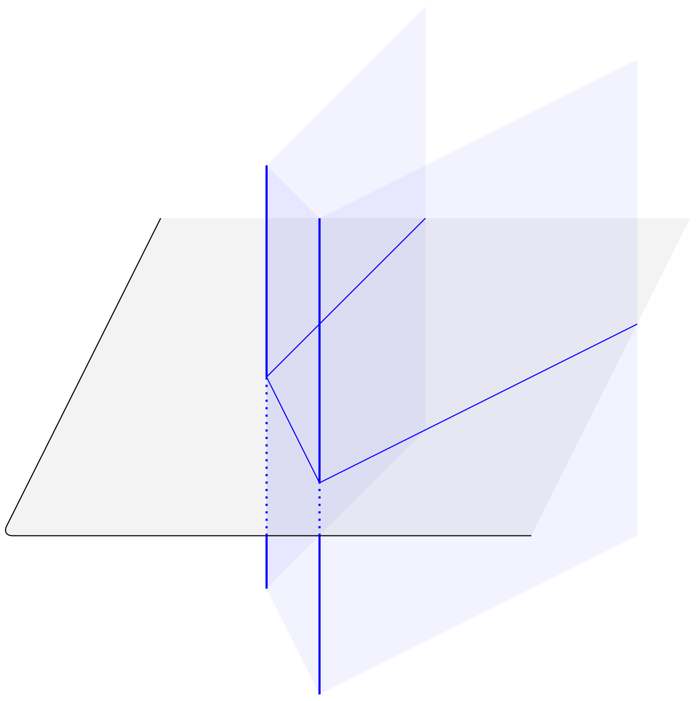


Figure 2.11: *A polyhedron in blue, with three facets and no vertices, its lineality space is the* y*-axis* $\{(0, \lambda, 0), \lambda \in \mathbb{R}\}$.

2.1.3 The structure of polyhedra

We know that a polyhedron is decomposable into the sum of a cone and a polytope. We call *characteristic cone* C (or *recession cone*) of the polyhedron $P = \{x \mid Ax \leq b\}$ the set $\{x \mid \forall y \in P, x + y \in P\}$. This is exactly the cone of the Minkowsky-Weyl theorem. In particular, if we find that a linear program is unbounded, we get a direction of unboundedness u. This vector u must be in the characteristic cone of the polyhedron associated to the linear program. If there is a vector u in the characteristic cone with $cu > 0$, then the program consisting in maximizing cx over P is unbounded. It implies that if for every vector u in the characteristic cone, $cu \leq 0$, then the linear program cannot be unbounded.

If the characteristic cone contains a vector $x \neq 0$ and its opposite $-x$, then it contains the subspace $\{\lambda x, \lambda \in \mathbb{R}\}$. The *lineality space* of P is $\{x \in C \mid -x \in C\} = \{x \mid Ax = 0\}$. If the dimension of the lineality space is 0, then P may have vertices (or P is empty). For example, for $P = \{(x, y) \mid x + y \leq 1\}$ has a lineality space $L = \{(\lambda, -\lambda), \lambda \in \mathbb{R}\}$.

The polyhedron P is not always of full-rank. An inequation $ax \leq b_i$ from $Ax \leq b$ is *implicit* if for every feasible solution x, $ax = b_i$. The set of implicit inequalities defines an affine subspace in which lie P, this subspace is called the affine hull of P.

A face of P is a set $F = \{x \mid Ax \leq b, cx = \mu\}$, where μ is the maximum of the linear program $\max cx$ s.t. $ax \leq b$, when this maximum is finite. By definition, a face is a polyhedron, and $F \subseteq P$. P is also a face of P (take $c = 0$). A face of a face of P is a face of P.

If the lineality space has dimension 0, then the minimal faces of P have dimension 0, that is they are points. Such points are called *vertices*. Then, in the Minkowsky-Weyl decomposition Q is the convex hull of the vertices of P. The optimum of a linear program will be reached by a face of P, hence by a minimal face.

Maximal faces distinct from P are called facets. If $Ax \leq b$ has no redundant inequality, then facets are in correspondance with inequalities $ax \leq b_i$. We will prove later that the number of faces (and hence facets, vertices) of a polyhedron is finite.

2.1.4 Farkas' Lemma

Another consequence of Theorem 2.1.3 is a certificate for proving that a system of inequalities is not feasible.

Corollary 2.1.11 (Farkas' lemma)**.** *The linear system $Ax = b$, $x \geq 0$ is infeasible if and only if there is a vector $y \in \mathbb{R}^m$ such that $yA \geq 0$ and $yb < 0$.*

Proof. The necessity is obvious: if such a y exists, then for any $x \geq 0$, $y(Ax) = (yA)x \geq 0$, and $yb < 0$ implies that $AX \neq b$. For the sufficiency, if the system $Ax = b$, $x \geq 0$ has no solution, it means that b is not in the cone genrated by the column of A, then by Theorem 2.1.3, there is a vector y with $yb < 0$ and $yA \geq 0$. □

We are more interested by the following version

Corollary 2.1.12. *The linear system $Ax \leq b$, $x \geq 0$ is infeasible if and only if there is a vector $y \in \mathbb{R}^m_+$ such that $yA \geq 0$ and $yb < 0$.*

Proof. By adding slack variables, $Ax \leq b$, $x \geq 0$ is equivalent to $Ax - \hat{x} = b$, $x, \hat{x} \geq 0$. Then by Farkas' lemma, this is infeasible if and only if there is a vector y with $yA \geq 0$, $y \geq 0$ and $yb < 0$. □

Remark. We are making progress toward a nice certificate for infeasibility: we only have to find a non-negative combination of the lines of a linear program (this is what yA means), such that the left-hand-side has only non-negative coefficient, and the right-hand side is negative. But we still need to explain how to find this combination. One way to do so would be to use the fact that we gave a constructive proof for Theorem 2.1.3. We will show later how to obtain y directly from the first phase of the two-phase simplex method.

We state a last variant of Farkas' lemma, before giving a geometric interpretation. All thse variants are equivalent.

Corollary 2.1.13. *The linear system $Ax \leq b$ is infeasible if and only if there is a vector $y \in \mathbb{R}^m_+$ such that $yA = 0$ and $yb < 0$.*

Proof. $Ax \leq b$ is equivalent to $A(x - \hat{x}) \leq b$, $x, \hat{x} \geq 0$. Then we can apply the previous variant of Farkas' lemma. □

Remark. Moreover, if there is a vector y^*, then we can also choose any vector solution of the system $yA = 0, yb = y^*b$, By Carathéodory's theorem, y can then be chosen with at most $d + 1$ non-zero coefficient, where d is the rank of the rows of A. This is actually valid for all the variants of Farkas' lemma.

Example: Geometric interpretation of Farkas' lemma.

Consider the inequalities drawn in Figure 2.12, that can be written:

$$\begin{aligned}(X - A) \cdot \vec{u} &\geq 0\\ (X - B) \cdot \vec{v} &\geq 0\\ (X - C) \cdot \vec{w} &\geq 0\end{aligned}$$

Farkas' lemma says that if the system is infeasible, we can scale the normal vectors $\vec{u}$, $\vec{v}$ and $\vec{w}$ by positive coefficient, such that their sum is $\vec{u}+\vec{v}+\vec{w} = 0$, and the sum of the constant terms is positive, *i.e.* $A \cdot \vec{u} + B \cdot \vec{v} + C \cdot \vec{w} > 0$. In that case, by summing the three inequalities, we get:

$$X \cdot (\vec{u} + \vec{v} + \vec{w}) + A \cdot \vec{u} + B \cdot \vec{v} + C \cdot \vec{v} \geq 0$$

The first term is zero, the three last terms sum to a negative number (yb), this inequality is a contradiction.

When a system of inequalities is infeasible, there is a subset of at most $d + 1$ inequalities that is also infeasible (by the previous remark using Carathéodory's theorem 2.1.10). Then, we can scale the normal vectors

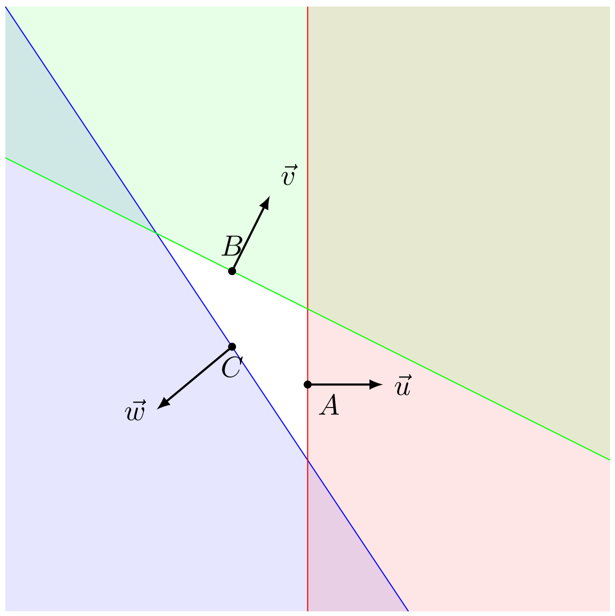


Figure 2.12: *An infeasible system of three inequations, with normal vectors for each of the hyperplane.*

of the hyperplane so that their sum is zero (the scaling vector is y). Moreover, by taking a point D that violates all the inequalities (one in the white zone in Figure 2.12), we have:

$$\begin{array}{rcl} (D-A)\cdot\vec{u} & < & 0 \\ (D-B)\cdot\vec{v} & < & 0 \\ (D-C)\cdot\vec{w} & < & 0 \end{array}$$

By summing these inequalities, we get:

$$D\cdot(\vec{u}+\vec{v}+\vec{w}) - A\cdot\vec{u} - B\cdot\vec{v} - C\cdot\vec{w} < 0$$

By removing the first term, we get $-yb > 0$.

2.2 Weak and strong duality

2.2.1 Looking for an upper bound

Thanks to Farkas' lemma, we know how to prove that a system of linear inequalities is infeasible. We would like to apply this result to check the optimality of a solution. This leads to do the following: for a linear program $\max cx$ subject to $Ax \leq b$, $x \geq 0$, suppose that we have a solution of value k. Then, the following system of inequalities in infeasible iff k is the maximum:

$$Ax \leq b, x \geq 0, cx > k$$

Unfortunately, there is a strict inequality in this system, and we do not know how to handle them. Anyway, for any k we can ask if there is a solution of value at least k, *i.e.* is the following system feasible:

$$Ax \leq b, x \geq 0, -cx \leq -k$$

To this problem we can apply Farkas' lemma: there is a solution of value at least k, or there is a vector y and a real z with the following properties:

$$\begin{array}{rl} y, z \geq & 0 \\ yA - zc \geq & 0 \\ yb - zk < & 0 \end{array}$$

By scaling y and z, we can suppose that $z = 1$, and with some rewriting, it means that we want a vector $y \geq 0$ such that $yA \geq c$ and $yb = k$.

Actually, any vector $y \geq 0$ with $yA \geq c$ gives us an upper bound yb on the solution: to see this, recall that y in this context is the coefficients of a positive combination of the lines of the linear program. The constraints tell us that the resulting inequation *dominates* c (every coefficient is greater in yA than in c), and as x must be positive, $yAx \geq cx$ (in other words, this is the easy direction of Farkas' lemma). But the right-hand side that we obtain is yb, so $cx \leq yAx \leq yb$.

Example: Consider the linear program:

$$\begin{array}{rrcrcrcr} \max & x_1 & + & 2x_2 & + & 3x_3 & \text{subject to} & \\ & & + & x_2 & + & 3x_3 & \leq & 21 \\ & 2x_1 & + & x_2 & + & x_3 & \leq & 13 \\ & & & & & x_1, x_2, x_3 & \geq & 0 \end{array}$$

If we multiply by three the second constraint, we get:

$$x_1 + 2x_2 + 3x_3 \leq 6x_1 + 3x_2 + 3x_3 \leq 39$$

This gives us an upper bound of 39 on the solution. We could be smarter by just adding the two first lines:

$$x_1 + 2x_2 + 3x_3 \leq 2x_1 + 2x_2 + 3x_3 \leq 34$$

This time, we have an upper bound of 34. Actually, we can get an even better upper bound by adding $\frac{1}{2}$ times the first constraint with $\frac{3}{2}$ times the second one:

$$x_1 + 2x_2 + 3x_3 \leq 3x_1 + 2x_2 + 3x_3 \leq 30$$

Is that the best possible? By using the simplex method, we find an optimal solution $(0, 9, 4)$, whose value is 30.

How can we find the best possible solution that can be obtained by summing inequalities of the original problem? As y represents a *linear combination* of the original constraints, it is natural to express this problem as a linear program, whose objective is to minimize the upper bound. Hence, we want to solve the following problem:

$$\begin{array}{rrcl} \min & yb & \text{subject to} & \\ & yA & \geq & c \\ & y & \geq & 0 \end{array} \qquad \text{(D)}$$

We are lucky, this program is indeed linear, so we know how to solve it with the simplex method.

Definition 2.2.1. *Given a linear program:*

$$\begin{array}{rrcl} \max & cx & & \textit{subject to} \\ & Ax & \leq & b \\ & x & \geq & 0 \end{array} \tag{P}$$

the program (D) *is known as the* dual *program of* (P)*. By opposition,* (P) *is called the* primal *program. Objectives, variables, constraints are refered as* primal *or* dual*, depending on the program on which they appear.*

Computing the dual.

As it is not always convenient to rewrite to program in standard form to compute the dual program, it is useful to know some simple rules to compute the dual of any linear program directly.

First, as the dual represents combination of constraints of the primal, the variable of the dual are associated to the constraints of the primal, hence we choose one variable for each constraint of the primal. As an exception, the non-negativity constraints or non-positivity constraints are dealt by another way, so they do not have associated dual variables.

How do we know if a dual variable is free, non-negative or non-positive? Again, the idea is to sum inequalities, so they must all have the same sign. Thus,

- if a variable is associated with a less-than constraint, this variable is taken non-negative,
- if a variable is associated with a greater-than constraint, it is taken non-positive,
- if a variable is associated with an equality constraint, it is taken free.

Then, for each variable x of the primal, we have an associated constraint C, whose right-hand side is the scalar appearing in the term of the primal variable in the primal objective. The left-hand side is given in the following way. There is one term for each constraint of the primal, except non-negativity or non-positivity, the dual variable y is the one associated with that constraint. The coefficient of y is the same as the coefficient of the primal variable x in the primal constraint associated to y. Simpler: the coefficient of y in the dual constraint associated with x is the coefficient of x in the primal constraint associated to y.

The sign of a dual constraint depends on the positivity of the associated primal variable:

- if x is free, the dual constraint is an equality,
- if x is non-negative, the sign of the dual constraint is a greater-than,
- if x is non-positive, the sign of the dual constraint is a less-than.

This choice is determined by the fact that we want an upper bound of the objective: we want that for each term $c_i x_i$ in the objective, $c_i x_i \leq c'_i x_i$ for all possible vector x satisfying the positivity constraints. If we do not know the sign of x_i, then we must impose $c_i = c'_i$, thus the corresponding constraint is an equality. If x_i is non-negative, then $c'_i \geq c_i$ is enough, and similarly for x_i non-positive.

Finally, the objective of the dual is just given by the right-hand side of the primal. We summarize these observations in the following array:

Primal	Dual
max	min
min	max
$a_{i\bullet}x \leq b_i$	$y_i \geq 0$
$a_{i\bullet}x = b_i$	y_i free
$a_{i\bullet}x \geq b_i$	$y_i \leq 0$
$x_j \geq 0$	$ya_{\bullet j} \geq c$
x_j free	$ya_{\bullet j} = c$
$x_j \leq 0$	$ya_{\bullet j} \leq c$

From this array, it is clear that the symmetries in the rule for computing a dual implies that the dual of the dual program is nothing else than the primal program:

Proposition 2.2.2. Taking the dual is an involution: the dual of the dual of a program P is P.

As a consequence, we will say that two programs are dual if one is the dual of the other.

Remark. About the term *duality*: in linear algebra, duality corresponds to the equivalence between vectors and linear forms $x \mapsto cx$. Both objects are determined by n scalars; the coordinates of the vector, and the coefficients of c for the linear form. A hyperplane is the inverse image of 0 by a linear form, and thus in some sense duality is also a correspondance between vectors and hyperplanes. The dual program is just about manipulating the hyperplanes supporting the polyhedron of the primal program, to get a new hyperplane "parallel" to the objective linear form. (the dual of $\max cx$ s.t. $ax \leq b$ is $\min yb$ s.t. $yA = c, y \geq 0$, so we want to get exactly the objective function). We will see geometric examples later.

Minkowsky-Weyl Theorem 2.1.9 is a typical example of a duality result: the same object (a polyhedron) can be described by conjonction of inequalities, or by combination of vectors. Farkas' Lemma (and the fundamental theorem) is also a duality result: a vector is described as a non-negative combination of given vectors, or there is a separating hyperplane.

2.2.2 Weak duality

We summarize our discussion in the following lemma:

Lemma 2.2.3. *Let* $\max cx$ *s.t.* $Ax \leq b, x \geq 0$ *and* $\min yb$ *s.t.*$yA \geq c, y \geq 0$ *be a pair of feasible dual linear programs. Then, for any solutions* x *and* y *to these programs,* $cx \leq yb$*.*

Proof.

$$cx \leq (yA)x = y(Ax) \leq yb \tag{2.3}$$

□

Remark. The main consequence of weak duality is that if we have solutions x and y with $cx = yb$ then both x and y are optimal to their respective programs.

2.2.3 Strong duality

The most important result of this chapter for us is the following result. It is due to Von Neumann (1947), Gale, Kuhn and Tucker (1951):

Theorem 2.2.4 (The duality theorem of linear programming)**.** *Let* A *be a* $m \times n$ *matrix,* $b \in \mathbb{R}^m$*,* $c \in \mathbb{R}^n$*. Then,*

$$\max\{cx \mid Ax \leq b, x \geq 0\} = \min\{yb \mid yA \geq c, y \geq 0\} \tag{2.4}$$

as long as the two linear programs in (2.4) *are feasible.*

Proof. By Lemma 2.2.3, if both programs are feasible, any pair of solutions x^* and y^* checks $cx^* \leq y^*b$, proving that the program are bounded, and that the maximum is less than the minimum in (2.4), so we only have to check the reverse inequality. We want to prove the existence of vectors x and y such that:

$$\begin{array}{rcrcl} Ax & & & \leq & b \\ & & yA & \geq & c \\ cx & - & yb & \leq & 0 \\ & & x,y & \geq & 0 \end{array} \tag{2.5}$$

By Farkas' Lemma, the system (2.5) is feasible iff for all vectors u, v, λ, $ub - cv \geq 0$ whenever:

$$\begin{array}{rcrcrcl} uA & & & + & \lambda c & \geq & 0 \\ & & -Av & - & b\lambda & \geq & 0 \\ & & & & u,v,\lambda & \geq & 0 \end{array} \tag{2.6}$$

Let u, v, λ satisfying inequations (2.6).

- if $\lambda = 0$, then :
$$ub - cv \geq u(Ax^*) - (y^*A)v = (uA)x^* + y^*(-Av) \geq 0$$
- if $\lambda \geq 0$, by scaling we can choose $\lambda = 1$, and then we have $uA \geq c$, $Av \leq b$ and $u, v \geq 0$. Thus, u and v are feasible dual and primal solution. by weak duality we have $ub - cv \geq 0$.

This proves the feasibility of (2.5), and hence the theorem follows. □

Remark. We state and prove the duality theorem for a standard program, but it can easily be extended to any kind of primal-dual pair of linear programs.

That is good! It means that we can find very clear and very simple optimality certificates, by solving the dual program. The dual solution gives us how to combine the inequations of the primal to get a bound on the primal program, and this bound reach the primal optimum. The only bad point is that we need to compute another simplex, to get this dual solution. Do we really need actually? It is time to give a second proof of the duality theorem, only in terms of dictionaries:

Proof. (dictionary version)

If both programs are optimal, then the primal has an optimal solution that can be found by the simplex method. Hence we get an optimal dictionary, where the objective line is:

$$z = z^* - \sum_i \lambda_i x_i - \sum_i \mu_i s_i$$

where the s_i's are the slack variables, and $\lambda, \mu \geq 0$. This line has been obtained by adding to the original objective line $z = cx$, the original lines of the variables. More exactly, the ith line, with slack variable s_i, has been added exactly μ_i times. Consider the line obtained by adding μ_i times the ith line, for each i, to the empty line $0 = 0$, we get:

$$\sum_i \mu_i s_i = \sum_i \mu_i b_i - \sum_j (\sum_i \mu_i a_{ij}) x_j$$

But this is exactly the difference between the final objective line and the original objective line, thus we have that:

$$\sum_i \mu_i b_i = z^* \tag{2.7}$$

and for each $j \in [\![1, n]\!]$,

$$\lambda_j = \sum_i \mu_i a_{ij} - c_j \geq 0$$

and so:

$$\sum_i \mu_i a_{ij} \geq c_j \tag{2.8}$$

Combining inequations (2.7) and (2.8), we have that μ is a dual solution with value z^*, proving the theorem. $\square$

This second proof gives us a very simple way to get the dual solution, only by looking at the last dictionary of the simplex method on the primal program: the dual solution is given by the coefficient of the slack variables in the objective lines, multiplied by -1. Here is the example (1.2) taken from the first chapter:

$$\begin{array}{rrcrcrcr}
\max & 3x_1 & + & x_2 & + & 2x_3 & & \\
\text{s.t.} & 2x_1 & + & 3x_2 & - & x_3 & \leq & 10 \\
 & x_1 & + & 5x_2 & + & x_3 & \leq & 15 \\
 & & & & & x_1, x_2, x_3 & \geq & 0
\end{array}$$

First, we write the dual program:

$$\begin{array}{rrcrcl}
\min & 10y_1 & + & 15y_2 & \text{subject to} & \\
 & 2y_1 & + & y_2 & \geq & 3 \\
 & 3y_1 & + & 5y_2 & \geq & 1 \\
 & -y_1 & + & y_2 & \geq & 2 \\
 & & & y_1, y_2 & \geq & 0
\end{array}$$

The final dictionary was:

$$\begin{array}{rclcrcrcr}
x_3 & = & \frac{10}{3} & + & \frac{1}{3}x_4 & - & \frac{7}{3}x_2 & - & \frac{2}{3}x_5 \\
x_1 & = & \frac{25}{3} & - & \frac{1}{3}x_4 & - & \frac{8}{3}x_2 & - & \frac{1}{3}x_5 \\
\hline
z & = & \frac{115}{3} & - & \frac{1}{3}x_4 & - & \frac{35}{3}x_2 & - & \frac{7}{3}x_5
\end{array}$$

The dual solution is then $y_1 = \frac{1}{3}$, $y_2 = \frac{7}{3}$. We can check that it is indeed a solution of the dual by checking all the inequalities:

$$2\frac{1}{3} + \frac{7}{3} = 3 \geq 3$$

$$3\frac{1}{3} + 5\frac{7}{3} = \frac{38}{3} \geq 1$$

$$-\frac{1}{3} + \frac{7}{3} = 2 \geq 2$$

It gives a dual value of $\frac{1}{3}10 + \frac{7}{3}15 = \frac{115}{3}$, hence the solutions are optimal. This can also be seen directly by combining the inequations of (1.2):

$$\frac{1}{3}(2x_1 + 3x_2 - x_3) + \frac{7}{3}(x_1 + 5x_2 + x_3) = 3x_1 + \frac{38}{3}x_2 + 2x_3$$

$$\frac{1}{3}10 + \frac{7}{3}15 = \frac{115}{3}$$

And thus the following inequation is implicit in the primal:

$$3x_1 + \frac{38}{3}x_2 + 2x_3 \leq \frac{115}{3}$$

Hence we are done.

What happens when one of the two programs is not feasible or unbounded? From the proofs, if a linear program has an optimal solution, then its dual also have an optimal solution. Then there are at most 3 cases:

- both programs are unbounded. Actually, this case is not possible by the weak duality: we would be able to choose a feasible solution for the maximization problem as large as we want (say positive), and for the minimization problem as small as we want (say negative), this would give a contradiction.
- both programs are unfeasible, for example:

$$\begin{array}{rrcrcr}
\max & x & + & y & \text{s.t.} & \\
 & x & & & \leq & 1 \\
 & -x & & & \leq & -3 \\
 & & & x,y & \geq & 0
\end{array}
\qquad
\begin{array}{rrcrcr}
\min & u & + & 3v & \text{s.t.} & \\
 & u & - & v & \geq & 1 \\
 & & & 0 & \geq & 1 \\
 & & & u,v & \geq & 0
\end{array}$$

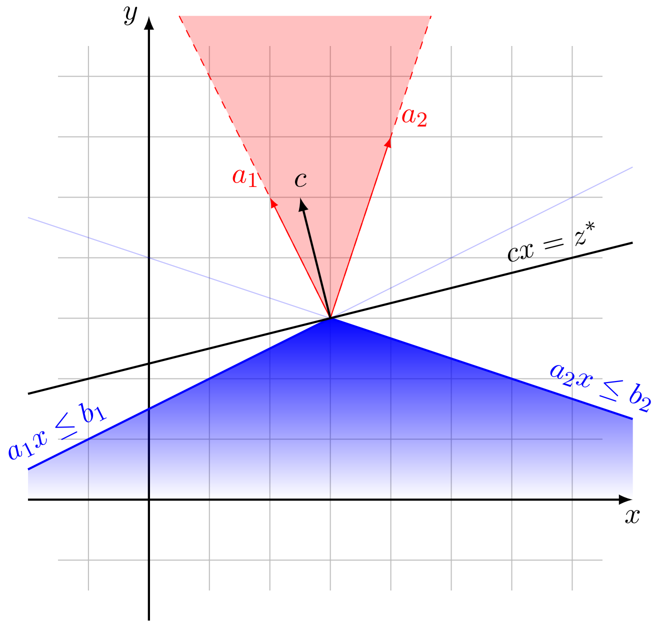


Figure 2.13: *Interpreting the duality theorem geometrically.*

- one is unfeasible, the other is unbounded.

$$\begin{array}{lrcrcr} \max & 5x & + & 7y & \text{s.t.} & \\ & x & + & y & \leq & 2 \\ & & & -y & \leq & -2 \\ & -2x & + & y & \leq & -3 \\ & & & x,y & \geq & 0 \end{array} \qquad \begin{array}{lrcrcrcr} \min & 2u & - & 2v & - & 3w & \text{s.t.} & \\ & u & & & - & 2w & \geq & 5 \\ & u & - & v & + & w & \geq & 7 \\ & & & & & u,v,w & \geq & 0 \end{array}$$

2.2.4 Geometric interpretation

The duality theorem can also be interpreted geometrically. We explain it with an example in two dimensions, which can be extended to any finite dimension.

As said before, finding an optimum max cx of a polyhedron P can be done by shifting the hyperplane $\{x \mid cx = 0\}$ as long as it intersects the polyhedron P. Then, any point in the intersection of the affine hyperplane constructed in this way, and the polyhedron, is optimal. Consider Figure 2.13.

The optimum x^* is given by the intersection of the lines $\Delta_1 : a_1x = b_1$ and $\Delta_2 : a_2x = b_2$. The affine hyperplane parallel to $\{x \mid cx = 0\}$

containing the optimal solution must have the polyhedron in only one of its side, otherwise it would contradict the optimality. But this is clearly equivalent for the normal vectors to the fact that c is in the cone generated by a_1 and a_2. In particular, the point x^* is an optimum for any objective vector $\lambda a_1 + \mu a_2$, with $\lambda, \mu \geq 0$. As $c = \lambda a_1 + \mu a_2$, the constraint $(\lambda a_1 + \mu a_2)x \leq \lambda b_1 + \mu b_2$ is valid for the linear program, and it also gives a dual solution with value $\lambda b_1 + \mu b_2$. Now, as the point x^* is on the two lines Δ_1 and Δ_2, it must satisfy with equality the equation of the new line $cx = (\lambda a_1 + \mu a_2)x = \lambda b_1 + \mu b_2$, so the primal value of x^* is also $\lambda b_1 + \mu b_2$, which is a dual value, proving the optimality of both solutions.

2.2.5 Complementary slackness

Pairs of optimal primal/dual solutions have a very special relationship, that is given by the following theorem:

Theorem 2.2.5 (Complementary slackness). *Let x^* and y^* be solutions to the primal problem* (P) *and dual problem* (D) *respectively. Then x^* and y^* are optimal if and only if:*

$$(y^*A - c)x^* = 0 \tag{2.9}$$

$$y^*(Ax^* - b) = 0 \tag{2.10}$$

Proof. We use inequations (2.3). By Theorem 2.2.4, these inequations are equations for optimal solutions, this means that x^* and y^* are optimal solution iff:

$$cx^* = y^*Ax^*$$

$$y^*Ax^* = y^*b$$

The complementary slackness follows. □

Complementary slackness is about tight constraints for the optimal solution. We need this definition:

Definition 2.2.6. *A non-negative variable is active in a solution if it has value non-zero in that solution. Otherwise, it is* inactive.

Hence, non-basic variables of a dictionary are inactive variables. Complementary slackness tells us that a dual variable can be active only if its associated constraint in the primal is tight for the primal solution, and conversely, a primal variable can be active only if the dual constraint is tight or the dual solution. Back to geometrical interpretation, we cannot get an upper bound by combining lines that do not intersect the optimal solution: we can only use lines that contains the optimal solution.

Complementary slackness comforts us in the intuition that an optimal solution is obtained by choosing some constraints and try to find an extreme solution for these constraints, *i.e.* make these constraints tight, while the other constraints are irrelevant: they are not really constraining us.

How can we use complementary slackness? Suppose we have a linear program, and a solution x^* that could be the optimal solution, and we want to check the optimality of x^*. Then, we only have to find a vector y satisfying the equations (2.9) and (2.10). The second system (2.10) immediately gives us which coefficients of y must be zero, so we only have to determine the other coefficients. Then the first system (2.9) gives us which dual constraints are tight: the dual constraint corresponding to non-zero primal variables. By solving the sub-system of $yA = c$, restricted to the lines corresponding to tight primal constraint and the columns corresponding to non-zero primal variable, we may get the dual optimal solution. If we find a feasible dual solution, we are done, x^* is optimal. Conversely, if there is no solution, or the value we get is not dual-feasible (with negative coefficients), then x^* is not optimal.

Remark. With the duality theorem and complementary slackness, we can improve our understanding of what is a face. Remember the definition of a face of $P = \{x \mid Ax \leq b\}$, F is a face if $F = \{x \mid Ax \leq b, cx = \mu\}$, for some c, and μ is the maximum of cx over P. By duality $\mu = \min\{yc|yA = c, y \geq 0\}$, call this linear program the *dual*. Then, for any optimal solution y to the dual, and any vector $x \in F$, we have by complementary slackness $y_i = 0 \Rightarrow a_i x = b_i$. Hence we define the set $I = \{i \mid y_i = 0 \text{ for some optimal solution } y \text{ of the dual}\}$, and the polyhedron $F' = \{x \mid \forall i \in I, a_i x = b_i\}$. Then, the complementary slackness says exactly that $F = F'$. It means that a face of P is the intersection of P and the solutions of a tight subsystem $A'x = b'$ of the system describing P: $Ax \leq b$, that is $F = P \cap F'$ It also implies that the number of faces of a polyhedron is finite, and even less that 2^m where m is the number of rows defining P.

Remark. Complementary slackness plays a special role when trying to solve problems in a combinatorial way. Many algorithms exploit complementary slackness to reach an optimal solution. Starting from a primal (or dual) solution, we check the complementary slackness. If the solution is optimal, we are done. Otherwise, we have some untight dual constraint corresponding to an active variable. This constraint usually gives us a natural way to improve our solution. Then, by iterating this procedure, we will finally reach an optimal solution.

2.2.6 Economic interpretation

In this section, we explain the economical significance of dual variables. For that, we consider the linear program to be a classical ressource allocation problem: given a set of materials indexed by $I = [\![1, m]\!]$, we want to produce objects indexed by $J = [\![1, n]\!]$. Each object is made with some given quantities of each material a_{ij}, each material is available in limited quantity b_i, and the profit c_j made for each object is known. As usual, our motives are only financial, so we want to solve the following problem:

$$\begin{array}{rrl} \max & cx & \text{subject to} \\ & Ax & \leq b \\ & x & \geq 0 \end{array} \tag{2.11}$$

First, let's make more precise some facts about this program. The constants all have a particular meaning: a_{ij} is the quantity of ressource i needed to make one object j, so the physical unit of a_{ij} is "unit of i per unit of j". Then b_i is the quantity of i available, its unit is thus "unit of i". We will get a homogeneous inequation if each x_j is given in "unit of j", which is indeed the case. Then, the unit of c_j is "dollars per unit of j". Now, we give the dual program:

$$\begin{array}{rrl} \min & yb & \text{subject to} \\ & yA & \geq c \\ & y & \geq 0 \end{array} \tag{2.12}$$

With the insight of our first remark, we get that the unit of y_i must be "dollars by unit of i", so the dual variables are values given to ressources. But we can actually fully interpret the dual, in the following way. y_i corresponds to a value that we give to the ressource i. The constraints say that these values must be chosen such that the cost of each object is more than its price. We can understand it as the price at which we would made no profit by buying it. Then the objective is to minimize the value of the stock of ressources.

The duality theorem tells us that by choosing good prices for each ressource, we get that we will earn the exact value of our stock given by the optimum dual solution. In some sense, the dual values give the economical value of a ressource: if we can buy one at less than its dual price, we will make profit, however if the price of it is more than the dual price, we have no interest in buying it. This also implies that each object produced must be sold at the price of the ressources consumed to make it, and that each ressource that is not fully consumed is worth nothing; this is complementary slackness. If a ressource is not fully consumed, it means that we have more than enough of it, so we do not want to buy it at any price, its dual price is zero. If the dual price of the ressources needed for a product is more than the final price of it, then we should not sell this product, and so not produce it.

Another way to explain it is the following. Suppose that somebody wants to buy our stock of ressources. He wants to find the minimum price at which we will accept the deal. He gives prices to each kind of ressource. He knows that if the prices are such that we earn more by making one of our product, than by directly selling to him the exact quantity of ressources needed to make this product, we will certainly refuse. This is why he has the inequalities of the form $ya_i \geq c_i$, it must be better for us to sell him than to make any of our product. Clearly, each price must be positive. Because he wants the best price, he minimizes yb, the total cost of our stock. This gives exactly the dual program. The duality theorem shows that he cannot propose us a price less than what we can earn anyway, which is completely coherent.

Definition 2.2.7. *The value of a dual variable corresponding to a constraint i is called the* shadow price*, or the* marginal cost *of i (i usually corresponds to a ressource).*

Remark. The contribution of b_i in the objective of the last dictionary is exactly the coefficient of s_i times b_i, where s_i is the slack variable of the line i. This can be seen as usual from the fact that s_i is basic in the original dictionary. But the coefficient of s_i is, as we have seen, the value of y_i in the optimal dual solution, so the ressource i contributes to exactly $y_i b_i$, and what we earn from each unit of ressource i is indeed y_i, confirming our discussion above.

Another consequence is that if someone want to sell us the ressource i at a price below y_i, we should accept it, and we will earn y_i times the number of units bought, and thus make a positive profit. Unfortunately, we should not buy too much of this ressource, as at some point we will be limited by another ressource, and the value of i will decrease. And we do not know yet how much we should buy. The marginal cost must be understood as a derivative of the objective with respect to b_i: if we increase our stock of ressource i by ε, we will earn εy_i, for $\varepsilon \to 0$ (this is the meaning of the *marginal*).

Chapter 3

Advanced topics on the simplex method

3.1 The revised simplex method

Commercial implementations of the simplex method do not deal with dictionaries, as their implementation is not very efficient in practice. The revised simplex method is an implementation of the simplex method that do not keep in memory the whole dictonary, but only the basic solution and the basis.

3.1.1 Factorizing matrices

The efficiency of the revised simplex method relies on our ability to quickly find solutions y to a system of equality $Ax = b$. We recall the Gaussian method and the decomposition that it gives.

Given a system $Ax = b$, where A is a non-singular square matrix of dimension n and b a column vector with n rows, we know that there is a unique x satisfying the equation. Moreover, x can be found by the so-called Gaussian elimination, that we describe here.

Definition 3.1.1. *A matrix A is* upper triangular *if all its coefficient below the diagonal are zero, that is $a_{ij} = 0$ for all $i > j$.*

If A is upper triangular, it is easy to solve $Ax = b$: indeed, $x_n = b_n$ is immediate. Then, substituting x_n by b_n leads to a new equivalent system with an upper triangular matrix with dimension $n - 1$. We thus iteratively find x_{n-1}, x_{n-2},. . . ,x_1. The Gaussian elimination consists in transforming any system $Ax = b$ in an equivalent system $A'x = b'$ with A' upper triangular.

During the process, we will compute matrices A^k with the property:

$$\text{for all } i \leq k, \text{ and } j > i, a_{ij}^k = 0$$

The matrix A^k, informally, looks triangular in its k first columns. It is then clear that $A = A^0$ checks this property, and that A^n is upper triangular. We only have to show how to compute A^{k+1} given A^k. Because we want to keep a system equivalent to $Ax = b$, we will also compute vectors B^k for each k, with $b^0 = b$. Thus we want that for each k, $Ax = b$ is equivalent to $A^k x = b^k$. In order to keep this equivalence, we will only do operations on the rows of A^k (multiplying a row by a non-zero scalar, adding one row to another,permuting rows), and we will do the same operation on b^k, to get A^{k+1} and b^{k+1}.

Suppose then that we have computed A^k and b^k. To compute A^{k+1}, we must get zero coefficients at position $i, k+1$, for i greater than $k+1$. If $a_{i(k+1)}^k = 0$ for all $i \geq k+1$, then we are done, $A^{k+1} = A^k$ and $b^{k+1} = b^k$ (actually, if A is non-singular, this case cannot happen, but the Gaussian elimination can be applied to singular matrices as well). Else there is a row i with $a_{i(k+1)}^k \neq 0$. By performing a permutation of the rows $k+1$ and i, we can assume that $i = k+1$. Then we substract to each row $i > k+1$ as many times the row $k+1$ as we need to cancel the coefficient $a_{i(k+1)}^k$, that is $\frac{a_{i(k+1)}^k}{a_{(k+1)(k+1)}^k}$. This gives the matrix A^{k+1}, and by the same row operations we also get b^{k+1}.

Example: We apply Gaussian elimination to the following system:

$$\begin{pmatrix} 2 & 3 & 1 & 2 \\ 2 & 3 & 0 & 1 \\ 1 & 2 & 1 & 2 \\ 4 & 7 & 0 & 0 \end{pmatrix} x = \begin{pmatrix} 1 \\ 3 \\ -2 \\ 4 \end{pmatrix} \tag{3.1}$$

We substract the first row to the second, half the first to the third, and twice the first to the last row:

$$\begin{pmatrix} 2 & 3 & 1 & 2 \\ 0 & 0 & -1 & -1 \\ 0 & 0.5 & 0.5 & 1 \\ 0 & 1 & -2 & -4 \end{pmatrix} x = \begin{pmatrix} 1 \\ 2 \\ -2.5 \\ 2 \end{pmatrix} \tag{3.2}$$

Then, we permute the second and third rows, and then substract twice the second row to fourth:

$$\begin{pmatrix} 2 & 3 & 1 & 2 \\ 0 & 0.5 & 0.5 & 1 \\ 0 & 0 & -1 & -1 \\ 0 & 0 & -3 & -6 \end{pmatrix} x = \begin{pmatrix} 1 \\ -2.5 \\ 2 \\ 7 \end{pmatrix} \tag{3.3}$$

Finally, we substract three times the third row from the fourth row:

$$\begin{pmatrix} 2 & 3 & 1 & 2 \\ 0 & 0.5 & 0.5 & 1 \\ 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & -3 \end{pmatrix} x = \begin{pmatrix} 1 \\ -2.5 \\ 2 \\ 1 \end{pmatrix} \tag{3.4}$$

The different matrices A that we will use are related to each other in a special way. To improve our method, we want to avoid to recompute the Gaussian elimination completely each time that we slightly change our matrix A. The solution is to keep a factorization of A in terms of the upper triangular matrices plus the matrices corresponding to the row operations that we made. More exactly, at each step of the Gaussian elimination, we multiply both A^k and b^k by a permutation matrix[1] P_k, corresponding to the permutation of the rows, and one simple matrix L^k corresponding to the addition of the row $k+1$ to each row $i > k+1$. We have that $A^{k+1} = L^{k+1}P^{k+1}A^k$ and $b^{k+1} = L^k P^k b^k$, and thus:

$$U = A^{n-1} = L^{n-1}P^{n-1}L^{n-2}P^{n-2} \dots L^1 P^1 A$$
$$b^{n-1} = L^{n-1}P^{n-1}L^{n-2}P^{n-2} \dots L^1 P^1 b$$

The matrix L^k is given by:

$$\begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & 1 & & & \\ & & & \frac{-a^k_{(k+1)k}}{a^k_{kk}} & 1 & & \\ & & & \vdots & & \ddots & \\ & & & \frac{-a^k_{nk}}{a^k_{kk}} & & & 1 \end{pmatrix} \tag{3.5}$$

This matrix is lower triangular[2]. The only non-zero coefficients are in the

[1] A permutation matrix is a matrix with exactly one '1' in each row and each column, the other coefficients being 0.

[2] The transpose of an upper triangular matrix

diagonal and the kth column. The permutation matrix P for exchanging tows i and j is given by $p_{kl} = 1$ if $i \neq k = l \neq j$ or $\{k, l\} = \{i, j\}$, and $p_{kl} = 0$ else.

Suppose that we know the decomposition $U = L^n P^n \dots L^1 P^1 A$, how can we easily solve the system of equations $Ax = b$? As all the matrices L^k and P^k are non-singular, it is equivalent to solve $Ux = L^n P^n \dots L^1 P^1 b$. But U being upper triangular, it is easy to compute the solution, once we have computed the right-hand side. Remark that all the matrices of the decomposition have a very special structure. Multiplying by a permutation matrices is just exchanging the columns of the vector, while multiplying one of the L^k requires only $n - k$ multiplication and $n - k$ additions. Hence, the computation of the right-hand side requires only $O(n^2)$ arithmetic operations. Then, solving the upper triangular system also requires $O(n^2)$ operations. Solving the same system by Gaussian elimination would have required $O(n^3)$ arithmetic operations, hence keeping the factorisation of A and using it many times is obviously a great improvement upon just solving everything from the beginning.

3.1.2 Rewriting the simplex method

The idea behind the revised simplex method is to get rid of all the bureaucracy consisting in updating a dictionary at each iteration: a dictionary has $n \times m$ elements, but we already know that the basic solution is determined by the tight constraints of this solution, that is only m elements. Actually, we will give an algorithm that keep in memory the basis (of size m), the basic solution (of size m also), and a factorized matrix of dimension $m \times m$: the submatrix corresponding to the basic variable. For now, just remember that we want to work only with the basis $I \in [\![1, n + m]\!]$, and a feasible solution x with $x_i = 0$ if $i \notin I$.

We still want to solve the standard form program:

$$\max cx \text{ s.t. } Ax \leq b, x \geq 0$$

The first step of the simplex method is to find an entering variable. For this we want to find a non-basic variable $x_j, j \notin I$ with positive coefficient in the objective line. Unfortunately, we do not have the objective line anymore. But we can get it by replacing the basic variables in the original objective by non-basic variables. For this, we need to find substitution formulas for the basic variable, that is, we need the upper part of the dictionary. This upper part is equivalent to:

$$\begin{pmatrix} A & I \end{pmatrix} x = b$$

where in this expression x contains the slack variables also. We distinguish between the basic variables x_B and the non-basic variables x_N, and rewrite it:

$$Bx_B + A_N x_N = b$$

Here, B consists in the column of $(A\ I)$ corresponding to the basic variables, and A_N the columns corresponding to non-basic variables. To get the substitution formulas, we isolate the basic variables, and inverse B, to get the upper part of the dictionary:

$$x_B = B^{-1}b - B^{-1}A_N x_N$$

Then, by substitution, the objective line is:

$$z = c_N x_N + c_B x_B = c_B B^{-1}b + (c_N - c_B B^{-1}A_N)x_N$$

To choose the entering variable, we need to evaluate $c_N - c_B B^{-1}A_N$, but for that we only need to know what are B and A_N, that is to know the basis. So we are done.

Then, we must find a leaving variable. Let $x_j, j \notin I$ be the non-basic entering variable. All that we need is $B^{-1}b$, but that is the basic solution and we assumed that we know it, and the column corresponding to x_j in $B^{-1}A_N x_N$. If a is the column of x_j in A_N, then we want to compute $d = B^{-1}a$. We can do this as we know B and a, so we are done, we can compute the ratios and choose the leaving variable x_i.

To conclude the revised simplex method, we need to compute the new basis, that is $I' = (I \setminus \{i\}) \cup \{j\}$, and the new basic solution $B^{-1}b - td + te_j$, which is easy.

Instead of updating the whole dictionary, we have to compute at each iteration $c_n - c_B B^{-1}A_n$ and $B^{-1}a$. At first glance, it seems that the revised version requires more computation than the classical version. To get an improvement, we must show how to compute these two values efficiently. The main difficulty is to inverse the matrix B, as this is the most expensive operation. Fortunately, we do not have to inverse B. To get $d = B^{-1}a$ and $y = c_B B^{-1}$, we only need to perform a Gaussian elimination, more exactly to solve $Bd = a$ and $yB = c_B$. But again, Gaussian elimination is expensive, so the next idea is to keep in memory a factorization of B. Our hope is that with this factorization, solving systems like $Bd = a$ and $yB = c_B$ will be cheap, and updating the factorization at the end of each iteration is easy.

In the original dictionary, $B = B_0 = I$, as the basic variables are the slack variables. More generally, we could start from any basis, that is any subset of m variables whose corresponding columns induce a non-singular matrix B_0, and perform a Gaussian elimination on B_0 to get a factorization $U = L^m P^m \dots L^1 P^1 B_0$. At the end of an iteration, one variable leaves the basis, so we must remove the corresponding column, and another one enters the basis, so we must add a new column. This new column is given the column a corresponding to the entering variable in the original matrix. But as we want to keep a factorization of B and not only B, we must find matrix operations that replace the old column by the new one. For this, we need to express a as a combination of the columns of B, but that is exactly what we did when we solved $Bd = a$. Then, the new matrix B' is obtained from B by keeping the same column, except the one corresponding to x_i which is replaced by Bd. In matricial notation, if the ith column of A appears in the kth column of B it gives:

$$B' = B \begin{pmatrix} 1 & & & d_1 & & & \\ & \ddots & & \vdots & & & \\ & & 1 & d_{k-1} & & & \\ & & & d_k & & & \\ & & & d_{k+1} & 1 & & \\ & & & \vdots & & \ddots & \\ & & & d_m & & & 1 \end{pmatrix}$$

This matrix is very simple as all coefficients except in the diagonal and one column are zero. So multiplying a vector by this matrix can be done in $O(m)$ operations only. Matrices of this form are called *eta-matrices.*

In the general case, we will have that the kth base matrix B_k is described by:

$$B_k = B_{k-1} E_k = B_0 E_1 \dots E_k$$

Solving $B_k a = d$ is then the same as solving:

$$U E_1 \dots E_k a = L^m P^m \dots L^1 P^1 d$$

and both sides only involve simple calculations as all the matrices have a special structure that makes matrix multiplication easy and efficient.

Similarly, solving $yB_k = c_B$ gives:

$$y B_0 E_1 \dots E_k = c$$

We introduce y' defined by $y = y' L^m P^m \dots L^1 P^1$, then we only have to solve:

$$y = y' L^m P^m \dots L^1 P^1$$

$$y'UE_1 \dots E_k = c_B$$

We first find y' and then deduce the value of y. Again, only easy computations are needed here.

We finally give the complete description of an iteration of the revised simplex method, and an example of computation:

Let $U = L^m P^m \dots L^1 P^1 B_0$, $B_k = B_0 E_1 \dots E_k$, and x^k a basic solution with basis the column of A appearing in B_k.

1. Compute the solution y of the system $y = y' L^m P^m \dots L^1 P^1$, $y'UE_1 \dots E_k = c_B$. If $c_N - yA_N$ is non-positive, x^k is optimal and the method ends. Otherwise, choose a non-basic variable x_j whose coefficient is positive.
2. Compute the solution d of the system $UE_1 \dots E_k d = L^m P^m \dots L^1 P^1 a$. If d is non-positive, the linear program is unbounded with basic solution x^k and feasible direction $d + e_j$, and the method ends. Else, find a basic variable x_i with $d_i > 0$ minimizing the ratio $\frac{x_i^k}{d_i}$.
3. Define $B_{k+1} = B_0 E_1 \dots E_k E_{k+1}$ where E_{k+1} is obtained by replacing the column l of the identity matrix by a, with l being the index of the column of B_k containing the ith of $(A\ I)$. Define $x_{k+1} = x_k - td + te_j$.

Example: We solve the following program with the revised simplex method:

$$\begin{array}{lccccccccl}
\max & x_1 & + & 3x_2 & - & x_3 & + & x_4 & \text{subject to} & \\
 & 2x_1 & - & x_2 & + & x_3 & - & 2x_4 & \leq & 6 \\
 & x_1 & + & x_2 & - & 3x_3 & + & x_4 & \leq & 4 \\
 & -x_1 & + & 2x_2 & + & x_3 & + & 2x_4 & \leq & 12 \\
 & & & & & & & x_1, x_2, x_3, x_4 & \geq & 0
\end{array} \tag{3.6}$$

We introduce slack variables x_5, x_6 and x_7, for the first, second and third constraints respectively. Then we get the following matrix A:

$$A = \begin{pmatrix} 2 & -1 & 1 & -2 & 1 & 0 & 0 \\ 1 & 1 & -3 & 1 & 0 & 1 & 0 \\ -1 & 2 & 1 & 2 & 0 & 0 & 1 \end{pmatrix}$$

The first basis is given by the slack variables, $I_0 = [5,6,7]$. The matrix B_0 is defined by:

$$B_0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The first basic solution is then $x^0 = (0,0,0,0,6,4,12)$. This is enough to start the revised simplex method.

First iteration. We solve the system $yB_0 = c_B$. B_0 is the identity, hence $y = (0,0,0)$. Then $c_N - yA_N = c_N = (1,3,-1,1)$. Among x_1, x_2 or x_4, we choose x_1 as entering variable.

Then we solve $B_0 d = (2,1,-1)^T$. Again, $B_0 = I$, then $d^T = (2,1,-1)$. The leaving variable must be x_5 with a ratio of $\frac{6}{2} = 3$.

We compute $x^1 = x^0 - 3d + 3e_1 = (3,0,0,0,0,1,15)$. The new basis is defined by $I_1 = [1,6,7]$, and the basis matrix is:

$$B_1 = B_0 \begin{pmatrix} 2 & 0 & 0 \\ 1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}$$

Second iteration. We solve $yB_1 = c_B$, that is:

$$y \begin{pmatrix} 2 & 0 & 0 \\ 1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} = (1,0,0)$$

The two last components of y are zero, and then the first is $\frac{1}{2}$, thus $y = \left(\frac{1}{2},0,0\right)$. Then

$$c_N - yA_N = (3,-1,1,0) - \left(\frac{1}{2},0,0\right) \begin{pmatrix} -1 & 1 & -2 & 1 \\ 1 & -3 & 1 & 0 \\ 2 & 1 & 2 & 0 \end{pmatrix} = \left(\frac{7}{2}, -\frac{3}{2}, 2, -\frac{1}{2}\right)$$

Hence, we can choose between x_2 and x_4, x_2 is our entering variable.

We solve $B_1 d = (-1,1,2)^T$:

$$\begin{pmatrix} 2 & 0 & 0 \\ 1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} d = (-1,1,2)^T$$

The first component is clearly $-\frac{1}{2}$, then we get $d^T = \left(-\frac{1}{2}, \frac{3}{2}, \frac{3}{2}\right)$. The ratio for x_6 is better that for x_7, so x_6 leaves with ratio $\frac{2}{3}$.

We update our partial solution. $x^2 = \left(\frac{10}{3}, \frac{2}{3}, 0, 0, 0, 0, 14\right)$, $I_2 = [1, 2, 7]$ and:

$$B_2 = B_0 \begin{pmatrix} 2 & 0 & 0 \\ 1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\frac{1}{2} & 0 \\ 0 & \frac{3}{2} & 0 \\ 0 & \frac{3}{2} & 1 \end{pmatrix}$$

Third iteration. We solve $yB_2 = c_B$:

$$y \begin{pmatrix} 2 & 0 & 0 \\ 1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\frac{1}{2} & 0 \\ 0 & \frac{3}{2} & 0 \\ 0 & \frac{3}{2} & 1 \end{pmatrix} = (1, 3, 0)$$

First we compute y' with:

$$y' \begin{pmatrix} 1 & -\frac{1}{2} & 0 \\ 0 & \frac{3}{2} & 0 \\ 0 & \frac{3}{2} & 1 \end{pmatrix} = (1, 3, 0)$$

We easily find that $y' = \left(1, \frac{7}{3}, 0\right)$. Then we have:

$$y \begin{pmatrix} 2 & 0 & 0 \\ 1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} = y' = \left(1, \frac{7}{3}, 0\right)$$

Hence we deduce $y = \left(-\frac{2}{3}, \frac{7}{3}, 0\right)$. Then:

$$c_N - yA_N = (-1, 1, 0, 0) - \left(-\frac{23}{3}, \frac{11}{3}, -\frac{2}{3}, \frac{7}{3}\right) = \left(\frac{20}{3}, -\frac{8}{3}, \frac{2}{3}, -\frac{7}{3}\right)$$

x_3 or x_5 can enter, we choose x_3.

Then we solve $B_2 d = (1, -3, 1)$. First, we solve:

$$\begin{pmatrix} 2 & 0 & 0 \\ 1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} d' = a$$

It gives $d' = \left(\frac{1}{2}, -\frac{7}{2}, \frac{3}{2}\right)^T$. Then we have:

$$\begin{pmatrix} 1 & -\frac{1}{2} & 0 \\ 0 & \frac{3}{2} & 0 \\ 0 & \frac{3}{2} & 1 \end{pmatrix} d = d'$$

We get $d = \left(-\frac{2}{3}, -\frac{7}{3}, 5\right)^T$. x_7 is the leaving variable, the ratio is $\frac{14}{5}$.

We update the partial solution: $x^3 = \left(\frac{26}{5}, \frac{36}{5}, \frac{14}{5}, 0, 0, 0, 0\right)$, $I_3 = [1, 2, 3]$, and:

$$B_3 = \begin{pmatrix} 2 & 0 & 0 \\ 1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\frac{1}{2} & 0 \\ 0 & \frac{3}{2} & 0 \\ 0 & \frac{3}{2} & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -\frac{2}{3} \\ 0 & 1 & -\frac{7}{3} \\ 0 & 0 & 5 \end{pmatrix}$$

Fourth iteration. We must first solve $yB_3 = c_B$:

$$y \begin{pmatrix} 2 & 0 & 0 \\ 1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\frac{1}{2} & 0 \\ 0 & \frac{3}{2} & 0 \\ 0 & \frac{3}{2} & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -\frac{2}{3} \\ 0 & 1 & -\frac{7}{3} \\ 0 & 0 & 5 \end{pmatrix} = (1, 3, -1)$$

We use intermediate solution $y_1 = \left(1, 3, \frac{4}{3}\right)$, $y_2 = \left(1, 1, \frac{4}{3}\right)$ and finally $y = \left(\frac{2}{3}, 1, \frac{4}{3}\right)$. Then:

$$c_N - yA_N = (1, 0, 0, 0) - \left(\frac{7}{3}, \frac{2}{3}, 1, \frac{4}{3}\right) = \left(-\frac{4}{3}, -\frac{2}{3}, -1, -\frac{4}{3}\right)$$

There is no candidate for entering the basis, hence the solution x^3 is optimal. In terms of the original problem the solution is $\left(\frac{26}{5}, \frac{36}{5}, \frac{14}{5}, 0\right)$. The optimal value is 24. The dual solution is given by y: $\left(\frac{2}{3}, 1, \frac{4}{3}\right)$.

3.2 Implementation details

3.2.1 Complexity

The complexity of the revised simplex method is dominated by the computation of the solution of $yB = c_B$ and $Bd = a$. As B is given in a factorized form, and each factor is a matrix with only $O(m)$ coefficients, plus the upper triangular matrix, with a good implementation of matrices we have a complexity in $O(m^2 + km)$ where k is the number of factors.

The matrices used in practice are usually very sparse: they have only a few non-zero coefficient in each column. In that case, we could obtain a triangular matrix that is also very sparse, by carefully choosing the pivot during the Gaussian elimination. Pivoting an arbitrary row could fill the matrix into a dense system, for example, we do not want to pivot the first line of the following matrix:

$$\begin{pmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
1 & 3 & & & & & \\
1 & & 4 & & & & \\
1 & & & 5 & & & \\
1 & & & & 6 & & \\
1 & & & & & 7 & \\
1 & & & & & & 8
\end{pmatrix}$$

as we would then get:

$$\begin{pmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
 & 1 & -3 & -4 & -5 & -6 & -7 \\
 & -2 & 1 & -4 & -5 & -6 & -7 \\
 & -2 & -3 & 1 & -5 & -6 & -7 \\
 & -2 & -3 & -4 & 1 & -6 & -7 \\
 & -2 & -3 & -4 & -5 & 1 & -7 \\
 & -2 & -3 & -4 & -5 & -6 & 1
\end{pmatrix}$$

If we want to create as few new non-zero coefficient as possible, we must choose a pivoting line that has many zeros. One possible rule was given by Markowitz. Choose a coefficient minimizing the product $(p-1)(q-1)$, where p is the number of non-zero coefficient in its row, and q the number of non-zero coefficient in its column. With this rule, the matrices that we get will tend to be sparse. Hence we will get a much more efficient factorization. For example, the first step of the Gaussian elimination on the matrix above would give:

$$\begin{pmatrix}
\frac{1}{3} & 0 & 3 & 4 & 5 & 6 & 7 \\
1 & 3 & & & & & \\
1 & & 4 & & & & \\
1 & & & 5 & & & \\
1 & & & & 6 & & \\
1 & & & & & 7 & \\
1 & & & & & & 8
\end{pmatrix}$$

Another way of improvement is the following. After many iterations, the factorization of B can become very big, and the computation of all these factors, as it is done at each iteration, begins to cost a lot. In that case, it is useful to compute a fresh factorization of the basis B by performing a Gaussian elimination on the original B. In practice, doing such a refactorization every twenty operations leads to good results.

3.2.2 Accuracy

Efficient implementations of the simplex method uses floating point arithmetic, instead of rational or real arithmetic. It means that the values encoded by the algorithm are rounded to the closest floating point, hence with a very slight approximation. Unfortunately, when performing arithmetic operation, the approximation can become bigger and bigger, to the point that the value obtained are completely false after a few operations. To understand how the floating point arithmetic can create wrong result because of the approximation, we solve a linear system of equations with five significant digits:

$$\begin{pmatrix} 10^{-6} & 2 \\ 2 & 1 \end{pmatrix} x = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

After applying Gauss elimination we get:

$$\begin{pmatrix} 10^{-6} & 2 \\ 0 & -4 \cdot 10^{6} \end{pmatrix} x = \begin{pmatrix} 1 \\ -2 \cdot 10^{6} \end{pmatrix}$$

Hence the solution is $x = (0, 0.5)$. But when we check this solution with the original system of equations, we get that the second equation is completely false. The exact solution rounded to five significant digits is $x = (0.75, 0.5)$, which is obviously far better when we compare to the original problem.

So we do not even need to do many computations to get errors in the approximation, and the previous example would have worked for larger precision by replacing the coefficient 10^{-5} by an even smaller value. The reason of the failure of the Gaussian elimination with rounding is due to the fact that by choosing a very small pivot, after rounding the second equation what we get as nothing to do with the original second equation: the terms of the original matrix are negligible compared to the inverse of the pivot. A better way to perform the Gaussian elimination would then be to choose a large pivot. Indeed, on our example, by chhosing the second coefficient of the first column as a pivot, we would get:

$$\begin{pmatrix} 0 & 2 \\ 2 & 1 \end{pmatrix} x = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

and then $x = (0.75, 0.5)$, which is correct.

The main reason for inaccuracy in the Gaussian elimination is the difference of scale between coefficients of the original matrix. One common technic to counter this is to change the coefficients, by playing with two possibilities:

- we can multiply one row of A and the corresponding row in b by a non-zero scalar without modifying the solution of the system,
- we can change a variable x_i by $\lambda^{-1}x_i$, which in practice means that we multiply the ith column by $\lambda \neq 0$.

With these two tools, we can reduce the differences of magnitude between the coefficient of the system. In our example, we would get:

$$\begin{pmatrix} 1 & 2000 \\ 2000 & 1 \end{pmatrix} x = \begin{pmatrix} 1000 \\ 2 \end{pmatrix}$$

Then, when applying Gaussian elimination, we can choose the pivot to maximize the absolute value among the candidates in the column. We can also add column permutation in the Gaussian elimination, that would allow us to take for pivot the biggest coefficient in the bottom-right submatrix. The former strategy is called *partial pivoting*, the latter *complete pivoting*.

A common approach when analysing the error made by an algorithm is not to see how much the original equations are violated (*forward error analysis*), but to consider how much we need to change the coefficients of the original problem to make the solution be correct (*backward error analysis*). In our example, we only have to change the coefficient 10^{-6} to 0 to get that $(0.75, 0.5)$ is a solution. It appears that Gaussian elimination with partial pivoting is very accurate for this measure.

One good reason for using backward error analysis is that anyway the values of the original matrix are not accurate themselves, in real-life applications: they are found empirically or are estimations only, hence slightly modifications of them have usually no significance. Therefore, a solution to a slightly modified version of the problem is as good as an exact solution.

3.3 Boxed simplex

In this section, we extend the two-phase simplex method to linear programs with explicit bound on variables. Let be a linear program with variables x in the following form:

$$\begin{array}{ll} \max & cx \qquad \text{subject to} \\ & Ax = b \\ & l \leq x \leq u \end{array} \tag{3.7}$$

where c, b, u and v are given vectors, A is a given matrix. The last line gives for every variable x_i a lower bound and an upper bound, $l_i \leq x_i \leq u_i$. We allow some of the l_i and some of the u_i to be undefined, that is x_i may be unbounded in one direction, but not both, that is x_i is not free. If x_i has no lower bound, we will note $-\infty \leq x_i$, if x_i has no upper bound we note $x_i \leq \infty$. Note that a standard form program to which we add slack variables can be written as a boxed simplex, by taking $l = 0$ and $u = \infty$, so boxed linear programs are not a restriction of general linear programs.

We postpone the treatment of phase one, *i.e.* how to find a basic feasible solution. We assume that we have a feasible basic solution. What does it look like? As usual, to be basic, it must happen that most of the inequalities are tight. More exactly, if we have m constraints (without counting the bounds) and n variables, then we have $2n+m$ constraints (with the bounds), and n of them must be tight in a basic solution. As the inequalities for the lower bound and for the upper bound of one variable cannot be checked at once, it means that $n - m$ variables have the value of either their upper bounds or their lower bounds, and the m other variables have their values determined by $Ax = b$. The former variables are called *non-basic*, and the latter *basic*, as usual. But this time, the non-basic variables do not have value zero, but they are equal to one of their bounds. Hence we distinguish between the set U of non-basic variables being equal to their respective upper bounds, and the set L of those being equal to their lower bounds. Hence a basic solution is determined by three sets: the set B of basic variables, and the sets L and U of variables reaching their lower and upper bounds respectively.

Note that in the classical case, when $l = 0$ and $u = \infty$, U is always empty (no variable can have value ∞), and the solution is thus determined by L and B. So we are just extending the results that we already know. Then, we would like to extend the simplex method as well: we can again write dictionaries in the exact same way, but this time we must remember that it is not the non-negativity that is implicit in the dictionary, but the bound on the variables.

What does it change? First, when choosing the entering variable, there are two cases: we can choose a variable in L that we could increase, we need that its coefficient in the objective line is positive (to increase the objective value). But we can also choose a variable in U, which means that we will decrease its value, so its coefficient must be negative.

Second, when we choose the leaving variable, we do not want the basic variable to stay non-negative, but to stay within their bounds. Hence, we do not want them to increase beyond their upper bound, or the decrease below their lower bound, so we must check both the positive coefficients of the column of the entering variable, and the negative coefficients. Moreover, we do not want either the entering variable to break its other bound: if we increase it, we must be careful not to increase it above its upper bound. This last remark also means that an entering variable may not enter: if we increase it to its upper bound (or decrease it to its lower bound), it stays non-basic, so the next basis would be the same, only U and L would change by one element.

We write it in terms of the revised simplex method. Given L, U, the submatrix B of A corresponding to the columns of the basis, and the corresponding solution x^0, we would have the following dictionary as usual (recall that the bounds are implicit):

$$\begin{array}{rcrcr} x_B & = & B^{-1}b & - & B^{-1}A_N x_N \\ \hline z & = & c_B B^{-1}b & - & (c_N - c_B B^{-1}A_N)x_N \end{array} \tag{3.8}$$

Given the vector $c_N - c_B B^{-1}A_N$, we can choose as entering variable any variable in L with a positive coefficient, or any variable in U with a negative coefficient. If there is no possibility, the basic solution is optimal. Else, once we have chosen an entering variable, we must choose the leaving variable x_j. For the variable x_i, if the coefficient in the line defining x_i of the entering variable is d_i, then we look how the variable i will be affected by the change of the entering variable: if $d_i > 0$ and $j \in L$, or if $d_i < 0$ and $j \in U$, the variable decreases, and it can decrease by at most $x_i^0 - l_i$, hence x_j cannot increase by more than $\left|\frac{x_i^0 - l_i}{d_i}\right|$. Else, x_i increases, limiting the change of x_j by $\left|\frac{u_i - x_i^0}{d_i}\right|$. Moreover, x_j cannot change by most than $u_j - l_j$. If one of the former constraints on x_j is the more constraining, then x_j becomes non-basic, and one of the basic variables becomes non-basic, either in L or in U. If x_j becomes basic, the matrix B must be updated. As in the classical method, degeneracy and cycling may happen, and may be avoided in the same way.

Example:

We illustrate the boxed simplex by solving the following problem:

$$\begin{array}{rrcrcrcl} \max & 3x_1 & + & 2x_2 & + & x_3 & \text{subject to} \\ & x_1 & - & 2x_2 & + & 2x_3 & \leq & 18 \\ & 4x_1 & + & x_2 & - & x_3 & \leq & 23 \\ & & & & & x_1 & \in & [2,6] \\ & & & & & x_2 & \in & [3,7] \\ & & & & & x_3 & \in & [4,15] \end{array}$$

We add non-negative slack variables x_4 and x_5, and start from the basic solution $(2, 3, 4, 14, 16)$. The basis is $[4, 5]$, while $L_0 = \{1, 2, 3\}$ and $U_0 = \emptyset$.

First iteration: As usual we solve $yB_0 = c_B$, B_0 is the identity and c_B is the null vector, hence we get that $c_N - yA_N = c_N = (3, 2, 1)$. We choose x_1 as entering variables.

Then, $B_0 d = (1, 4)^T$ leads to $d = (1, 4)^T$. The ratio for x_4 is 14, and for x_5 it is 4. Moreover, x_1 can be increased by at most 4 until it reaches its upper bound. So we can choose between keeping x_1 out of the basis, and making x_1 basic with x_5 leaving. We do the second possibility: x_5 leaves.

We get that:

$$B_1 \begin{pmatrix} 1 & 1 \\ 0 & 4 \end{pmatrix}$$

$$x^1 = (6, 3, 4, 10, 0)$$

We have $L_1 = \{2, 3, 5\}$, $U_1 = \emptyset$, and the basis is $[4, 1]$.

Second iteration: we solve $yB_1 = (0, 3)$, it gives $y = \left(0, \frac{3}{4}\right)$. Then:

$$c_n - yA_N = (2, 1, 0) - \frac{3}{4}(1, -1, 1) = \left(\frac{5}{4}, \frac{7}{4}, \frac{-3}{4}\right)$$

x_5 is at its upper bound, it cannot be decreased, but x_2 and x_3 are at their respective lower bounds, so we can increase them: we choose to increase x_2 that become our entering variable.

Then $B_1 d = (-2, 1)^T$ gives $d = \left(-\frac{9}{4}, \frac{1}{4}\right)^T$. There is no upper bound for x_4, and x_1 can be decreased by at most 4, which gives a ratio of 16. x_2 can be increased by at most 4 because of its upper bound, this gives the most constraining constraint: x_2 does not enter the basis, but reach its upper bound. Therefore, the basis is not changed, $B_2 = B_1$, and only the solution is changed to

$$x^2 = (5, 7, 4, 19, 0)$$

We also have $L_2 = \{3, 5\}$, $U_2 = \{2\}$ and the basis is $[4, 1]$.

Third iteration: We solve $yB_2 = (0, 3)$, it gives again $y = \left(0, \frac{3}{4}\right)$ and then $c_N - yA_N$ is also $\left(\frac{5}{4}, \frac{7}{4}, \frac{-3}{4}\right)$. This time x_2 is at its upper bound, hence cannot be increased, hence only x_3 can enter the basis.

Then, $B_2 d = (1, 2, -1)^T$ leads to $d = \left(\frac{9}{4}, -\frac{1}{4}\right)^T$. x_4 has value 19, it can be decreased to 0, giving a ratio of $\frac{76}{9}$. x_1 can be increased by 1, this gives a ratio of 4, and x_3 can be increased by 11, so x_1 is the leaving variable.

The next solution is then:

$$B_3 = \begin{pmatrix} 1 & 2 \\ 0 & -1 \end{pmatrix}$$

$$x^3 = (6, 7, 8, 10, 0)$$

and $L_3 = \{5\}$, $U_3 = \{1, 2\}$, the new basis is $[4, 3]$.

Fourth iteration: $yB_3 = c_B = (0, 1)$ gives $y = (0, -1)$, and $c_N - yA_N = (3, 2, 0) + (4, 1, 1) = (7, 3, 1)$. x_1 and x_2 cannot increase more, but x_5 can, so x_5 is the entering variable.

$B_3 d = (0, 1)^T$ gives $d = (2, -1)^T$. We get a ratio of 5 for x_4, and 7 for x_3, while x_5 has no upper bound, so x_4 leaves.

Hence we have:

$$B_4 = B_3 \begin{pmatrix} 2 & 0 \\ -1 & 1 \end{pmatrix}$$

$$x^4 = (6, 7, 13, 0, 5)$$

and $L_4 = \{4\}$, $U_4 = \{1, 2\}$, the basis is $[5, 3]$.

Fifth iteration: $yB_4 = (0, 1)$ gives $y = \left(\frac{1}{2}, 0\right)$. We get $c_N - yA_N = \left(\frac{5}{2}, 3, -\frac{1}{2}\right)$. But x_1 and x_2 cannot increase and x_4 cannot decrease, so the previous solution x^4 is optimal. The solution to the original problem is $(6, 7, 8)$, the objective value is 40.

The dual of (3.7) is:

$$\begin{array}{lccccccl} \min & by & + & ls & + & ut & \text{subject to} & \\ & yA & + & s & + & t & \geq & c \\ & & & & & s & \leq & 0 \\ & & & & & t & \geq & 0 \end{array} \tag{3.9}$$

From the structure of this dual, because $u > l$, it is clear that an optimal solution will have only one of s_i and t_i different from 0 for each variable x_i. This is also a consequence of complementary slackness: a variable cannot reach both its lower and upper bounds with the same value. If s_i is not zero $x_i = l_i$, while if $t_i \neq 0$ then $x_i = u_i$. The dual solution of the previous example would be $y_1 = \frac{1}{2}$, $t_1 = \frac{5}{2}$, $t_2 = 3$, and the other dual variables are zero. We get this solution by taking the value of $c_N - yA_N$ in the last iteration.

We conclude this section by explaining how we can find a basic feasible solution. We use the same trick as for the classical simplex method: we introduce artificial variables, and then try to cancel them. Given the linear program (3.7), we want to start from a solution that gives to each variable its lower bound or its upper bound. Such a solution could violate some of the equalities, so we add one artificial variable for each violated equality, and take this variable with a positive or a negative sign, depending on how the equality is violated. Thus all the artificial variables are non-negative. Then we first solve the problem consisting in minimizing the sum of the artificial variable. The minimum is zero if and only if the initial program is feasible, and if it is not feasible we will get a dual solution that gives a certificate of infeasibility. Else, we find a feasible solution. We continue with the same problem but back to the original objective, and we impose the artificial variables to stay equal to 0 (just adding an upper bound of 0). We can even completely remove the non-basic artificial variables, as we did for the classical simplex method.

3.4 Dual simplex

Let's go back to non-boxed linear programs:

$$\begin{array}{rrl} \max & cx & \text{subject to} \\ & Ax & \leq b \\ & x & \geq 0 \end{array}$$

The dual of this program is as we know:

$$\begin{array}{rll} \max & -yb & \text{subject to} \\ & -yA & \leq -c \\ & y & \geq 0 \end{array}$$

This gives the dictionary for the dual problem:

$$\begin{array}{ccccc} s & = & -c^T & + & A^T y^T \\ \hline w & = & & - & b^T y^T \end{array}$$

If we compare it to the dictionary for the primal program, we get a surprise:

$$\begin{array}{ccccc} s & = & b & - & Ax \\ \hline w & = & & & cx \end{array}$$

One is obtained from the other by taking a *mirror image* and reversing the signs. We could actually complete this mirroring effect: to any primal dictionary we can associate a dual dictionary by taking the transpose image and reversing the signs. This does not look significant by itself, but the main consequence is that we can solve the dual by the simplex method without writing the dual or any dual dictionary. Note that the dual dictionary obtained by mirroring a primal feasible dictionary is not always feasible, and conversely, but it is true for optimal dictionaries: indeed, an optimal primal dictionary has non-positive coefficient in the objective lines, which implies that the constant column of the dual dictionary is positive. This inspire the following definition:

Definition 3.4.1. *A dictionary is* dual feasible *if the coefficients of the non-basic variables in the objective line are all non-positive.*

Hence, a primal dictionary is dual-feasible if its mirror, a dual dictionary, is (primal-) feasible. Remember that the simplex method is computing a sequence of primal-feasible dictionaries, that is dictionaries with non-negative constants in each line. The dual simplex method must then compute a sequence of dual feasible dictionary: dictionaries with non-positive coefficients in the objective line.

What was driving us in the simplex method, when trying to get a new dictionary, is that we wanted to remove positive coefficients from the objective line. Similarly, in the dual simplex method, we want to remove the negative coefficients in the constant column. We can summarize this idea by this two enlightening ideas:

- In the *primal* method, we work with *primal feasible dictionaries*, to get a *primal and dual* feasible dictionary.
- In the *dual method*, we work with *dual feasible dictionaries*, to get a *primal and dual* feasible dictionary.

The reason why we are looking for a dictionary that is both primal and dual feasible is that this dictionary immediately gives us an optimal primal solution, and an optimal dual solution as we have seen. Note that by the definition of dictionaries, primal and dual values of the dictionary satisfy the complementary slackness condition, hence as soon as both of them are feasible, they are optimal.

The details of how the dual simplex method is performed are just a translation of the primal simplex method by the *mirror operation.* First, we need to find a leaving variable. As we want to get rid of negative constant, we choose any basic variable, with a negative constant in the line containing this variable. If there is no negative coefficient, the dictionary is primal feasible and the method terminates. Else we have an leaving variable and a pivoting line, we must find an entering variable. We will replace this entering variable in the objective line, by the substitution formula given by the pivoting line. As we want to keep the dictionary dual-feasible, we must be careful when choosing the entering variable. We need a variable with a positive coefficient in the pivoting line (because the leaving variable must have a negative coefficient after the substitution), and we want to minimize the ratio $\frac{c_j}{a_{ij}}$ over all the possibilities, so that other variables stay non-positive. Then, we compute the substitution formula, and substitute any occurence of the entering variable by this formula, and this ends one iteration of the dual simplex method. Note that if we cannot find an entering variable, it means that the original problem is infeasible (the dual is unbounded, hence the primal is infeasible).

As for the primal simplex method, it is more interesting, for a complexity point of view, to only compute the feasible solution encoded by the dictionary. For this, we recall that a dictionary is still determined by the set of basic variable, and can be written:

$$\begin{array}{rcrcr} x_B & = & B^{-1}b & - & B^{-1}A_N x_N \\ \hline w & = & c_B B^{-1}b & + & (c_N - c_B B^{-1}A_N)x_N \end{array} \tag{3.10}$$

We keep the *primal solution* x^k (actually it is not a feasible solution, as some variables may be non-positive) and the dual solution y^k. $B^{-1}b$ is readily available as it is x^k. Hence we can choose immediately a leaving variable x_i. Then to find the entering variable, we must compute the coefficient of the line containing x_i. If u is the line of B^{-1} corresponding to variable x_i, then the line is given by:

$$x_i = ub - uA_N x_N$$

We only have to solve $e_i B^{-1} = u$, where e_i is the row vector with coefficient 0 except the one corresponding to variable x_i which is 1. We found u by solving $Bu = e_i$, and then $v = uA_N$. Then we compute the ratios $\frac{v_j}{y_j^k}$ for every negative coefficient of v_j, and keep the non-basic variable minimizing these ratios, this is the leaving variable.

Finally, we must compute the new dual feasible solution y_j^k. If t is the value of the minimal ratio, we get $y^{k+1} = y^k + tu$ for the non-basic variable, and $y_i^{k+1} = -t$. We also need to update B, by replacing the leaving column by the entering column, that is done in the same way as in the primal simplex method, and upgrade x^k to x^{k+1}, by removing t times the entering column (again as in the primal simplex method). For both we need to solve $Bd = a$ where a is the entering column.

Example:

We solve the following program with the dual revised simplex method, $\max cx$ s.t.$Ax = b, x \geq 0$ with:

$$A = \begin{pmatrix} -8 & -6 & 1 & -5 & 1 & 0 & 0 \\ -4 & -4 & -2 & 2 & 0 & 1 & 0 \\ 3 & -2 & -3 & -2 & 0 & 0 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} 25 \\ -14 \\ -18 \end{pmatrix}$$

and $c = (-10, -30, -20, -40, 0, 0, 0)$. We take for initial solution $y^0 = (-10, -30. -20. -40, 0, 0, 0)$ (dual feasible) and $x^0 = (0, 0, 0, 0, 25, -14, -18)^T$ (not primal feasible). The base is $[5, 6, 7]$ and B_0 is the identity matrix.
First iteration.

x_6 and x_7 are non-positive in x^0, we can choose one of them as leaving variable, say x_6. Then we need the line corresponding to x_6 in the dictionary, this is given by the coefficients of the second line in $B_0^{-1}A_N$, *i.e.* $(-4, -4, -2, 2)$. y_N^0 is $(-10, -30, -20, -40)$, it gives ratios of $\left(\frac{5}{2}, \frac{15}{2}, 10, \infty\right)$, hence x_1 is the entering variable, and the minimal ratio is $\frac{5}{2}$. y_N is then modified by $\frac{5}{2}(4, 4, 2, -2)$, this gives the new solution:

$$y^1 = \left(0, -20, -15, -45, 0, -\frac{5}{2}, 0\right)$$

Then we must update the primal values and the basis, for this we compute the column d corresponding to x_1 in the dictionary: $B_0 d = (-8, -4, 3)^T = d$. This gives:

$$B_1 = B_0 \begin{pmatrix} 1 & -8 & 0 \\ 0 & -4 & 0 \\ 0 & 3 & 1 \end{pmatrix}$$

We want to cancel the value of x_6, so we need to substract $\frac{14}{4}d$ to the values of the basic variables in x^0, this gives:

$$x^1 = \left(\frac{7}{2}, 0, 0, 0, 53, 0, -\frac{57}{2}\right)^T$$

Second iteration.

From x^1, it is clear that x_7 is the only possible leaving variable. We compute its line in the dictionary by solving first $vB_1 = (0, 0, 1)$, this gives $v = \left(0, \frac{3}{4}, 1\right)$. Then the line is given by $vA_N = \left(-5, -\frac{9}{2}, -\frac{1}{2}, \frac{3}{4}\right)$. $y_N = \left(-20, -15, -45, -\frac{5}{2}\right)$, hence the ratios are $\left(4, \frac{10}{3}, 90, \infty\right)$, x_3 is the entering variable with ratio $\frac{10}{3}$. The modification of the dual solution is then

$\frac{10}{3}\left(5, \frac{9}{2}, \frac{1}{2}, \frac{3}{4}\right)$ and this gives the solution:

$$y^2 = \left(0, -\frac{10}{3}, 0, -\frac{130}{3}, 0, -5, -\frac{10}{3}\right)$$

Then we update the primal values: we compute the column d corresponding to the entering variable: $B_1 d = (1, -2, -3)^T$. This gives $d = \left(5, \frac{1}{2}, -\frac{9}{2}\right)^T$. Then we want to use d to cancel the coefficient of x_7, for that we need to substract $\frac{19}{3}$ times d to x^1, it gives:

$$x^2 = \left(\frac{1}{3}, 0, \frac{19}{3}, 0, \frac{64}{3}, 0, 0\right)^T$$

All these coefficients are positive, these primal values define a primal feasible solution, hence x^2 is an optimal solution. The optimality comes immediately from the complementary slackness.

3.5 Sensitivity Analysis

Sensitivity analysis is about how much the optimal solution changes when the parameters in the linear programs are modified. As we said when we studied accuracy, the datas used for writing the linear model are not, in practice, always precise, and may even change. For instance, consider the prices of raw materials, the conversion rates between differente moneys, etc. Hence, a solution to a problem solved one month can be non-optimal the next month, and we would have to recompute a new solution again. We can also have new variables or new constraints appearing (when one develops a new product...). But we can avoid this additional computation, or most of it, taking into account that we already have a solution to a very similar problem, and that could give us a starting point to find the new solution.

3.5.1 Changing the objective

First, suppose we have solved the linear program $\max cx$ s.t. $Ax \leq b, x \geq 0$, and that we want to change the objective functions (for example, the prices at which we sold our products have changed). Now we want to maximize $c'x$ under the same constraints, and we have a solution x^* that maximizes cx. If c and c' are similar, we can hope that the face of optimal solutions for c' is close to the face of optimal solutions for c in the polyhedron of feasible solutions. It could even happen that the optimal solution is the same. Therefore, a good idea is to initialize a simplex starting from the solution x^*, instead of starting from a completely new solution. x^* is still primal feasible, because the constraints are the same, and basic. With the revised simplex method, is is quite easy to start the simplex method from any basic solution, hence we are done.

3.5.2 Changing the right-hand side

Then, if we want to change the constraints from $Ax \leq b$ to $Ax \leq b'$, for instance because our supplying has changed. If we take the dual program, it means that we have changed the objective in the dual. Hence, we can apply the same tactic to the dual program. In practice, we would apply the dual simplex method starting from the dual solution of the original problem.

3.5.3 Changing everything

More generally, we could also completely change the program to

$$\max c'x \text{ s.t. } A'x \leq b', x \geq 0$$

Then:

- if the basis of x^* is still feasible, we can use it to initialize a simplex. This is the case when we add new variables (a new product) whose values can be zero.
- if the dual basis of x^* is still dual-feasible, we can use it to initialize a dual simplex. This is the case when we add new constraints to the linear program.
- otherwise, it is possible to use more complicated tricks to start from the solution x^* or its dual, for example we could add artificial variables with a big penalty, to get the feasibility of x^*. We could also introduce the changes in the parameters in two steps, the first step would keep the solution dual feasible. We compute a new solution, and then introduces more changes but keeping the new solution primal feasible, and then we solve again to get the final solution.

Example:

Consider the following linear program:

$$\begin{array}{rrrrrrrrrl} \max & 12x_1 & + & 15x_2 & + & 9x_3 & + & 8x_4 & \text{subject to} & \\ & 6x_1 & + & 4x_2 & + & x_3 & + & 6x_4 & \leq & 180 \\ & 3x_1 & + & 3x_2 & + & 2x_3 & + & 2x_4 & \leq & 100 \\ & 8x_1 & + & 13x_2 & + & 6x_3 & & & \leq & 260 \\ & & & & & & & x_1, x_2, x_3, x_4 & \geq & 0 \end{array} \tag{3.11}$$

The optimal solution is $x^* = \frac{1}{7}(0, 116, 52, 124)$, with an objective value of $z^* = \frac{3200}{7}$. The dual optimal solution is $y^* = \frac{1}{7}(1, 25, 2)^T$.

Suppose that we change the objective solution to $c' = (14, 14, 8, 9)$. Then the solution x^* is still basic as the constraints are not changed. The basis is $\{x_2, x_3, x_4\}$, hence we can start an iteration of the simplex method for the modified linear program with x^* and the corresponding submatrix, we want to solve:

$$y \begin{pmatrix} 4 & 1 & 6 \\ 3 & 2 & 2 \\ 13 & 6 & 0 \end{pmatrix} = (14, 8, 9)$$

We should compute a factorization for our basis matrix:

$$\begin{pmatrix} 1 & & \\ & 1 & \\ \frac{6}{5} & & 1 \end{pmatrix} \begin{pmatrix} 1 & -3 & \\ & 1 & \\ & & 1 \end{pmatrix} \begin{pmatrix} 4 & 1 & 6 \\ 3 & 2 & 2 \\ 13 & 6 & 0 \end{pmatrix} = \begin{pmatrix} -5 & -5 & 0 \\ 3 & 2 & 2 \\ 13 & 6 & \end{pmatrix} =$$

We state this matricial equality by $L_2 L_1 B = B'$. Then we solve $y'B' = (14, 8, 9)$ and $y' L_2 L_1 = y$, to get $y' = \frac{1}{70}(14, 315, 15)$ and $y = (32, 219, 15)$.

Then we compute $c_N - yA_N = \frac{1}{70}(11, -32, -219, -15)$. x_1 is a candidate for entering the basis. We solve $Bd = (6, 3, 8)^T$, which gives $d = \frac{1}{70}(44, -2, 41)^T$. After a few computations, we get the new feasible solution $\frac{1}{11}(290, 0, 90, 25)$. Then, we compute the new dual $y = \left(\frac{1}{2}, 3, \frac{1}{4}\right)$, and the objective line $-\left(\frac{1}{4}, \frac{1}{2}, 3, \frac{1}{4}\right)$, the solution is optimal.

Let's go back to the first linear program, and let's change the right-hand side to $(160, 120, 240)^T$. x^* is not feasible, but y^* is still dual feasible. We can start a dual simplex method with y^* as an initial value. First, we must compute the primal solution associated to this dual value in the new program, given by:

$$\begin{pmatrix} 4 & 1 & 6 \\ 3 & 2 & 2 \\ 13 & 6 & 0 \end{pmatrix} x = (160, 120, 320)^T$$

The solution is $x = \frac{1}{7}(80, 200, 100)^T$, this is a primal feasible solution, hence the solution to the new linear program is $\frac{1}{7}(0, 80, 200, 100)^T$, with an objective value of $\frac{3800}{7}$. If one of the values of x was negative, we would have proceeded to an iteration of the dual simplex method until we found a solution.

3.5.4 Changing parameters without changing the solution

We can also determine how much we can change the values without effect on the solution. For example, if x_i is non-basic in the solution, we can decrease its value c_i in the objective: it would then be even less interesting to use it, so this would not affect the solution. But if we increase its value, there is a point beyond which it is interesting to choose x_i to be basic. This point is quite easy to determine: we can increase c_i by the coefficient δ_i of x_i in the objective line (the marginal cost of the non-negativity of x_i, that is what we gain by keeping $x_i = 0$), without changing the optimality of the solution. If we increase it by exactly δ_i, then the solution becomes degenerate, and we can choose x_i to be basic. Increasing c_i by more than δ_i will force x_i to be basic.

If x_i is basic in the optimal solution, how much can we change c_i without modifying the basis? If we increase c_i, it will likely increase x_i as well while possibly decreasing the other variables. While decreasing c_i enough will force us to make x_i leave the basis. To be more precise, we have to compute the effect of c_i on the objective in terms of the non-basic variables. We know that the objective is given by:

$$z = c_B B^{-1} b + (c_N - c_B B^{-1} A_N) x_N$$

We want to compute the line of B^{-1} asociated with variable x_i, say it is the jth line, we want to solve $d = e_j B^{-1}$, that is $dB = e_j$. Then the effect of changing c_i by one unit on the objective will be $db - dA_N x_N$. Then we have to find the ratios that will give a non-negative value to at least one of the non-basic variables. This will give bounds on the change of c_i that does not affect the basis.

By working on the dual problem, we can get similar results for modification of the right-hand side of the constraints. But we can do it directly as well. By modifying b, we only modify the constant column of the optimal dictionary, that is given by $B^{-1}b$. Hence, changing b to b', we will get $B^{-1}b'$. We want to keep this vector non-negative, again this defines bounds between which we may change an individual parameter in the right-hand side, without changing the basis' for constraints that are tight.

We can also modify non-tight constraints: if $ax \leq b_i$ is non-tight, then increasing b_i will not change the solution at all (this constraint was not really constraining us, and now it is even looser), and decreasing it by the value of the slack variable will make it tight. Decreasing it further will then modify the value of the optimal solution.

Chapter 4

Applications

4.1 Matrix games

Consider a game with two players, Alice and Bob, where Alice has m possible actions, and Bob has n possible actions. Alice and Bob choose simultaneously their actions (hence they do not what their opponent choice), and depending of their choices, Alice gives to or receives from Bob a given amount of money. More precisely, if Alice chooses $i \in [1, m]$, and Bob chooses $j \in [i, n]$, then Alice receives a_{ij} dollars and Bob receives $-a_{ij}$ dollars (receiving a negative amount of money means paying this amount). Hence the game is defined by some matrix $A \in \mathbb{R}^{m \times n}$.

Some popular games can be stated within this setting. One of the most popular, *rock-paper-scissors*, is defined by the following matrix:

$$\begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix}$$

where *rock*, *paper* and *scissors* are respectively actions 1, 2 and 3. These games are called two-player zero-sum matrix games, as they are defined by a matrix, and the sum of the gains of the players is zero.

Suppose that the matrix A in Alice and Bob's game is defined by:

$$\begin{pmatrix} 1 & 2 & -2 \\ 3 & -1 & -1 \\ -2 & 1 & 4 \end{pmatrix}$$

What is a good strategy for the two players? Consider the case of Alice. If she plays 1, then she can hope to gain 2 if Bob plays 2, but Bob could as well play 3, in which case she would lose 2. Playing 2 or 3 would also lead to a bad response from Bob. It is not possible for her to be sure to gain something. The situation is as bad for Bob, who has no trivial winning choice.

A *strategy* is how a player chooses an action. For instance, one of Alice's strategy is to play 2. A *pure strategy* is a strategy where the player choose an action determinisically. Hence, Alice has three possible pure strategies, as well as Bob.

Bob, seeing no way to ensure a win wit ha pure strategy, decides to play randomly. He chooses to play each possible action with the same probablity. His strategy can be written as a (column) vector with non-negative coefficients whose sum is 1, here it is $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})^T$ (those vectors are called stochastic). Each coefficient gives the probability that Bob plays that action. Hence a pure strategy is a special case of a *mixed strategy*, where one coefficient is 1 and the others are 0. Suppose that Bob tells Alice that this game is boring and that he has decided to play the mixed strategy $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})^T$. Then Alice can find a much better way to play by analysing her expectation. If she plays 1, she expects to win

$$\frac{1}{3} \times 1 + \frac{1}{3} \times 2 + \frac{1}{3} \times -2 = \frac{1}{3}$$

Similarly, by playing 2, she expects to win $\frac{1}{3}$, and 1 by playing 3, hence she chooses to play 3.

More generally, if Alice knows Bob's strategy x, she want to find a strategy (pure or not), that maximizes her gain. She want to solve:

$$\begin{array}{rrcl} \max & yAx & \text{subject to} \\ & \sum_{i=1}^{m} y_i & = & 1 \\ & y & \geq & 0 \end{array}$$

Knowing x this is just a linear program, so she can use the simplex method to determine what is her best response strategy to Bob's strategy x. Moreover, she will get a pure strategy, because the polyhedron defined by the constraints has pure strategies as vertices. This is an important remark: given a strategy for one player, there is a best response form the other player that is a pure strategy.

In our example, she solves the following linear program:

$$\begin{array}{rrcrcrcl} \max & \frac{1}{3}y_1 & + & \frac{1}{3}y_2 & + & y_3 & \text{subject to} \\ & y_1 & + & y_2 & + & y_3 & = & 1 \\ & & & & & y_1, y_2, y_3 & \geq & 0 \end{array}$$

This gives as expected the solution $(0, 0, 1)$.

Then Alice and Bob start their game. After a few rounds, the results are as follow:

- First round, Alice chooses 3, Bob chooses 2. Alice gives 1 from Bob.
- Second round, Alice chooses 3, Bob chooses 1, Alice gives 2 to Bob.
- Third round, Alice chooses 3, Bob chooses 1, Alice gives 2 to Bob.
- Fourth round, Alice chooses 3, Bob chooses 3, Alice takes 4 from Bob.
- Fifth round, Alice chooses 3, Bob chooses 1, Alice gives 2 to Bob.

At this point, Bob notes that Alice always choose 3, and that it would be better for him to play only 1. But then, Alice would quickly notice that he changed his strategy, and she would then find a new one. And again, Bob would change his strategy in response. One possible question is then: are there a strategy for Alice and one for Bob, such that none of them would want to change their strategies? More exactly, Alice would have no reason to change her strategy if Bob keeps the same, and conversely Bob would have no interest to change if Alice does not.

Another question is: is there a strategy that ensures Alice some gain, whatever is Bob's strategy? In other word, Alice wants a strategy y^* such that for every possible strategy x for Bob, Alice would expect a gain of at least some value c. This is definitely possible, if $c = -2$, Alice cannot lose more than 2 per round. But Alice wants to find c as big as possible. Actually, Alice wants to solve the following problem:

$$\begin{array}{rrcl} \max_y & \min_x yAx & \text{subject to} & \\ & \sum_{j=1}^n x_j & = & 1 \\ & \sum_{i=1}^m y_i & = & 1 \\ & x, y & \geq & 0 \end{array}$$

We rewrite it in the following form. Note that this is not a linear program, as we have quadratic terms in the constraints.

$$\begin{array}{rrcll} \max & z & \text{subject to} & & \\ & z & \leq & yAx_j & (\forall j \in [1, n]) \\ & \sum_{j=1}^n x_j & = & 1 & \\ & \sum_{i=1}^m y_i & = & 1 & \\ & x, y & \geq & 0 & \end{array}$$

This says that Alice wants to maximize her gain, supposing that Bob will play the worst strategy for her (that is his own best strategy), knowing Alice's strategy. As we have seen before, if Bob knows Alice strategy, he can easily find a good strategy by evaluating each of its pure strategies, and then keep the best one. It means that in (4.1), Alice only needs to consider Bob's pure strategies. We hence get the following simpler and linear program:

$$\begin{array}{lrcll} \max & z & \text{subject to} & & \\ & z & \leq & yA_{\bullet j} & (\forall j \in [1,n]) \\ & \sum_{i=1}^m y_i & = & 1 & \\ & y & \geq & 0 & \end{array}$$

In the case of Alice, it gives:

$$\begin{array}{llcrcrcrcl} \max & z & \text{subject to} & & & & & & & \\ & z & - & y_1 & - & 3y_2 & + & 2y_3 & \leq & 0 \\ & z & - & 2y_1 & + & y_2 & - & y_3 & \leq & 0 \\ & z & + & 2y_1 & + & y_2 & - & 4y_3 & \leq & 0 \\ & & & y_1 & + & y_2 & + & y_3 & = & 1 \\ & & & & & & & y_1, y_2, y_3 & \geq & 0 \end{array} \tag{4.1}$$

and the solution is $\frac{1}{39}(12, 11, 16)$ with an optimal value of $\frac{29}{39}$. With this strategy, Alice is sure to win $\frac{29}{39}$ in average, whatever Bob chooses to do. The dual program should give us a proof that she cannot ensure a better gain. Let's write the dual program in the general case:

$$\begin{array}{lrcll} \min & w & \text{subject to} & & \\ & \sum_{j=1}^n x_j & = & 1 & \\ & w - A_{i\bullet}x & \geq & 0 & (\forall i \in [1,m]) \\ & x & \geq & 0 & \end{array} \tag{4.2}$$

This linear program is what we would have got by simplifying Bob's problem for finding a best strategy, supposing Alice plays her best response:

$$\begin{array}{llcl} \min_x & \max_y yAx & \text{subject to} & \\ & \sum_{j=1}^n x_j & = & 1 \\ & \sum_{i=1}^m y_i & = & 1 \\ & x, y & \geq & 0 \end{array}$$

So we get that the dual program of (4.1) is finding a best strategy for Bob. In our case, Bob should play the mixed strategy $\frac{1}{39}(17, 14, 8)$, and he will lose in average only $\frac{29}{39}$. The value achieved by the linear program is called the *value* of the game: this is the best value that the firt player can expect to win.

There are some interesting points about those solutions:

- The min-max theorem for game theory is equivalent to the duality theorem of linear programming. Actually, they were discovered independantly by Von Neumann (for the game theory version), and Gale, Kuhn and Tucker (for the linear programming version).

- If both Alice and Bob plays with these optimal strategies, neither of them is interested in changing its strategy. If Alice want to change and Bob keeps the same strategy, Alice will get a smaller gain, and conversely for Bob. Such a pair of strategies is called a *Nash equilibrium*, the name reflecting the fact that no player is willing to change the present state.

- The fact that a strategy is optimal here is not affected by the knowledge of that strategy by the opponent. Alice can explicitely reveal her mixed strategy to Bob, she will still gain in average the same value.

- The best strategy for Bob is always a linear combination of best pure response to Alice strategy (this is the game-theory equivalent to complementary slackness).

- Not all games have a Nash equilibrium, and some may have multiple Nash equilibria with different values. We can define more complex games, non-zero sum games, where the gains of the two players are not related. We define two $m \times n$ matrices A and B. A gives the payoffs for Alice, while B gives the payoffs for Bob. A common way to represent these two payoff matrices is to give a single matrix, whose coefficients are couple: the first term is the gain for Alice, the second is the gain for Bob. For example, we could have this matrix:

$$\begin{pmatrix} (3,4) & (-1,-2) \\ (-2,-1) & (4,3) \end{pmatrix}$$

  In that game, if Alice chooses 1 and Bob chooses 2, then Alice loses 1 and Bob 2. There are two pure Nash equilibria: Alice and Bob both play 1, or they both play 2.

- Matrix games are just a very small part of game theory: it is useful for two-player games with complete knowledge and finitely many pure strategies.

A nice example of study of a two-player game is Kuhn's poker. Kuhn's example shows that bluffing is a rational strategy in some games. The rules are the following: Alice and Bob bet one unit, and receive one card each, chosen uniformly among a deck of three cards numbered from one to three. The third card is kept unseen They know what is their own cards but not the opponent's ones. Then Alice and Bob take turns, by either betting one more unit or passing. The game ends as soon as the two players passes, the two players bets, or one bets and then the other passes. Here are the possibilities:

- Alice passes, Bob passes. The player with the highest card takes the pot of 2,
- Alice passes, Bob bets, Alice passes. Bob takes the pot of 3.
- Alice passes, Bob bets, Alice bets. The player with the highest card takes the pot of 4.
- Alice bets, Bob passes. Alice takes the pot of 3.
- Alice bets, Bob bets. The player with the highest card takes the pot of 4.

For example, suppose that Alice gets a 2, and Bob a 3. Alice chooses to pass, then Bob bets, then Alice passes againt. Alice loses 1 (her initial bet), and Bob gains 1 (the pot is 3 and he bets 2).

The possible strategies for Alice, depending on her card, are:

- passing, then passing if Bob bets.
- passing, then betting if Bob bets.
- betting.

Hence one pure strategy could be: pass if she has a 1, bet if she has a 2, pass then bet if she has a 3. This gives 27 pure strategies. For Bob, again depending on the card:

- if Alice passes, bet, else pass.
- if Alice passes, pass, else bet.
- bet, whatever does Alice.
- pass, whatever does Alice.

Thus Bob has 64 pure strategies. We can reduce this number, as some strategies are clearly stupid. Bob will never pass if he has a 3. Alice will not bet on the third round if she has a 1. Then, if Alice has a 2, we have the following payoff matrix:

Alice's first move	Bob has a 1	Bob has a 3
bets	$+1$	-2
passes	≥ 1	≥ -2

Her payoff is better when passing, hence she will never bet on first move when she has a 2. The possible strategies for Alice are:

1. with a one: always pass, with a two: always pass, with a three: pass then bet.
2. with a one: always pass, with a two: always pass, with a three: bet.
3. with a one: always pass, with a two: pass then bet, with a three: pass then bet.
4. with a one: always pass, with a two: pass then bet, with a three: bet.
5. with a one: bet, with a two: always pass, with a three: pass then bet.
6. with a one: bet, with a two: always pass, with a three: bet.
7. with a one: bet, with a two: pass then bet, with a three: pass then bet.
8. with a one: bet, with a two: pass then bet, with a three: bet.

Similarly, if Bob holds a 2 and Alice passes on her first move, then it would be stupid for Bob to bet: it could only lead to a better gain for Alice. Hence Bob has four different strategies. In each of them, always bet with a 3, pass with a two if Alice pass, pass with a one if Alice bets. The differences are:

1. with a 1: bet if Alice passes, with a 2: bet if she bets.
2. with a 1: bet if Alice passes, with a 2: pass if she bets.
3. with a 1: pass if Alice passes, with a 2: bet is she bets.
4. with a 1: pass if Alice passes, with a 2: pass if she bets.

This gives the following payoff matrix, by considering each deal with equal probability.

$$\frac{1}{6}\begin{pmatrix} -1 & -1 & 0 & 0 \\ -1 & -1 & 1 & 0 \\ 1 & 1 & -1 & -1 \\ 1 & 0 & 0 & -1 \\ -3 & 0 & -2 & 1 \\ -3 & -1 & -1 & 1 \\ -1 & 2 & -3 & 0 \\ -1 & 1 & -2 & 0 \end{pmatrix}$$

This gives the following linear program for Alice:

$$\begin{array}{ll}
\max & g \text{ subject to} \\
& g + \frac{1}{6}x_1 + \frac{1}{6}x_2 - \frac{1}{6}x_3 - \frac{1}{6}x_4 + \frac{1}{2}x_5 + \frac{1}{2}x_6 + \frac{1}{6}x_7 + \frac{1}{6}x_8 \leq 0 \\
& g + \frac{1}{6}x_1 + \frac{1}{6}x_2 - \frac{1}{6}x_3 + \frac{1}{6}x_6 - \frac{1}{3}x_7 - \frac{1}{6}x_8 \leq 0 \\
& g - \frac{1}{6}x_2 + \frac{1}{6}x_3 + \frac{1}{3}x_5 + \frac{1}{6}x_6 + \frac{1}{2}x_7 + \frac{1}{3}x_8 \leq 0 \\
& g + \frac{1}{6}x_3 + \frac{1}{6}x_4 - \frac{1}{6}x_5 - \frac{1}{6}x_6 \leq 0 \\
& x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 = 1 \\
& x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 \geq 0
\end{array}$$

One possible couple of solutions (among many) is $\left(\frac{1}{3}, 0, 0, \frac{1}{2}, \frac{1}{6}, 0, 0, 0\right)$ and $\left(\frac{1}{3}, 0, 0, \frac{2}{3}\right)^T$, and the value of the game is $-\frac{1}{18}$. With this strategy, Alice bluffs with a one with probability $\frac{1}{6}$, and with a 3 with probability $\frac{1}{2}$. And Bob bluffs with a 1 with probability $\frac{1}{3}$.

4.2 Column generation

The cutting-stock problem consists in finding how to cut rolls (called *raws* by that industry) of raw material (metal sheet, textile, paper) from one (large) width, to several smaller widths (called *finals*). The rolls are cut orthogonally to their axis. Suppose for example that we dispose of raws of width 120cm. We are asked to deliver 65 finals of width 45cm, 56 finals of width 37cm, and 78 finals of width 22cm. We want to cut a minimal number of rolls to satisfy this demand, and all the parts not used in a cut roll are considered to be lost.

To modelize this problem, we first need to give decision variables. To each decision variable, we will associate a way to cut a roll, then that variable gives the number of raws cut in this way. One way of cutting a raw would be to cut it into two finals of 45cm plus one of 22cm, and 8cm are lost. We want to enumerate all the possible ways of cutting a raw of width 120cm into finals of size 45cm, 37cm and 22cm. Actually, we do not need to compute them all, but only does that are maximal, that is we cannot add them one more final (then, we will try to find more finals that we really want, and the potential extras will be considered as lost). This is given by Figure 4.1.

With that, it is quite easy to write a linear program solving (a fractional version of) our problem. We must express the demand for each kind of final,

Variable name	45cm	37cm	22cm	lost
x_1	2	0	1	8cm
x_2	1	2	0	1cm
x_3	1	1	1	16cm
x_4	1	0	3	9cm
x_5	0	3	0	9cm
x_6	0	2	2	2cm
x_7	0	1	3	17cm
x_8	0	0	5	10cm

Figure 4.1: *Decision variable for our cutting-stock problem.*

this gives:

$$\begin{array}{rcl} 2x_1 + x_2 + x_3 + x_4 & \geq & 65 \\ 2x_2 + x_3 + 3x_5 + 2x_6 + x_7 & \geq & 56 \\ x_1 + x_3 + 3x_4 + 2x_6 + 3x_7 + 5x_8 & \geq & 78 \\ x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 & \geq & 0 \end{array} \tag{4.3}$$

Then the objective is minimizing the total number of raws used, that is:

$$\min x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 \text{ subject to (4.3)}$$

An optimal solution is $\frac{1}{5}(152, 21, 0, 0, 0, 119, 0, 0)$ with objective value $58 + \frac{2}{5}$. This is not really satisfactory because we want an integer solution, but a good solution can be find by rounding these values down to $(30, 4, 0, 0, 0, 23, 0, 0)$. With that partial solution, we still need 1, 2 and 2 finals of size 45cm, 37cm and 22cm, which can be done with only two more raws ($x_2 + x_6$ for example), for a total of 59 raws. As the fractional optimal is strictly more than 58, this is an optimal integral solution, but we would have been satisfied even with 60 raws, as long as we do not use to many additional raws in the rounding operation.

This method looks fine, but there is actually one problem: when the number of finals with different width increases, the number of decision variable increases exponentially. For example, if we still have raws of size 120cm, but finals of size 12cm, 17cm, 22cm, 25cm, 29cm, 33cm, 35cm, 42cm and 53cm, the number of way to cut a raws into finals is... quite big. And this is still as small example. Moreover, there can be different sizes of raws, and possibly additional constraints that make things even uglier. Anyway, we can not compute every possible way of cutting a raw, as there are far too many.

Fortunately, in the revised simplex method, we do not need to know each variable individually. All we need is to know which are the basic variables, and find a potential entering variable. We start from any basic solution. It is better to use a simple heuristic to find a solution which is already close to the optimal. For example, we can use a greedy algorithm, consisting in packing the finals in lexicographic order: first the largest one, until there is not enough space left, then the second largest, etc. For example, consider the following finals, with raws of width 120cm:

width	47cm	43cm	36cm	32cm	21cm
demand	50	44	48	28	43

Then, with the greedy algorithm, we consider to cut a raw into two finals of size 47cm (three would be too much), it leaves 26cm, in which we add a final of width 21cm. Hence our first variable x_1 corresponds to $(2, 0, 0, 0, 1)$, and in the first solution we can take $x_1 = 25$. We still have a demand of $(0, 44, 50, 29, 18)$, we introduce x_2 corresponding to $(0, 2, 0, 1, 0)$, as we can place at most to finals of size 43cm, leaving the place for one final of size 32cm. We choose $x_2 = 22$ to satisfy the demand for 43cm finals. Note that with this procedure, we may find a non-integral solution (otherwise, the solution may not be basic). At the end, we will get this initial solution:

variable	47cm	43cm	36cm	32cm	21cm	initial value
x_1	2				1	25
x_2		2		1		22
x_3			3			16
x_4				3	1	2
x_5					4	4
total	50	44	48	28	43	69

We now want to find if there is a way to cut the raws that would give a better solution. The simplex method asks us to find a negative coefficient in $c_N - yA_N$ with $yB = c_B$. As we do not want to consider all the non-basic variables (there are too many), we only want to decide if there is one, x_j, with $c_j - e_j yA_N$ negative. First we must compute y.

$$y \begin{pmatrix} 2 & & & & \\ & 2 & & & \\ & & 3 & & \\ & 1 & & 3 & \\ 1 & & & 1 & 4 \end{pmatrix} = (1, 1, 1, 1, 1)$$

This gives $y = \left(\frac{3}{8}, \frac{3}{8}, \frac{1}{3}, \frac{1}{4}, \frac{1}{4}\right)$. This dual values corresponds to some proportion of a raw, representing the value that we associate to a final, so they should approximate the widths of the finals, which is roughly correct here.

The coefficients of c_N are all 1's, therefore we want to find a way to cut a raw, with y-value strictly more than 1. We want to find an integer solution to the following problem

$$\begin{array}{ccccccccccccc} 47k_1 & + & 43k_2 & + & 36k_3 & + & 32k_4 & + & 21k_5 & \leq & 120 \\ \frac{3}{8}k_1 & + & \frac{3}{8}k_2 & + & \frac{1}{3}k_3 & + & \frac{1}{4}k_4 & + & \frac{1}{4}k_5 & > & 1 \end{array} \tag{4.4}$$

Here a possible solution is given by $x_6 = (0, 0, 2, 0, 2)$, with total width 114 and dual value $\frac{7}{6}$. We will not show how to find a solution to a problem like (4.4) (what is difficult here is to find an integral solution, for a non-integral solution we could write it as a linear program), solving this kind of problems is the subject of the following course, *Discrete Optimization 2*. Our entering variable is x_6, we must find the leaving variable, as always by computing the column corresponding to the entering variable. We solve:

$$d = \begin{pmatrix} 2 & & & & \\ & 2 & & & \\ & & 3 & & \\ & 1 & & 3 & \\ 1 & & & 1 & 4 \end{pmatrix} (0, 0, 2, 0, 2)^T$$

giving $d = \left(0, 0, \frac{2}{3}, 0, \frac{1}{2}\right)^T$, it gives a ratio of 8 with leaving variable x_5. the new solution is then $(25, 22, \frac{16}{3}, 2, 8)$ with basis given by:

$$B = \begin{pmatrix} 2 & & & & \\ & 2 & & & \\ & & 3 & & 2 \\ & 1 & & 3 & \\ 1 & & & 1 & 2 \end{pmatrix}$$

We start a new iteration, by computing the dual values corresponding to each final width. $yB = (1, 1, 1, 1, 1)$ gives $y = \left(\frac{5}{12}, \frac{13}{36}, \frac{1}{3}, \frac{5}{18}, \frac{1}{6}\right)$ (this again is a good approximation of the widths, this is a good indication that we are right). Then $x_7 = (0, 1, 2, 0, 0)$ is a candidate for entering the basis, because the total width for this cut is 115cm, and the dual value is $\frac{37}{36}$. We find the corresponding column:

$$\begin{pmatrix} 2 & & & & \\ & 2 & & & \\ & & 3 & & 2 \\ & 1 & & 3 & \\ 1 & & & 1 & 2 \end{pmatrix} d = (0, 1, 2, 0, 0)^T$$

We find $d = \left(0, \frac{1}{2}, \frac{11}{18}, -\frac{1}{6}, \frac{1}{12}\right)^T$. Then x_3 is leaving, the ratio is $\frac{288}{33} \approx 8.7$, the new solution is $\left(25, \frac{146}{11}, \frac{192}{11}, \frac{54}{11}, \frac{72}{11}\right)$ with matrix:

$$B = \begin{pmatrix} 2 & & & & \\ & 2 & 1 & & \\ & & 2 & & 2 \\ & 1 & & 3 & \\ 1 & & & 1 & 2 \end{pmatrix}$$

It is time for the next iteration, but we stop here. After a few more iterations we would have reached the solution with basis:

$$\begin{pmatrix} 2 & & 1 & & \\ & 2 & & 1 & \\ & & 2 & & 1 \\ & 1 & & 3 & \\ 1 & & & 1 & 4 \end{pmatrix}$$

and the basic solution is $\frac{1}{17}\left(\left(\right) 249, 374, 352, 34, 112\right)$ and has an objective value of $\frac{1121}{17} \approx 65.95$. Rounded down, this gives $(14, 22, 20, 2, 6)$, and we have still to find $(2, 0, 2, 0, 3)$ finals, this can easily be done with two additional rows, giving a solution with only 66 rows. This solution is necessarily optimal as the optimal fractional solution is strictly larger that 65.

What we learn from this problem is that thanks to the revised simplex method, we do not even have to manipulate all the variables. We only need to keep the value of the basic variables, and to be able to find an entering variable or prove that none exists. In particular, when we face a problem where the set of possibilities is very huge, we can only work with a subset of these possibilities, and add those that are interesting when we need them to improve the solution.

Finally, let us say a word about the problem of finding an entering variable for the cutting-stock problem. This is a special case of the knapsack problem: given a set of objects, each defined by a weight and a utility, we want to maximise the sum of the utilities of the object that we put in our bag, without exceding the weight that we can carry. Finding an integral optimal solution is theoretically difficult, but some algorithms perform very well. The most popular is a *branch-and-bound* technic: we explore the set of all possibilities. This set can be represented as a tree, where a solution A is an ancestor of a solution B, if B is a subset of A. Instead of exploring all the tree, we try to cut branches that are not promising: if we are able to find an upper bound of the utility of the best solution in a branch, and this upper bound is less than a solution that we already have computed, we do not explore the branch. All the efficiency of this technic lies in our capacity to find good upper bounds for branches, and also a good initial solution. These upper bounds are usually provided by duality, and this is a reason why duality (or more generally, finding upper bounds) is very important in practice.

Chapter 5

The network simplex

The network simplex is a specialization of the revised simplex method to transportation problems in networks. Networks are modelized as *graphs* with capacities. We will see that again we only need to consider basic solutions, which in this case can be determined by a very special structure in the graph representing the network. First, we will give basics in graph theory, before introducing the problem that we want to solve. Then we will see how the simplex method can be simplified to solve this problem. Finally we will give a list of different problems that can be solved by the simplex method, and deduce nice combinatorial results only from the theory of linear programming.

5.1 Introduction to graph theory

A (simple, loop-free) *directed graph* is a couple $G = (V, E)$, where V is a finite set and $E \subseteq V^2 \setminus \{(v, v) \mid v \in V\}$. In our application, V can be understand as a set of locations (for example cities), and E describes the transportation possibilities between cities: an element of E could represent a one-way highway or railway for example. Having only one-way highway is no loss of generality, as a two-ways highway can be represented as two one-way highways. The set of vertices of G will be denoted by $V(G)$, its edges by $E(G)$. We denote $n = |V(G)|$ and $m = |E(G)|$.

An element of V is called a *vertex*, and an element of E is called an *arc*. The name vertex comes from the fact that vertices and edges of a 3-dimensional polytope P define a graph, where V is the set of the vertices of P, and if there is an edge between the vertices u and v in P, then one of (u, v), (v, u) is in E. Note that in that case, we do not want to make a

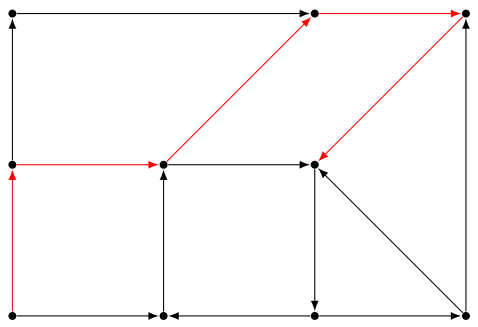


Figure 5.1: *An example of graph, and a directed path in red.*

distinction between (u, v) and (v, u), graphs for which this is true are called *undirected graphs*, and there arcs are called *edges*. We will use the term *edge* when the orientation does not matter. Any directed graph induces an undirected graph, simply by neglecting the orientation. But we will only work with directed graphs.

Small graphs are usually represented in figures in the following way. To each vertex is associated a distinct point in the plane. Then, for each arc (u, v), we draw an arrow from the point representing u, to the point representing v. Figure 5.1 illustrates a graph with 10 vertices and 15 arcs.

We will note uv for the arc (u, v). u is the *source* or *origin* of *tail* of uv. v is the *sink* or *destination* or *head* of uv. A *directed path* is a sequence of distinct arcs $u_1v_1, \dots, u_kv_k$, with $v_i = u_{i+1}$ for all $i \in [\![1, k-1]\!]$. If $u_1 = v_k$, this is called a *directed cycle*. For a path P, we will refer to the set of its edges as $E(P)$, and its vertices (that is the vertices of its edges) as $V(P)$. u_1 is the *source* or *origin* of the path, v_k is the *sink* or *destination* of the path. u_1 and v_k are its *extremities*.

For a vertex v, we denote by $\delta^+(v)$ the set of arcs having tail v, this is the set of *leaving arcs* of v. Similarly, $\delta^-(v)$ is the set of *entering arcs* of v, the arcs having v for head. The set of *incident* arcs of v is $\delta(v) = \delta^+(v) \cup \delta^-(v)$. The *degree* of a vertex is the number of its incident arcs, it is denoted $d(v)$. Similarly, $d^+(v) := |\delta^+(v)|$ and $d^-(v) := |\delta^-(v)|$ are the *out-degree* and *in-degree* of v.

Undirected paths and *undirected cycles* are similar, except that we neglect the orientation (equivalently, we can use arcs in the wrong direction): an *undirected path* is a sequence $u_1u_2, \dots, u_{k-1}u_k$, where u_iu_{i+1} or $u_{i+1}u_i$ is an arc of G for all $i \in [\![1, k-1]\!]$. Moreover, all the arcs must be distinct. $u_{i+1}u_i \in E(P)$ is then called a *backward arc*, while $u_iu_{i+1} \in E(P)$ is a *forward arc*. In this way, forward arcs are the arcs taken in the "good" direction, while backward arcs are the arcs taken in the "wrong" direction. Directed paths are special cases of undirected paths, that have only forward arcs. A graph is *acyclic* if it has no cycle.

A graph is connected if for every pair (u, v) of vertices, there is an undirected path having u and v as extremities. The relation "there is an undirected path between u and v" is an equivalence relation, we only need to show the transitivity (that s also true for directed paths, but the directed paths relation is not symmetric):

Proposition 5.1.1. If there is a (directed) path from u to v and a (directed) path from v to w, then there is one from u to w.

Proof. Let $u_1u_2, \ldots, u_{k-1}u_k$ be a path with $u_1 = u$ and $u_k = v$. Let $v_1v_2, \ldots, v_{l-1}v_l$ be a path with $v_1 = v$ and $v_l = w$. We would like to say that $u_1u_2, ldots, u_{k-1}u_k, v_1v_2, \ldots, v_{l-1}v_l$ is a path, using $u_k = v_1$, but this is not true as an arc can appear twice in this sequence. Let i be the minimal index such that u_i is some v_j. Then $u_1u_2, \ldots, u_{i-1}u_i, v_jv_{j+1} \ldots, v_{l-1}v_l$ is a path between u and w. Indeed, all the arcs are distinct by the choice of i. □

The equivalence classes are called the *connected components* of the graph. A connected graph is then a graph with only one connected component. A connected component is a maximal subset of vertices such that for any two of them there is a path between them. Two vertices are in different connected components iff there is no undirected path between them. (We could define strongly connected components by replacing undirected paths by directed paths in this characterization, but we will not use them). We will only consider connected graphs in the following (non-connected graphs are not interesting, in the sense that solving a problem on a non-connected graph is usually the same as solving it in all its connected components).

A very useful result gives a necessary and sufficient condition for having a directed path between two vertices, or even two sets of vertices. We state and prove it:

Lemma 5.1.2. *Let $G = (V, E)$ be a directed graph, and $X, Y \subset V$ two disjoint subsets of vertices. There is a directed path with origin in X and destination in Y iff there is no set C with $X \subseteq C \subseteq V \setminus Y$, such that there is no arc with tail in C and head in $V \setminus C$.*

Proof. Let C be a set with $X \subseteq C \subseteq V \setminus Y$, and $P : u_1u_2, u_2u_3, \ldots, u_{k-1}u_k$ a directed path, with $u_1 \in X$ and $u_k \in Y$. Let i be the minimal index such that $u_i \notin C$. $i > 1$ as $u_1 \in X$, and $i < k$ as $u_k \in Y$. Then the arc $u_i u_{i+1}$ has its tail in C and its head in $V \setminus C$. This proves the necessity.

We prove the sufficiency. Let R be the set of vertices r for which there is a path with origin in X and destination r. If $R \cap Y = \emptyset$, then $R \subseteq V \setminus Y$, and obviously $X \subseteq R$. We only have to prove that there is no arc with tail in R and head outside R. Let $e = uv$ be an arc with $u \in R$, then by Proposition 5.1.1, $y \in R$. □

Remark. The proof leads to the following algorithm to find a directed path form u to v. Suppose that we found a set R of vertices reachable by a directed path from u, and that for each vertex r in R, we have a directed (u, r)-path. Then, consider the arcs with tail in R and head outside R. If there is none, we are done, we get a (u, v)-path if $v \in R$, otherwise there is no directed (u, v)-path. Else, there is an arc rs with $r \in R$, $s \notin R$. Then we get a new set of reachable vertices $R \cup \{s\}$, and we build a (u, s)- path by adding rs to the (u, s)-path already found. We can then iterate this procedure until termination.

The efficiency of this algorithm depends on the data structure used to find arcs leaving R. Usually, we keep a set of arcs to consider. When we found a new reachable vertex s, we add to this set all the arc leaving s. To find an arc leaving arc, we consider the arcs in our set and remove them, until we get one leaving R, or there is no more arc. In this way, every arc will be considered twice: when it is added to the set of arcs, and whenit is removed from that set. This gives an algorithm with a complexity depending on the number of edges in the graph.

The same algorithm applies to find paths in undirected graphs.

A subgraph of $G = (V, E)$ is a graph (V', E') with $V' \subseteq V$ and $E' \subseteq E \cap V'^2$. A subgraph (V', E') is *spanning* if $V' = V$ and every vertex is contained in at least on arc of E'. We will identify a subset E' of arcs with its induced subgraph (V', E'), where V' is the set of vertices contained in at least one arc of E'. Our main tool for the network simplex will be maximal acyclic subgraphs. This leads to the definition of a *spanning tree*, which is a maximal acyclic subgraph of G. A *tree* is just an acyclic connected graph (the name is quite obvious, as can be seen in Figure 5.2).

Theorem 5.1.3. *$T \subset E(G)$ is a spanning tree (maximal acyclic subgraph) of G iff T is a minimal spanning connected subgraph of G.*

Proof. Suppose that T is a spanning tree. We must prove that T is connected, and removing any edge from T would disconnect it. Suppose it is not connected, then there is u and v in $V(G)$ such that there is no path

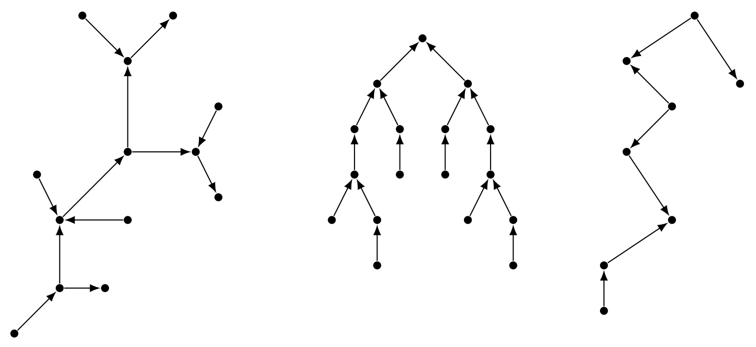


Figure 5.2: *Examples of three different trees. The tree in the middle is a directed tree: every arc is directed toward a unique node. The rightmost one is degenerate, it is only a path (but still a tree).*

in T between u and v. But G is connected, hence there is a path P in G with extremities u, v. Starting from u, consider the first edge $e = st$ in P such that there is a path in T between u and s but not between u and t. We claim that $T \cup st$ has no cycle, which contradicts the maximality of T. Indeed, if there is a cycle in $T \cup st$, this cycle must contain st because T is acyclic. Removing st from that cycle gives a path with extremities s and t, then there is also a path between u and t, contradiction. Then, for any edge uv in T, there is no path between u and v in $T \setminus \{uv\}$, otherwise such a path plus uv would be a cycle in T, but T is acyclic, contradiction.

Suppose now that T is a connected subgraph of G. If T has a cycle C, then we can remove any edge of the cycle without violating the connectivity: indeed if we remove uv, the other edges of a cycle define a path with extremities u and v. Take any pair of vertices s and t, and choose a path in T between them. If this path does not use uv we are done, else it gives two paths, one between s and u, and the other between v and t (without loss of generality). Then s, u, v and t are in the same component in $T \setminus \{uv\}$, so there is a path between s and t. Hence, a minimal spanning connected subgraph is acyclic. Moreover, adding any edge would create a cycle, by adding the path connecting the two extremities of the added edge. Therefore, T is maximally acyclic, proving the theorem. □

Remark. It is not hard to build a spanning tree of a connected graph. Use the algorithm that find paths from u to any vertex. Then, the set of edges used by the algorithm to increase the set of reachable vertices R is a spanning tree: it is clear that it is connected, as it connects every vertex to u. And it is acyclic, as during the algorithm, we only add edges between on vertex already reached, and one vertex not reached, so we do not create cycle.

Actually, we can even find minimum spanning tree: if we give a cost to each edge, we can find a spanning tree minimizing the sum of the cost of its edges.

We will use the following fact.

Proposition 5.1.4. Every acyclic subgraph (in particular a tree) has at least two vertices with degree 1.

Proof. Take a maximal path (as an edge set) of G. This exists, because G has no cycle. Then the extremities of this path have degree 1. $\square$

Remark. If G has n vertices, a spanning tree of G has $n-1$ edges. This can be seen by induction: take a tree T, then there is a vertex v in the tree with degree one, let e be the edge of T incident to v. Let $G' = G - v$ and $T' = T - e$, T' is connected, acyclic and spanning, hence it is a spanning tree og G'. By induction hypothesis, we are done. We will see an algebraic proof of this fact latter.

5.2 Network flows: basis

5.2.1 What do we want to solve?

As we said in the introduction of this chapter, we want to solve transportation problems, and we modelize the infrastructure on which we transport by a graph. Vertices represent locations and facilities, arcs represent the fact that we can transport material between two given locations corresponding to the two extremities of an arc. For example, vertices could be cities and arcs highways.

Usually, we need more information to describe one arc. We need to know its capacity, that is how much could we transport on that arc. We will denote the capacities by a function $u : E(G) \to \mathbb{R}^+$, that maps each arc to a non-negative real number. u stands for *upper bounds.* and we will also allow ourselves to have lower bounds $l : E(G) \to \mathbb{R}^+$, with $l(e) \leq u(e)$ for each arc. We also need to give a cost for each arc, corresponding to how much its cost us to transport one unit along that arc. We will assume that these costs are linear: transporting two units costs twice the price of transporting one unit. Then costs are defined by a function $c : E(G) \to \mathbb{R}^+$. This should be enough to describe a transportation network in an abstract way.

Then we have to specify what we want to transport. We will only transport one kind of good (or commodity). These goods are available in limited quantity in some parts of the network, and must be transported to some other parts of the network. We are not concerned by the destination of each one, in the sense that we make no distinction between one unit coming from one place and one unit coming from another place. Thus we only specify where are the goods at the beginning, and where they must be at the end. We give a function $d : V(G) \rightarrow \mathbb{R}$. For a vertex v, if $d(v)$ is negative, it means that $d(v)$ units are available in v at the beginning (v is a *source*), while if $d(v)$ is positive, $d(v)$ units must be transported to the vertex v (v is a *sink*). d is refered as the *demand* function. Obviously, if $d(v) = 0$, it means that there is no supply here, and we do not want to deliver anything at that location. But this vertex can still be useful to route the demand.

Hence, a *network flow* problem is defined by a graph $G = (V, E)$, two capacity functions on the edges $l : E \rightarrow \mathbb{R}^+$ and $u : E \rightarrow \mathbb{R}^+$, a cost function $c : E \rightarrow \mathbb{R}^+$, and a demand function $d : V \rightarrow \mathbb{R}$. We will also assume that the sum of the demands is zero: $\sum_{v \in V} d(v) = 0$. That is, every unit available in a source must be routed to a sink.

A solution is to match units of supply and units of demand, and for each pair, give a path in the graph between the source of the supply and the sink of the demand. These paths must satisfy the capacity constraints: for each edge e, e is used by between $l(e)$ and $u(e)$ paths.

The number of paths could be very large, so it is not efficient to take decision variables corresponding to paths. What we do is that we only give for each arc, how many units must be transported along that arc. For each arc $e \in E$, we have a variable x_e with the following constraint:

$$l(e) \leq x_e \leq u(e) \tag{5.1}$$

In order to do this, we must make sure that there is enough unit at the tail of an arc, to be transported to its head. For that, we count the units entering any vertex minus the unit leaving any vertex. This amount must be equal to the demand for that vertex: for a vertex v, if i units enter and j leaves, it will remain $j - i$ units at the end less that at the beginning, so we want $i - j = d(v)$, In particular, if $d(v)$ is equal to zero, all that enters must leave. This is known as the *conservation law*, or *Kirchhoff's law*:

$$\sum_{e \in \delta^-(v)} x_e - \sum_{e \in \delta^+ v} x_e = d(v) \tag{5.2}$$

With this strategy, for a given unit of demand, starting from a given point, we do not know in advance where it will end, but we do not mind as we supposed that the units are not distinguishable. We need the following theorem:

Theorem 5.2.1 (Fulkerson)**.** *Let x be a vector on edges satisfying constraints* (5.1) *and* (5.2)*, then there is a set $\mathcal{P}$ of directed paths and a function $\lambda : \mathcal{P} \to \mathbb{R}^+$, such that $\sum_{P \in \mathcal{P}, P \ni e} \lambda_P = x_e$ for all arc e.*

This theorem just says that for a vector x satisfying bounds and conservation lows, we can find a trajectory (a directed path) for each unit of demand from a supply vertex ($d(v) < 0$) to a demand vertex ($d(v) > 0$), respecting all the capacity constraints. The converse of the theorem is trivial.

Proof. (sketch) There is a directed path form a source to a sink that uses only arcs with $x_e > 0$. Indeed, take the subgraph G_x corresponding to arcs with $x_e > 0$. Let C be a subset of vertices containing all the sources but no sink. Then, we add the conservation law constraints corresponding to all the vertices in C. As every arc with both end in C appears once positively, once negatively, it only remains the arcs with tail in C and head outside C in the left-hand side, and the right-hand side is negative, so there must be one arc leaving C with positive value. By Lemma 5.1.2, this proves the existence of our directed path $v_1 v_2, \dots, v_{k-1} v_k$.

We may assume that $d(v_i) = 0$ for all $i \in [\![2, k-1]\!]$, $d(v_1) < 0$ and $d(v_k) > 0$ (by removing vertices at the extremities of the path, until we get this property). Let $m = \max\{d(v_k), -d(v_1), \max_{e \in P} x_e\}$, take P in $\mathcal{P}$ with $\lambda(P) = m$. Then decrease x_e by m for each edge of P, increase $d(v_1)$ by m, decrease $d(v_k)$ by m. The new vectors x' and d' satisfy the conservation law, hence we can iterate this procedure until the demand vector is null. This procedure terminates, because at each step, we either have one more arc e with $x_e = 0$, or one more vertex with $d(v) = 0$. It is easy to check that λ satisfy the condition. $\square$

This proof is constructive and directly gives an algorithm. Therefore, we only want to solve the following problem:

$$\begin{array}{ll} \min & cx \text{ subject to} \\ & \sum_{e \in \delta^-(v)} x_e - \sum_{e \in \delta^+(v)} = d(v) \qquad \text{(for all } v \in V) \\ & l \leq x \leq u \end{array} \tag{5.3}$$

This is a linear program, we know how to solve it. But we will see that we can say a lot more actually, in solve them more efficiently than by the simplex method.

Remark. The vector b can be chosen to be null. We rewrite this program into an equivalent program in which $d = 0$. As we do not distinguish the supply, we may as well add a new vertex s, with an arc to each source v with capacities $l(sv) = u(sv) = -d(v)$. We then reduce the supply on v to zero, and add this quantity to s: $d(s) = \sum_{\text{source}\, v} d(v)$. It means that we consider that all the supply come form only one vertex, and are distributed to the original sources. We can do the same with the sinks: add a vertex t with an arc from each original sink v, with capacities $l(vt) = u(vt) = d(v)$. Then define $d(t) = \sum_{\text{sink}\, v} d(v)$, and now $d(v) = 0$ for every original vertex. Every unit must now move from s to t. We can then completely cancel b by adding an arc ts with capacities $l(ts) = u(ts) = d(t) = -d(s)$. The new linear program can be simply written:

$$\begin{array}{ll} \min & cx \text{ subject to} \\ & \sum_{e\in\delta^-(v)} x_e - \sum_{e\in\delta^+(v)} = 0 \qquad \text{(for all } v \in V) \\ & l \leq x \leq u \end{array} \tag{5.4}$$

Now, we have a system in which supplies are being moved, and there is no accumulation at any vertex. A feasible solution to this system is called a (constrained) *circulation*.

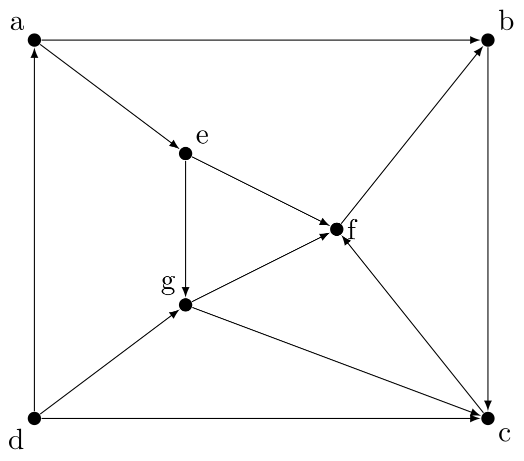
5.2.2 The basis

The linear program (5.3) contains two kind of constraints: bounds, that will be taken care of as in the boxed simplex method, and conservation laws. Let's study in detail the conservation law constraints. We can rewrite them using the following matrix. The *incidence matrix* of a directed graph G is a matrix with $|V(G)|$ rows and $|E(G)|$ columns, indexed respectively by vertices and columns, with coefficients $\{0, 1, -1\}$. That is $M \in \{0, 1, -1\}^{V(G)\times E(G)}$. It is defined by:

$$m_{v,e} = \begin{cases} 1 & \text{if } v \text{ is the head of } e, \\ -1 & \text{if } v \text{ is the tail of } e, \\ 0 & \text{otherwise.} \end{cases}$$

(See Figure 5.3). Then the conservation laws may be rewritten as $Mx = b$.

Each column of M contains exactly one 1 and one -1. So the sum of the rows of M is zero. As M has only $n = |V(G)|$ rows, M has rank at most $n - 1$. Choose a subset M' of columns of M, this corresponds to a subset E' of edges of G. If E' contains a cycle, this subset of columns is obviously not linearly independant. Conversely, consider a minimal subset



$$\begin{pmatrix}
-1 & & & 1 & & & & -1 & & & & \\
1 & -1 & & & & & & & 1 & & & \\
 & 1 & 1 & & & & & & & -1 & 1 & \\
 & & -1 & -1 & & & & & & & & -1 \\
 & & & & -1 & & -1 & 1 & & & & \\
 & & & & 1 & 1 & & & -1 & 1 & & \\
 & & & & & -1 & 1 & & & & -1 & 1
\end{pmatrix}$$

Figure 5.3: *A directed graph and its incidence matrix.*

of linearly dependant columns E'. Then there is a function $w : E' \to \mathbb{R}^*$, such that for every vertex $v \in V(G)$, $\sum_{E' \cap \delta^+(v)} w(e) = \sum_{E' \cap \delta^-(v)} w(e)$. If E' is acyclic, there is a vertex v with degree one in E', say $uv \in E'$. Then the previous equality for v reads for v: $w_e = 0$, contradicting the minimality of E'. Hence, E' is not acyclic, it must be a cycle.

Therefore, independant sets of columns are in one-to-one correspondance with acyclic subgraphs of G. Moreover the rank of M is $n-1$: we have already seen that is it at most $n-1$. Now, if we take a linear combination $y \in \mathbb{R}^V$ of its lines, with $y_v = 0$ for some vertex v, and $yM = 0$, then every vertex u adjacent to a vertex u' with $y_{u'} = 0$ satisfies $y_u = 0$ (by checking the column of uu'). By connectivity, $y = 0$. Hence, maximal independant sets of column, which corresponds to spanning trees, have cardinality $n-1$.

Lemma 5.2.2. *Every spanning tree of a graph $G = (V, E)$ contains exactly $|V| - 1$ edges.*

If we go back to the simplex method, it means that basis for the simplex method correspond to spanning trees of the graph. Recall our linear program 5.3:

$$\begin{array}{ll} \min & cx \text{ subject to} \\ & \sum_{e \in \delta^-(v)} x_e - \sum_{e \in \delta^+(v)} = d(v) \qquad \text{(for all } v \in V) \\ & l \le x \le u \end{array}$$

Let T be a basis, that is a spanning tree of G. It should define a unique basic solution, when the non-basic variables have a fixed value. We check it. Let $E' := E(G) - E(T)$ be the indices of the non-basic variable, and $x : E' \to \mathbb{R}$ a function, in practice x_e is in $\{l_e, u_e\}$ but the argument works for any value for non-basic variables. We want to show that there is a unique solution x^* extending x to the conservation laws. We prove by induction on $|T|$ that if T is an acyclic subgraph of G, and x is defined outside T, then there is a unique extension of x to $E(G)$ checking the conservation laws. If T is not empty, T has a vertex v of degree one by Proposition 5.1.4, and let $e \in E(T)$ be incident to v. Then, by applying the conservation law at vertex v, we have that:

$$x_e^* = \varepsilon(d(v) + \sum_{f \in \delta^+(v) \setminus T} x_f - \sum_{f \in \delta^-(v) \setminus T} x_f)$$

where $\varepsilon = 1$ if e enters v, and $\varepsilon = -1$ if e leaves v. Hence x_e^* is uniquely determined. Then, by induction on $T - e$, we are done. Moreover, this proof is algorithmic. We can prove slightly more (though the corresponding algorithm is less efficient, this proof will turn out to be useful latter). For an arc e in T, $T-e$ consists in two connected component T_1 and T_2 that are both acyclic. We may assume that e leaves T_1 and enters T_2. Then by summing the conservation laws on T_1 (or equivalently on T_2), we get:

$$x_e^* = \sum_{e' \in \delta^-(T_1)} x_{e'} - \sum_{e' \in \delta^+(T_1)-e} x_{e'} - \sum_{v \in V(T_1)} d(v) \tag{5.5}$$

Hence, x^* is uniquely determined.

It is also interesting to describe the dual values associated to a given spanning tree T. Dual values are associated with vertices (because there is one ceonservation law per vertex), and can be interpreted as the cost of delivering one unit of flow to this vertex. We denote p_v the dual variable for vertex v. As M as rank $n-1$, we can determine the dual values up to a constant: adding the same constant to all the dual variable will change neither the feasibility nor the objective value of a dual solution. Hence, we choose an arbitrary vertex r, and fix $p_r = 0$. More precisely, the dual program of (5.3) is given by:

$$\begin{array}{lrl} \max & \sum_{v \in V} p_v d_v + \sum_{e \in E} u_e z_e^+ + l_e z_e^- & \text{subject to} \\ & p_v - p_u + z_e^+ + z_e^- = c_e & \text{(for all } e = uv \in E) \\ & z^- \geq 0 & \\ & z^+ \leq 0 & \end{array} \tag{5.6}$$

Once the values of the p_v's are fixed, the values for z^+ and z^- are determined. Indeed, if $z_{uv}^+ + z_{uv}^- = c_e - p_v + p_u$ is positive, we maximize the objective by taking $z_{uv}^+ = 0$, and if it is negative, we maximize the objective by taking $z_{uv}^- = 0$.

Here is how to interprete these values: z^+ and z^- are additional penalties or bonus (respectively) added to arcs, in such a way that then the price to pay to transport one unit from any vertex u to any vertex v is $p_v - p_u$, whatever directed path we choose. Hence, the minimal cost (and actually the cost of any feasible solution) with the additional penalties is $\sum_{v \in V} p_v d_v$. To get an upper bound on the minimum cost flow, we just consider a flow that get as much penalty, and as few bonus as possible: any flow would be better. This is given by taking the upper capacity for positive penalty, and lower capacity for negative penalty, and it gives immediately the dual objective. Hence a primal solution must use an arc at its upper bound if $p_v - p_u > c_{uv}$, and at its lower bound if $p_v - p_u < c_{uv}$.

The complementary slackness also implies that for an arc uv in the basic tree T, the dual constraint associated to uv is tight, which means that $p_v - p_u = c_{uv}$, and this is true for dual values during the simplex method.

Thus, we can easily compute the dual values associated to a basic tree at any moment, by summing the cost of the arcs in a path between r and any vertex (taking the inverse of the cost for backward arcs).

5.3 Description of the network simplex

5.3.1 Iteration

We start from a basic feasible solution, and want to decide if either it is optimal, or there is a better basic feasible solution. Assume that our base is described by the tree T, the non-basic variables are partitioned into $L \uplus U = E(G) - E(T)$. We are looking for a non-basic variable $e \in E(G) - E(T)$ that can improve the solution. What is the effect of increasing the value of e on the solution?

As T is a maximal acyclic subgraph, $T \cup \{e\}$ has a cycle C containing e as a forward arc. Each edge e' of $T - C$ has still its value uniquely determined by the values of non-basic variables minus e: indeed, $T + e - e'$ has still two components, one of them containing C, the other one determining the value of $x^*_{e'}$ by Equation (5.5). Consequently, only the values of the arcs in C are changed by changing the value of e.

Therefore we want to change the values along C without breaking the conservation laws. Let $\alpha : C \to \mathbb{R}$ be the change made on the cycle: we are going from solution x^* to solution $x^* + \alpha$ ($\alpha_e = 0$ if $e \notin C$). Then, for a vertex v in the cycle C, we must have that:

$$\sum_{e' \in \delta^+(v) \cap C} \alpha_{e'} = \sum_{e' \in \delta^-(v) \cap C} \alpha_{e'}$$

Every vertex of C has degree 2 in C. It follows that there is a constant ε such that

$$\alpha_{e'} = \begin{cases} \varepsilon & \text{if } e' \text{ is forward in } C, \\ -\varepsilon & \text{if } e' \text{ is backward in } C. \end{cases}$$

The sign of ε depends on if e is at its upper bound (negative) or at its lower bound (positive). We want to choose $|\varepsilon|$ as large as possible while respecting all the bounds: this gives one bound on $|\varepsilon|$ for each arc in the cycle C. By choosing the minimal bound, it will give a leaving variable f (possibly many edges can be chosen, and we can even have $f = e$ in some cases). The value of x^* is changed on C by $\pm\varepsilon$ (depending on the direction of the arc, forward or backward). The spanning tree (the basis) goes from T to $(T \setminus \{e\}) \cup \{f\}$.

Let F be the forward arcs of C, and B the backward arcs. Changing the value on the cycle by ε would lead to the following change in the objective:

$$\varepsilon(\sum_{f\in F} c_f - \sum_{b\in B} c_b)$$

This implies that we want to choose an entering arc e if and only if:

- either $x_e = l_e$ and $c_e < \sum_{f\in F-e} c_f - \sum_{b\in B} c_b$,
- or $x_e = u_e$ and $c_e > \sum_{f\in F-e} c_f - \sum_{b\in B} c_b$.

We hence know how to find an entering variable and how to change a basic solution to a new non-basic solution. As a consequence of the general simplex method, if there is no candidate arc for entering the basis, the current solution is optimal.

There is however a simpler way to find an entering arc. We know how to determine the value of the dual variables for any basic tree, by summing the cost of the arcs between r and any other vertex in a path in T. Moreover, p_v represents the cost of delivering one unit of flow to vertex v. If we have an arc st, with $x_{st} < u_{st}$, it means that if we deliver a unit to s (at a cost of p_u) and then move that unit to t through st, it will cost us $p_s + c_{st}$, and this value must be more than p_t or p_t is over-evaluated (it means that the dual solution is not feasible, and st is a potential entering arc). Similarly, if $x_{st} > l_{st}$, $p_s + c_{st}$ should be less that p_t, otherwise we get an entering arc. Thus we only have to find the dual values, and check whether there is an arc outside T violating these rules. If not, the complementary slackness conditions are satisfied and the solution is optimal.

Example: Consider the example of Figure 5.4.

The first problem is to find a basic feasible solution. We will see how to find one latter, here we start from the solution of Figure 5.5. The basic spanning tree is any spanning tree containing all the arcs that are not at one of their bounds (if these arcs induce a cycle, the solution is not basic). Once we found a basic tree we compute the associated potential: we choose one vertex r and define its potential to be zero. Then the potential of any other vertex v is the sum of the cost of the arcs of the unique path between r and v, where backward arcs are counted negatively, forward arcs positively. For instance, the potential of 6 here is $8 - 5 + 4 = 7$, corresponding to the path 14,24,26.

The complementary slackness conditions say that if for each arc at its lower bound, the cost of it is greater than the difference of potential, and

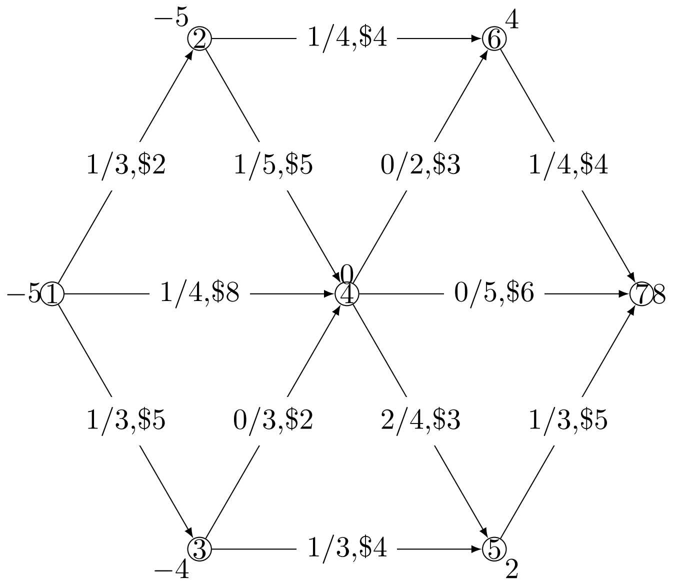


Figure 5.4: *An instance of network flow. Each arc has three values: the lower and upper capacities and the cost of transporting one unit along that arc. The value outside each vertex is its demand (the value inside is just an identifier).*

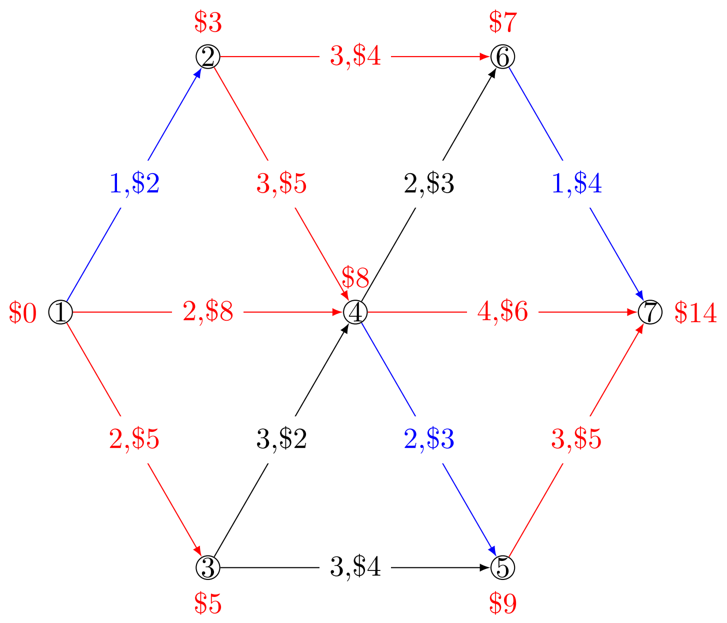


Figure 5.5: *A basic solution, the red tree is the basic spanning tree. Blue arcs are at their lower bound, black arcs at their upper bound. The value of each node (the potential) corresponds to a dual value, obtained by fixing one node to be* \$0*, and obtaining the other values thanks to the basic spanning tree. To improve the solution, we need either a black arc with cost greater than the difference of potential, or a blue edge with difference of potential smaller than the difference of potential. Here, the arc 12 has cost* \$2 *but the difference of potential is* \$3*: we should use more of its capacity.*

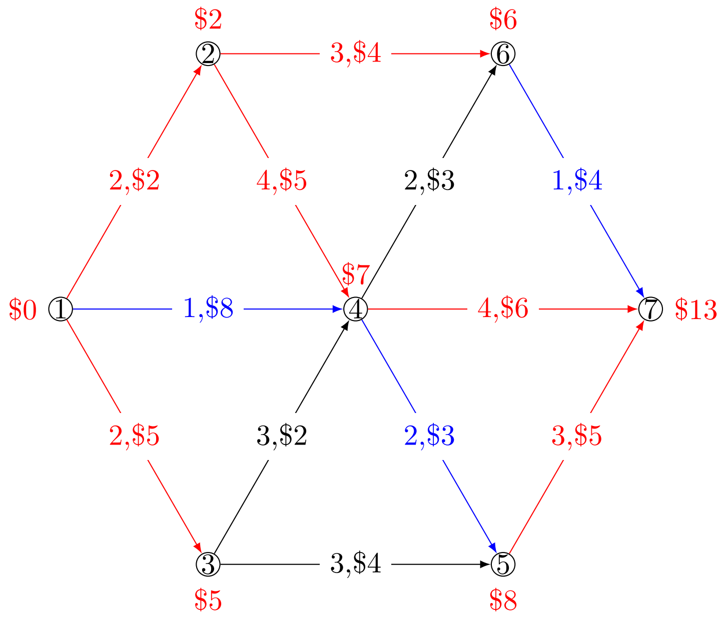


Figure 5.6: *An improved solution, after increasing the cycle 12,24,14. We must compute again the potentials. This is done by considering the arcs of the tree.*

for each arc at its upper bound, the cost of it is smaller than the difference of potential, then the solution is optimal. Hence, it gives an entering arc. In Figure 5.5 the possible entering arcs are 12, 46, and 67.

Consider the arc 12. It is more avantageous to increase the flow on 12,24, and decrease it on 14. Indeed, the former solution costs \$7, the latter \$8. This is exactly what says the condition for being an entering arc: given a spanning tree and an arc, they induces exactly one cycle (here,12,24,14). If we increase the flow on every forward arc of this cycle, and decrease it on every backward arc by the same value, we can easily check that the conservation laws are still satisfied. Hence, depending on the difference of the cost, we want to change the flow in one direction. In our example, we want to increase the flow by 1 in the forward direction (12,24), and decrease it by 1 in the backward direction (14). We cannot do more, because we are limited by the lower bound on the arc 14; 14 is the leaving arc. We thus obtain a new solution, and a new basic tree by removing 14 and adding 12, as depicted in Figure 5.6.

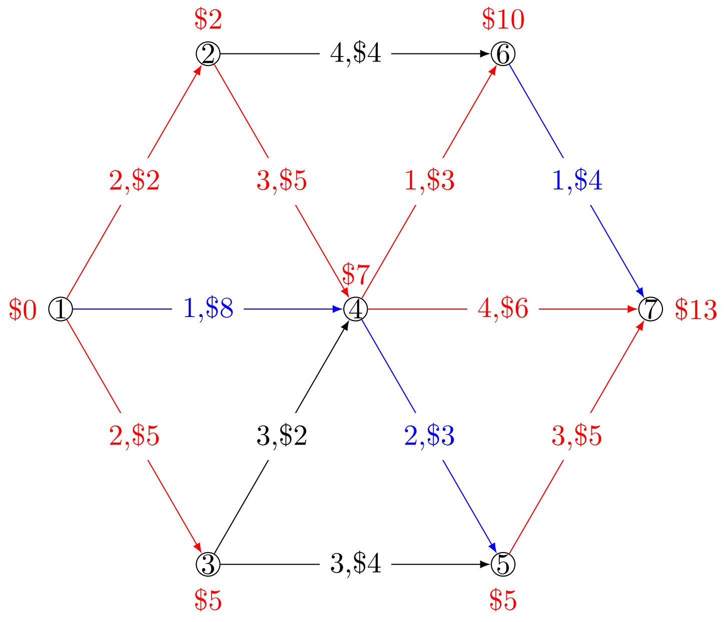


Figure 5.7: *An improved solution, after increasing the cycle 26,46,24.*

Possible entering arcs are 46, 67 and 35. Consider arc 46. This arc is at its upper bound: we are transporting as much as possible. However, the cost is greater than the difference of potential: we are paying more than what the tree would give us. Consider the cycle on the tree plus 46: 24,46,26. The cost of backward arcs is \$4, the cost of forward arc is \$8, hence we want to decrease as much as possible the forward arcs, and increase the flow on the backward arcs. The arc 26 can only accpet one more unit of flow, hence this is the leaving arc, we change the flow by one unit along this cycle. We get the solution of Figure 5.7, after finding the new potential values.

From the potentials, we get that there is only one candidate for entering the basis, the arc 35. The induced cycle is 35,57,74,24,12,13 and we want to decrease the flow on backward arc. We are limited by arcs 13, 12 and 47, we can choose any of them as leaving arc. We choose 13, it gives the solution of Figure 5.7.

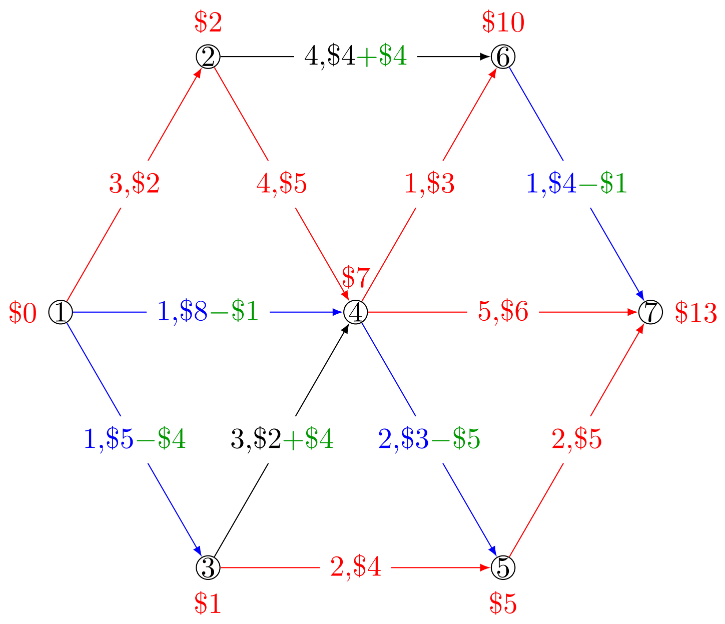


Figure 5.8: *An optimal solution. To see the optimality, consider adding the green values as penalties on the arcs. With these penalties, the cost of transporting one unit from u to v, for any directed path, is exactly* $p_v - p_u$*, hence any possible flow has value* $\sum_v p_v d_v = \$140$*. But our solution maximizes the penalties paid, because each arc with positive value is at its upper bound, and each arc with negative value is taken at its lower bound. As we maximize the penalty, our solution minimizes the cost over the problem without penalty, it is a minimum cost flow.*

5.3.2 Initialization

We still have to settle the question of the initialization of the network simplex method. A first attempt would be to adapt the phase I of the simplex method: we would have to add an artificial variable for every vertex, to measure the excess of flow in each vertex. Then we would have to minimize the sum of those values to get a feasible solution (or prove that none exists). Unfortunately, the problem that we would got would not be a network flow problem, hence we would have to use the revised simplex method to solve it. Then, using the network simplex method would have no interest.

To get a better solution, we must transform this idea into a network flow problem. We want to start from a solution that is easy to find, for instance $x = l$. The conservation laws are usually violated by doing so. Let $\mathrm{exc}_x(v)$ be the excess at vertex v in this solution, defined by:

$$\mathrm{exc}_x(v) = \sum_{e\in\delta^-(v)} x_e - \sum_{e\in\delta^+(v)} x_e - d(v)$$

This is the amount of flow entering the vertex minus the flow leaving the vertex. We have:

$$\begin{aligned}\sum_{v\in V}\mathrm{exc}_x(v) &= \sum_{v\in V}(\sum_{uv\in E} x_{uv} - \sum_{vu\in E} x_{vu}) - \sum_{v\in V} d(v)\\ &= \sum_{uv\in E}(x_{uv} - x_{uv}) - \sum_{v\in V} d(v)\\ &= 0\end{aligned}$$

We add a new vertex, call it s. For each vertex v with positive excess we add an arc vs with $l(sv) = 0$, $u(vs) = \mathrm{exc}_x(v)$. For each vertex v with negative excess we add an arc sv with $l(vs) = 0$, $u(sv) = -\mathrm{exc}_x(v)$. Then x extends obviously to the new graph and checks the conservation laws.

We then have two possibilities. Solve the new problem with a cost 1 one the arc incident to s and 0 elsewhere. Then a flow of value 0 is a feasible flow for the original problem. We can then use the algorithm below to find the minimum cost flow. This is an analog of the two-phase method.

Another way is to solve everything in only one call to the network simplex. For this, we add a prohibitive cost on the new arcs. Such a cost could be the sum of the cost of the original graph: it is easy to see (by using flow decomposition) that a solution that does not use the new arcs is strictly better than one using even only one unit of capacity in the new arcs (to see this, we need to remark that we the capacities are integral, there is always an integral optimum solution, we will come back to this latter). This method is similar to the big-M method for regular linear program.

In either way, what happens if we do not find a feasible solution that uses only the original arcs? Then, for an optimal solution x to the auxiliary problem, there are basic arcs in both $\delta^+(s)$ and $\delta^-(s)$. Consider the component C' of $T - \delta^+(s)$containing s, and let $C = C' - s$. We can define a dual potential y with $y_s = 0$, then $y_v = -1$ if $v \in C$, and $y_v = 1$ otherwise. Because the solution is optimal, there can be no arc e in G from C to $V - C$ with $x_e \neq u_e$, or from $V - C$ to C with $x_e \neq l_e$. Hence we have:

$$\sum_{v \in U} d(v) + \sum_{e \in \delta^+(U)} u_e - \sum_{e \in \delta^-(U)} l_e = x_{us} > 0$$

It is clear that the subset U is an obstacle to the feasibility of the network flow problem: there is not enough capacity to route outside all the flow entering U. We have just proved the following theorem:

Theorem 5.3.1. *(Hoffman's condition)*
(G, l, u, c) has a solution if and only if there is no set $U \subset V$ with:

$$\sum_{v \in U} d(v) + \sum_{e \in \delta^+(U)} u_e - \sum_{e \in \delta^-(U)} l_e > 0$$

5.3.3 Cycling and termination

As in the simplex method, the network simplex method may cycle. Also as in the simplex method, there are rules to avoid cycling, for example Cunningham's rule. However, those rules usually slow down the execution of the network simplex, hence they are not used in practice.

A difference with the simplex method is that we know pivoting rules for the simplex method that leads to polynomial algorithm (even strongly polynomial algorithm: it means that the complexity depends only on the size of the graph, and not on the numerical values like the capacities and the costs). But again, these rules are only interesting theoretically, as more simple rules does better in practice.

5.3.4 Implementation of the network simplex method

The practical complexity of the network simplex relies on the three following points:

- Finding the entering arc.

- Finding the leaving arc.
- Updating the basic tree and the potential.

The two last points are usually solved by adopting a good data structure for the basic tree. The best known structure is the dynamic trees (Sleator, Tarjan), that allows to perform them in $O(\log n)$ elementary operation in the worst-case. One of the trick here is to remark that the potentials need not to be completely recomputed. It is quite easy to see that less than half of the potentials are changed during an iteration, as we need to change only one of the two parts created when removing the leaving edge from the basic tree. However, this is not enough to get a logarithmic complexity. To this end, one need more involved ideas, relying on computation of the nearest common ancestor of the vertices in the basic tree, to find the difference of potential (what we really need). With these remarks, every iteration can be performed really fast.

The first point, finding the entering arc, is the most important to get a efficient algorithm. We do not want to try every possible arc, as the number of arcs can be quite large ($O(n^2)$ in the worst case). We need to apply *partial pricing* technics, that is an evaluation of only a small part of the non-basic variables to find the entering variable (even in the more general context of the simplex method). This is by opposition to *total pricing* where every non-basic variable is considered when choosing the entering variable. Obviously, we still need to check every variable during the last iteration to check that our solution is optimal, but once we found an entering variable, we can just improve our basic solution immediately. We would gain a lot on the computation of entering variable in this way. However, the quality of the entering variable would be very low (increasing the number of iteration). Hence in partial pricing, we want to find several candidates, but without considering every non-basic variable, and choose among these candidates the most promising. This is a compromise between the time spent on looking for entering variables, and the quality of the entering variable. A good network simplex algorithm is one that solve this compromise in the best way.

5.4 Applications and consequences

One very important important consequence of our work on the network simplex is this theorem:

Theorem 5.4.1 (Integrality of the flow polyhedron)**.** *For a graph G with capacities l, u and demands d integral, and for any cost function c (not* *necessarily integral), if there is a minimum cost flow, there is an integral minimum cost flow.*

Proof. We can prove it by analyzing an iteration of the network method and see that basic solutions stay integral. We can also prove it directly: any basic solution is integral. Indeed, a basic solution is described by a partition (T, L, U) of its arc, where T is a spanning tree, L, U are the arcs at their lower and upper bounds respectively. The flow on the arcs of L and U is clearly integral. But as we have already seen, the flow on an arc e of T is given by (5.5), a sum of integral values, and hence is also integral. $\square$

Remark. The name of the theorem can be justified in the following way. The theorem says that any extreme point of the polyhedron $\{x \mid Mx = d, l \leq x \leq u\}$ is integral. This is an extremely strong property, as it means that finding integral optimum solutions is computationaly just the same as finding fractional solutions. However, this is false in many problems, as finding integral solutions to a linear program is usually a very difficult problem.

Network simplex has many applications. The most natural ones are obviously transportation problem. First, we give some famous special cases of the minimum cost flow problem.

5.4.1 Maximum flow

Suppose that we have a graph G, with a (upper) capacity function u, and we are given two vertices s and t. We want to transport as many units as possible from s to t. This defines the *maximum flow problem*, which can be written as a linear program, using the incidence matrix M of G:

$$\begin{array}{lll} \max & d(t) & \text{subject to} \\ & Mx = d & \\ & 0 \leq x \leq u & \\ & d(v) = 0 & (\text{for all } v \in V - \{s,t\}) \end{array} \tag{5.7}$$

There is a simple way to reduce this problem to a minimum cost flow problem. Add an additional arc ts with cost -1, $l(ts) = 0$ and $u(ts) = +\infty$. Then impose $d(v) = 0$ on every vertex, and $c(e) = 0$ for every arc except ts. Then it is quite easy to see that a solution x to the minimum cost flow problem defined by these values implies a maximum flow in the original graph with the same value. Indeed, remove the arc ts, and you get a feasible solution to the maximum flow problem with value $x(ts)$. Conversely, a solution to the maximum flow problem implies a solution to the minimum cost flow problem, thus the two problems are equivalent.

Then by applying Hoffman's condition, we get the *Max Flow – Min Cut* theorem (MFMC):

Theorem 5.4.2 (Ford, Fulkerson 1956)**.** *The maximum flow between s and t is equal to the minimum capacity of a cut between s and t.*

Here, a *cut* between s and t is a set of vertices S containing s but not t. The capacity of the cut S is $\sum_{e\in\delta^+(S)} u(e)$. It is obvious, by summing the conservation law on S, that the capacity of a cut is an upper bound on the value of the flow. Moreover, this is a generalization of the Lemma 5.1.2 on the existence of a directed path, as a directed path can be seen as a flow between two vertices (with value the minimum upper capacity of the arcs in the path). It also implies that finding a minimum cut between two vertices can be solved by the network simplex.

There are many algorithms to solve the maximum flow problem, but all can be explained in terms of linear programming. The main idea is that at the end we want:

- (i) a primal feasible solution x,
- (ii) a dual feasible solution y,
- (iii) complementary slackness for x and y.

It is quite easy to get two of these conditions simultaneously. Any algorithm for solving the maximum flow problem (and actually many different combinatorial problems) start with two conditions satisfied, and modify the values of x and y while keeping these two conditions, until the third one is also satisfied. The simplex method for instance keeps condition (i) and (iii) satisfied, while the dual simplex method keeps condition (ii) and (iii). It is also possible to design an algorithm keeping (i) and (ii), this would be called a *primal-dual* algorithm.

5.4.2 The assignment problem

An artist has 5 paintings to sell. It happens that there are 5 potential costumers, each of them wanting to buy exactly one painting. But two different costumers does not give the same value to the same painting. More precisely, what they offer to pay is given by the following matrix, each row is a painting, each column is a costumer:

$$\begin{pmatrix} 1200 & 800 & 1000 & 0 & 0 \\ 1100 & 900 & 1200 & 1400 & 0 \\ 1000 & 0 & 1100 & 1300 & 900 \\ 0 & 800 & 0 & 1200 & 700 \\ 0 & 700 & 0 & 0 & 500 \end{pmatrix}$$

How should the artist sell his paintings? This is an instance of the assignment problem. More generally, we are given a matrix, and we want to pick some coefficients of the matrix, but exactly one per row and one per column. The goal is to maximize the sum of the chosen coefficients.

This is not a linear problem, as we only want an integral solution (by no way we could cut a painting in pieces whose values sum to the original value of the painting). Anyway, we first look for a fractional solution (because we are in the network simplex chapter, we can hope that this is just a special case of the minimum cost flow, and hence there would be an integral optimal solution). We say that the following problem is a fractional relaxation of the assignment problem. Each coefficient of the matrix has a non-negative variable. The idea is that we choose that coefficient if the corresponding variable has value 1, and we do not choose it if it has value 0.

$$\begin{array}{lll} \max & \sum_{i,j} m_{i,j}x_{i,j} & \text{subject to} \\ & \sum_j x_{i,j} \leq 1 & \text{(for all row } i\text{)} \\ & \sum_i x_{i,j} \leq 1 & \text{(for all column } j\text{)} \\ & x \geq 0 & \end{array} \tag{5.8}$$

Note that the constraints imply that every variable has value at most one, and if the coefficient of the matrix are non-negative, an integral solution induces a solution to the assignment problem. The dual of this linear program is:

$$\begin{array}{lll} \min & \sum_i y_i + \sum_j z_j & \text{subject to} \\ & y_i + z_j \geq m_{i,j} & \text{(for all } i, j\text{)} \\ & y, z \geq 0 & \end{array} \tag{5.9}$$

This can be seen has placing a value on each painting and each costumer, such that the price associated by a costumer to a painting is less than the sum of the value placed on this costumer and this painting. From this, it is clear that a solution to the dual problem gives an upper bound for the assignment problem: an assignment will never give us more than the value placed on paintings and costumers.

The assignment problem actually turns out to be a special case of the minimum cost flow problem. To see this, we build a graph whose vertices are the costumers and the paintings. Add an arc e from a painting i to a costumer j, $c(e) = -m_{i,j}$ (do not add an arc of $m_{i,j} = 0$). Moreover, add two additional vertices s and t. There is an arc with cost 0 from s to every painting, and from every costumer to t. All the arcs have capacities $l = 0$, $u = 1$. Finally, we have $-d(s) = d(t) = k$ where k is the number of rows (or columns). See Figure 5.9.

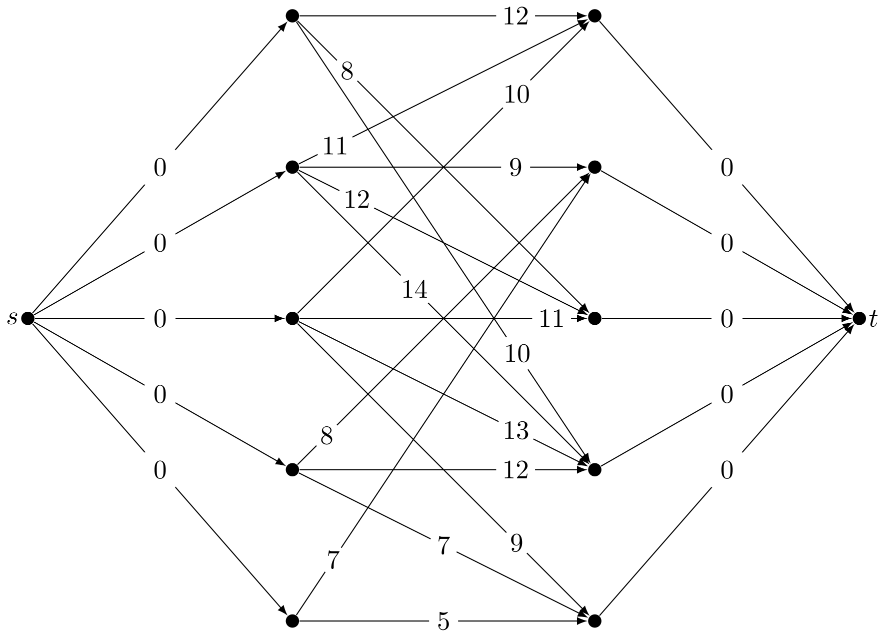


Figure 5.9: *Reduction of the assignment problem to a minimum cost flow problem.*

If we want to satify the demand, we must take every arc incident to s and t at their upper capacities. Thus, an integral extreme point would gives us exactly one arc leaving each painting, and one arc entering each painting. The are not incident to s and t gives us an assignment.

The assignment problem can be solved more directly by using a dual algorithm (we keep condition (*ii*) and (*iii*)), the so-called *Hungarian method.* We need a feasible dual solution, for a start we can assign a value of 0 to each costumer, and the maximal value of its row to each painting. In our example, this gives $y = (12, 14, 13, 12, 7)$, $z = (0, 0, 0, 0, 0)$. We also want a vector x, possibly violating the primal constraint, but validation the slackness condition. Here, the slackness condition says that

- if some value $x_{i,j}$ is not zero, then $y_i + z_j = m_{i,j}$,
- if $y_i \neq 0$, then $\sum_j x_{i,j} = 1$,
- if $z_j \neq 0$, then $\sum_i x_{i,j} = 1$.

As an initial condition, we can choose one maximal element in each row and set its x-value to 1, all other primal variable having value zero. Hence, the complementary slackness is satisfied. We will even keep $\sum_j x_{i,j} = 1$, hence the only violated constraints are $\sum_i x_{i,j} = 1$ for some columns j.

The computation goes as follow. If x is primal-feasible, then x and y are optimal by complementary slackness. Else, as each row has an x-value of 1, some column j has an x-value strictly greater than 1. The idea is then to increase z_j, and decrease all the y_i such that $x_{i,j} = 1$ by the same amount (thus keeping complementary slackness). We must be careful, as we want ot keep dual feasibility and complementary slackness. More exactly, we will have two kinds of improvement:

- improvement of the primal values, without changing the dual values. For some row i, we change the costumer to which is assigned the painting i. Actually, we may have to change simultaneously the assignment for several paintings. But every costumer who had at least one painting must still have one painting, and if a painting i is assigned to a costumer j, we must have $y_i = z_j = m_{i,j}$.
- improvement of the dual values, we decrease the values on a subset of k paintings, and increase them on a subset of k' costumers, with $k' < k$. In that case, we must enforce these rules:

  (a) If we decrease y_i and there is j with $m_{i,j} = y_i + z_j$, we must increase z_j by the same quantity to keep the dual feasibility.

  (b) If $x_{i,j} = 1$ and we increase z_j, we must decrease y_i by the same quantity to keep the complementary slackness.

The rules on the improvement of the dual values suggest the following algorithm. Starting with a column j with $\sum_i x_{i,j} > 1$, we compute a smallest possible subset of rows and columns checking those rules (a) and (b). We start with column j only, and as long as one of the rules is violated, we add the violating row or column. At the end, we reach a subset I of rows, and a subset J of column. We then distinguish between two cases:

- there is a column $j' \in J$ with $\sum_i x_{i,j} = 0$. In that case we can improve the primal vector x, by removing one painting from j and adding one to j'. To find the new primal vector, we use the following procedure. Because j' is in J, there must be a i with $m_{i,j'} = y_i + z_{j'}$, but still the painting i is not allocated to j', but some other costumer j'', $x_{i,j''} = 1$. We define $x'_{i,j'} = 1$ and $x'_{i,j''} = 0$. Now, if $j = j''$ (or if $\sum_i x_{i,j''} > 1$) we are done. Otherwise the fact that j'' is in J implies that there is some painting $i' \in I$. Then we iterate this procedure from i', j''. At the end, we get a new primal vector x'.
- every column of J is attributed one painting at least. then $|J| < |I|$ (because j has at least two paintings), we decrease $y_j, j \in J$, and $x_i, i \in I$, as long as we do not violate the dual feasibility on some coefficient of the matrix. In that case, the value of the dual solution is improved.

Example: We apply the Hungarian method to our problem. Recall the matrix (dividing every value by 100 for the convenience):

$$M = \begin{pmatrix} 12 & 8 & 10 & 0 & 0 \\ 11 & 9 & 12 & 14 & 0 \\ 10 & 0 & 11 & 13 & 9 \\ 0 & 8 & 0 & 12 & 7 \\ 0 & 7 & 0 & 0 & 5 \end{pmatrix}$$

We start with the dual values $y = (12, 14, 13, 12, 7)$ (the maximum of each line), and $z = (0, 0, 0, 0, 0)$. For each line we must choose one coefficient $m_{i,j} = y_i + z_j$. We mark those coefficients with $m_{i,j} = x_i + y_j$ by a circle, and a grey circle if $x_{i,j} = 1$. Hence the first solution may be written as:

$$\begin{array}{c|ccccc} & 0 & 0 & 0 & 0 & 0 \\ \hline 12 & \textcircled{12} & 8 & 10 & 0 & 0 \\ 14 & 11 & 9 & 12 & \textcircled{14} & 0 \\ 13 & 10 & 0 & 11 & \textcircled{13} & 9 \\ 12 & 0 & 8 & 0 & \textcircled{12} & 7 \\ 7 & 0 & \textcircled{7} & 0 & 0 & 5 \end{array}$$

Column 4 has too many rows assigned to it. Hence we want to increase the dual value on that column, and decrease it on rows 2, 3 and 4. We can change these values by at most two units, at which point some other coefficient will become tight ($m_{i,j} = y_i + z_j$).

$$\begin{array}{c|ccccc}
 & 0 & 0 & 0 & 2 & 0 \\ \hline
12 & \fcolorbox{black}{lightgray}{12} & 8 & 10 & 0 & 0 \\
12 & 11 & 9 & \textcircled{12} & \fcolorbox{black}{lightgray}{14} & 0 \\
11 & 10 & 0 & \textcircled{11} & \fcolorbox{black}{lightgray}{13} & 9 \\
10 & 0 & 8 & 0 & \fcolorbox{black}{lightgray}{12} & 7 \\
7 & 0 & \fcolorbox{black}{lightgray}{7} & 0 & 0 & 5
\end{array}$$

We did not change the primal vector, hence column 4 has still too many rows assigned to it. But this time, if we want to increase z_4 by α, we must decrease y_2, y_3, y_4 by α, and hence increase z_3 by α. But column 3 has no painting assigned to it, this means we can move some painting, for example painting 3, to get a new primal vector:

$$\begin{array}{c|ccccc}
 & 0 & 0 & 0 & 2 & 0 \\ \hline
12 & \fcolorbox{black}{lightgray}{12} & 8 & 10 & 0 & 0 \\
12 & 11 & 9 & \textcircled{12} & \fcolorbox{black}{lightgray}{14} & 0 \\
11 & 10 & 0 & \fcolorbox{black}{lightgray}{11} & \textcircled{13} & 9 \\
10 & 0 & 8 & 0 & \fcolorbox{black}{lightgray}{12} & 7 \\
7 & 0 & \fcolorbox{black}{lightgray}{7} & 0 & 0 & 5
\end{array}$$

Then, if we want to decrease the dual value of column 4, we must increase the dual values rows 2 and 4. Then, because $m_{2,3}$ is tight we need to increase column 3 also. But then, to keep complementary slackness on $x_{3,3}$, we must decrease y_3. We get $I = \{2, 3, 4\}$ and $J = \{3, 4\}$, and hence, and we can move the dual values by one unit, this gives:

$$\begin{array}{c|ccccc}
 & 0 & 0 & 1 & 3 & 0 \\
\hline
12 & \colorbox{lightgray}{$\textcircled{12}$} & 8 & 10 & 0 & 0 \\
11 & \textcircled{11} & 9 & \textcircled{12} & \colorbox{lightgray}{$\textcircled{14}$} & 0 \\
10 & \textcircled{10} & 0 & \colorbox{lightgray}{$\textcircled{11}$} & \textcircled{13} & 9 \\
9 & 0 & 8 & 0 & \colorbox{lightgray}{$\textcircled{12}$} & 7 \\
7 & 0 & \colorbox{lightgray}{$\textcircled{7}$} & 0 & 0 & 5
\end{array}$$

We still have too much on column 4, we want to increase it again. By a similar reasoning, we get that we must increase on $J = \{1, 3, 4\}$, and decrease on $\{1, 2, 3, 4\}$. We are limited by $m_{4,2}$ and $m_{3,5}$, so we get:

$$\begin{array}{c|ccccc}
 & 1 & 0 & 2 & 4 & 0 \\
\hline
11 & \colorbox{lightgray}{$\textcircled{12}$} & 8 & 10 & 0 & 0 \\
10 & \textcircled{11} & 9 & \textcircled{12} & \colorbox{lightgray}{$\textcircled{14}$} & 0 \\
9 & \textcircled{10} & 0 & \colorbox{lightgray}{$\textcircled{11}$} & \textcircled{13} & \textcircled{9} \\
8 & 0 & \textcircled{8} & 0 & \colorbox{lightgray}{$\textcircled{12}$} & 7 \\
7 & 0 & \colorbox{lightgray}{$\textcircled{7}$} & 0 & 0 & 5
\end{array}$$

Then, increasing z_4 would force us to decrease y_2 (and y_4), which in turns force us to increase z_3 (and z_1 and z_2 as well), which force us to decrease y_3, which force us to increase z_5. But column 5 has no assignment yet, this means we can improve the primal solution, by moving the assignment following our reasoning. Painting 3 goes from 3 to 5, painting 2 goes from 4 to 3, and we are done. The primal vector is now feasible, our solution is optimal:

$$\begin{array}{c|ccccc}
 & 1 & 0 & 2 & 4 & 0 \\
\hline
11 & \colorbox{lightgray}{$\textcircled{12}$} & 8 & 10 & 0 & 0 \\
10 & \textcircled{11} & 9 & \colorbox{lightgray}{$\textcircled{12}$} & \textcircled{14} & 0 \\
9 & \textcircled{10} & 0 & \textcircled{11} & \textcircled{13} & \colorbox{lightgray}{$\textcircled{9}$} \\
8 & 0 & \textcircled{8} & 0 & \colorbox{lightgray}{$\textcircled{12}$} & 7 \\
7 & 0 & \colorbox{lightgray}{$\textcircled{7}$} & 0 & 0 & 5
\end{array}$$

It is quite easy top prove with the dual values that this solution is indeed optimal. The value of any coefficient is at most the same of the corresponding dual values. As we must pick exactly one coefficient in each row, and one in each column, we cannot gain more than the sum of the dual values. But this is exactly what we get with this solution.

This example illustrates how linear programming can be used to solve integral problems efficiently in some cases. Even if linear programming deals with fractional solutions, some problems like network flows have a special structure that makes easy to find integral solution. Moreover, the methods of linear programming can be applied to some problems outside the initial scope of linear programming.

We conclude this chapter by stating an immediate consequence of the Hungarian method. A *bipartite graph* is a graph whose vertices can be partitioned in two parts, such that every edge has one extremity in each part. A *matching* in a graph is a set of edges such that no two of them have a common extremity (equivalently, each vertex is incident to at most one edge of the matching). Finding an assignment of the paintings to the costumers is just the same as finding a matching in the graph whose vertices are the costumers (A) and the paintings (B), and there is an edge between a painting and a costumer if this costumer wants to buy this painting.

Theorem 5.4.3 (Kőnig, Egerváry 1931). *Let G be a bipartite graph and $w : E(G) \to \mathbb{R}^+$. The maximum weight $\sum_{e \in M} w(e)$ of a matching M in G is equal to the minimum cover $y \in \mathbb{R}_+^V$ of G, that is the minimum of $\sum_{v \in V(G)} y_v$ subject to $y_u + y_v \geq w(uv)$ for all edge $uv \in E(G)$. Moreover, if w is integral, y can be chosen integral.*

Note that this is false for non-bipartite graph, for example take the triangle with weight 1 on every edge, the maximum matching has value 1, but a minimum cover has value $\frac{3}{2}$.