Optimization

Karen Aardal, Leo van Iersel and Remie Janssen

September 6, 2022

Contents

Part I

Modeling

Chapter 1

Modeling Linear and Integer Linear Optimization Problems

In optimization, developing and analyzing models are key activities. Designing a model is a skill and requires a lot of practise before one can "see" what a natural model is. There are, however, some basic techniques one can learn in order to make the process easier. It is very important to be careful in all the steps, and to use good notation.

In this part we will go through some techniques and give some general hints with the aim of making it more easy and fun to design good models. We focus on linear models. The part consists of two chapters. This first chapter concerns linear and linear integer models and Chapter 2 treats mixed-integer 0-1 models, where a subset of the variables are restricted to take the values 0 or 1. Later, in Part II, Chapter 3 we give some theoretical foundations behind the mathematical objects that result from modeling, and in Part IV, Chapter 12 we give examples of how to compare the quality of different models.

1.1 Introduction

In order to design an optimization model, our starting point is input, which is given, and a description of a "goal", such as "maximize the profit", "minimize the cost", "maximize the energy output", and so on. We also have a description of restrictions that we need to take into account, as otherwise the output value would be infinitely large or small. We start with a description of a simple so-called product-mix problem [11].

Example 1.1. Giapetto's Woodcarving Inc., manufactures two types of wooden toys: soldiers and trains. A soldier sells for \$27 and uses \$10 worth of raw materials. Each soldier that is manufactured increases Giapetto's variable labor and overhead costs by \$14. A train sells for \$21 and uses \$9 worth of raw materials. Each train built increases Giapetto's variable labor and overhead costs by \$10. The manufacture of wooden soldiers and trains require two types of skilled labor: carpentry and finishing. A soldier requires 2 hours of finishing and 1 hour of carpentry labor while a train requires 1 hour of finishing and 1 hour of carpentry labor. Each week Giapetto can obtain all the needed raw material but only 100 finishing hours and 80 carpentry hours. Demand for trains is unlimited, but at most 40 soldiers are bought each week. Giapetto wants to maximize the weekly profit (revenues − costs). Formulate a linear optimization model of Giapetto's situation that can be used to maximize his weekly profit. □

When designing a model, the first, very important, step is to separate input from the decision variables. The input represents what is known, whereas the variables represent the unknown. The variables are assigned values as a result of applying an algorithm to the model. In order to increase readability and clarity it is important to use good notation. For very uncomplicated models it can be acceptable to use letters such as x to represent variables, but the more complicated a model gets, the more useful it will be to give names that provide a stronger association to the meaning of the variables. The same is true for the input.

We will formulate this problem quite generally and start by defining the input. We will use the index j to represent the products; $j = 1$ represents soldiers and $j = 2$ represents trains. Once we have looked at the general formulation, we will write down the model for the specific input of Giapetto's.

Input. Let

$$\begin{aligned}
r_j &= \text{the revenue per sold item of product } j,\ j = 1,2,\\
m_j &= \text{the per unit costs of raw material for producing}\\
&\ \text{product } j,\ j = 1,2,\\
l_j &= \text{the per unit cost of labor and overhead for producing}\\
&\ \text{product } j,\ j = 1,2,\\
f_j &= \text{the number of hours of finishing for one unit of}\\
&\ \text{product } j,\ j = 1,2,\\
c_j &= \text{the number of hours of carpentry for one unit of}\\
&\ \text{product } j,\ j = 1,2,\\
FinCap &= \text{the total number of finishing hours available per week,}\\
CarpCap &= \text{the total number of carpentry hours available per week,}\\
MaxDem_j &= \text{maximum demand per week for product } j,\ j = 1,2.
\end{aligned}$$

Next, we define the decision variables. The choice of decision variables for such a simple model is straightforward, but for more complicated models it could be quite complicated and clearly not unique. In order to come up with a good variable definition it is useful to think in terms of what kind of answer we would like to get from the model. In this case we would like to know how many soldiers and how many trains we would like to produce per week and this is therefore the very natural choice.

Variable definition. Let

x_j = the number of product j produced per week, $j = 1,2$.

Since we can only produce an integer number of trains and soldiers we will restrict the variables to take integer values. We are now ready to write down our optimization model:

$$\begin{aligned}
\text{maximize } z(\boldsymbol{x}) = (r_1 - m_1 - l_1)x_1 &\ +\ (r_2 - m_2 - l_2)x_2\\
\text{subject to } f_1x_1 + f_2x_2 &\ \leq\ FinCap\\
c_1x_1 + c_2x_2 &\ \leq\ CarpCap\\
x_j &\ \leq\ MaxDem_j, \quad j = 1,2\\
x_j &\ \geq\ 0, \quad j = 1,2\\
x_j &\ \in\ \mathbb{Z}, \quad j = 1,2\,.
\end{aligned}$$

We will now go through all the elements of the model. The first expression is what we call the *objective function*:

$$\text{maximize } z(\boldsymbol{x}) = (r_1 - m_1 - l_1)x_1 + (r_2 - m_2 - l_2)x_2.$$

We want to maximize the weekly profit for Giapetto's. The coefficient for variable x_1, which is the number of produced soldiers per week, is $(r_1 - m_1 - l_1)$, which is the revenue $-$ the material cost

− the labor cost per unit. Similarly for the production of trains. Instead of writing "maximize" (or "minimize") we typically write the abbreviations "max" (or "min"). Also, instead of writing $z(\boldsymbol{x})$ as the notation for the objective function, we typically just use z.

Next, we have three *constraints* that limit the possible values x_1 and x_2 can take. Sometimes we distinguish between the *structural constraints*, which in this case are the constraints

$$\begin{aligned} f_1x_1 + f_2x_2 &\leq FinCap \\ c_1x_1 + c_2x_2 &\leq CarpCap \\ x_j &\leq MaxDem_j, \quad j = 1,2, \end{aligned}$$

the *nonnegativity constraints* $x_j \geq 0,\ j = 1,2$, and the *integrality constraints* $x_j \in \mathbb{Z},\ j = 1,2$, but this distinction is not really essential. The first structural constraint models the limitation we have in the number of available finishing hours, and the second constraint models the limitation we have in the number of available carpentry hours. The last set of structural constraints sets a limit on the number of sold items. Instead of writing out "subject to" we use the abbreviation "s.t."

We can write this model even more formally, such that it is valid for n products. In this more general case, our product mix model looks as follows:

$$\begin{array}{lrcl} \max & z = \sum_{j=1}^n (r_j - m_j - l_j)x_j & & \\ \text{s.t.} & \sum_{j=1}^n f_jx_j & \leq & FinCap \\ & \sum_{j=1}^n c_jx_j & \leq & CarpCap \\ & x_j & \leq & MaxDem_j, \quad j = 1,\dots,n \\ & x_j & \geq & 0, \quad j = 1,\dots,n. \\ & x_j & \in & \mathbb{Z}, \quad j = 1,\dots,n\,. \end{array}$$

You may think that it is a bit overkill to write this model so formally since we only have two variables here, but in general it is actually quite useful to have such a formal description of the model since we then clarify that the model in fact can be used for any number n of products.

Finally, let us actually write down the model for the specific case of Giapetto's with the relevant values of the input. Notice that $(r_1 - m_1 - l_1) = 27 - 10 - 14 = 3$, $(r_2 - m_2 - l_2) = 21 - 9 - 10 = 2$, and $MaxDem_2 = \infty$, which yields:

$$\begin{array}{lrcrcrl} \max & z = 3x_1 & + & 2x_2 & & & \\ \text{s.t.} & 2x_1 & + & x_2 & \leq & 100 & \\ & x_1 & + & x_2 & \leq & 80 & \\ & x_1 & & & \leq & 40 & \\ & x_1, & & x_2 & \geq & 0, & j = 1,2 \\ & x_1, & & x_2 & \in & \mathbb{Z}, & j = 1,2\,. \end{array}$$

It is left as an exercise for the reader to solve this problem.

1.2 Using variables with multiple indices

The product-mix problem we saw in the previous subsection was quite straightforward to model. Let us look at a slightly more challenging problem [11].

Example 1.2. Powerco has three power plants that supply the need of four cities. Each power plant can supply the following numbers of kilowatt hours (kWh) of electricity: plant 1: 40 million; plant 2: 50 million; plant 3: 40 million. The peak power demands in these cities, which occur at the same time (2 pm), are as follows (in kWh): city 1: 45 million; city 2: 20 million; city 3: 30 million; city 4: 30 million. The costs of sending 1 million kWh of electricity from plant to city depend on the distance the electricity must travel. These costs, and the other input as well, are given in Table 1.1.

Formulate the problem of minimizing the total cost while meeting each city's peak power demand as a linear optimization problem.

From	City 1	City 2	City 3	City 4	Supply million kWh
Plant 1	\$ 8	\$ 6	\$ 10	\$ 9	40
Plant 2	\$ 9	\$ 12	\$ 13	\$ 7	50
Plant 3	\$ 14	\$ 9	\$ 16	\$ 5	40
Demand milion kWh	45	20	30	30	

Table 1.1: Input for the Powerco problem

□

Again, we start by introducing the notation for the input and the decision variables. In our previous example we used one index for each product. Here we could introduce one index j for each combination of plant and city obtaining $j = 1, \ldots, 12$. This, however, makes the model non-transparent to read, and when we interpret the outcome we have to translate the index back to the particular combination. This might be acceptable for 12 combinations, but imagine that we have 50 plants and 500 cities. What we typically do in a case like this is to introduce a double-index notation ij, where i represents a plant, and j a city.

Input. Let

c_{ij}	$=$	the cost of transporting one million kWh from plant i to city j, $i = 1, \ldots, 3$, $j = 1, \ldots, 4$
s_i	$=$	supply, in million kWh, at plant i, $i = 1, \ldots, 3$
d_j	$=$	peak demand, in million kWh, in city j, $j = 1, \ldots, 4$.

Variable definition. Let

x_{ij} $=$ the amount of electricity, in million kWh, transported from plant i to city j, $i = 1, \ldots, 3$, $j = 1, \ldots, 4$.

Before formulating the problem, we will illustrate the problem by a graph, see Figure 1.1. This often helps in the formulation, not only in this case, but in most cases where some underlying graph structure is present. Assume that we have m plants and n cities.

We see from this network that the solid arrows represent input and the dashed lines represent what we would like to know: how much should be transported between the plants and the cities? This precisely corresponds to our choice of decision variables x_{ij}.

We now first formulate the problem in its general form, and then with the specific input for this instance:

$$\begin{array}{lrcl}
\min & z = \sum_{i=1}^{m} \sum_{j=n}^{n} c_{ij} x_{ij} & & \\
\text{s.t.} & \sum_{j=1}^{n} x_{ij} & \leq & s_i, \quad i = 1, \ldots, m \\
& \sum_{i=1}^{m} x_{ij} & = & d_j, \quad j = 1, \ldots, n \\
& x_{ij} & \geq & 0, \quad i = 1, \ldots, m, \; j = 1, \ldots, n.
\end{array}$$

Notice that for the demand constraints $\sum_{i=1}^{m} x_{ij} = d_j$, $j = 1, \ldots, n$, we could also have chosen to use the $\geq$-sign since all costs are positive, but for clarity we choose equality here. Next, we give the formulation for the specific instance:

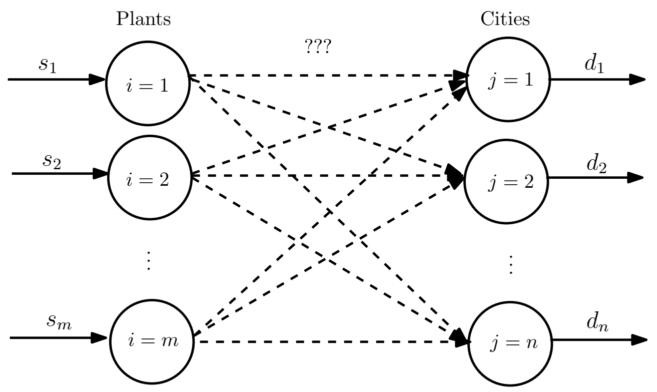


Figure 1.1: Network structure of Example 1.2.

$$\min z = 8x_{11} + 6x_{12} + 10x_{13} + 9x_{14} + 9x_{21} + 12x_{22} + 13x_{23} + 7x_{24} + 14x_{31} + 9x_{32} + 16x_{33} + 5x_{34}$$

$$\begin{array}{lcccccccccccccc}
\text{s.t.} & x_{11}+ & x_{12}+ & x_{13}+ & x_{14} & & & & & & & & & \leq & 40 \\
 & & & & & x_{21}+ & x_{22}+ & x_{23}+ & x_{24} & & & & & \leq & 50 \\
 & & & & & & & & & x_{31}+ & x_{32}+ & x_{33}+ & x_{34} & \leq & 40 \\
 & x_{11}+ & & & & x_{21}+ & & & & x_{31} & & & & = & 45 \\
 & & x_{12}+ & & & & x_{22}+ & & & & x_{32} & & & = & 20 \\
 & & & x_{13}+ & & & & x_{23}+ & & & & x_{33} & & = & 30 \\
 & & & & x_{14}+ & & & & x_{24}+ & & & & x_{34} & = & 30 \\
 & x_{11}, & x_{12}, & x_{13}, & x_{14}, & x_{21}, & x_{22}, & x_{23}, & x_{24}, & x_{31}, & x_{32}, & x_{33}, & x_{34} & \geq & 0.
\end{array}$$

If we solve this problem we obtain the following solution $\boldsymbol{x}^*$. (We typically denote the optimal solution by $\boldsymbol{x}^*$ and the optimal objective value by z^*.) The objective value of this solution is $z^* = 1,005 \times 10^6$:

$$\begin{array}{rcrrcrrcrrcr}
x^*_{11} & = & 0, & x^*_{12} & = & 15, & x^*_{13} & = & 25, & x^*_{14} & = & 0, \\
x^*_{21} & = & 45, & x^*_{22} & = & 0, & x^*_{23} & = & 5, & x^*_{24} & = & 0, \\
x^*_{31} & = & 0, & x^*_{32} & = & 5, & x^*_{33} & = & 0, & x^*_{34} & = & 30.
\end{array}$$

Due to the choice of indices it is very easy to interpret the solution in terms of the graph from Figure 1.1, see Figure 1.2.

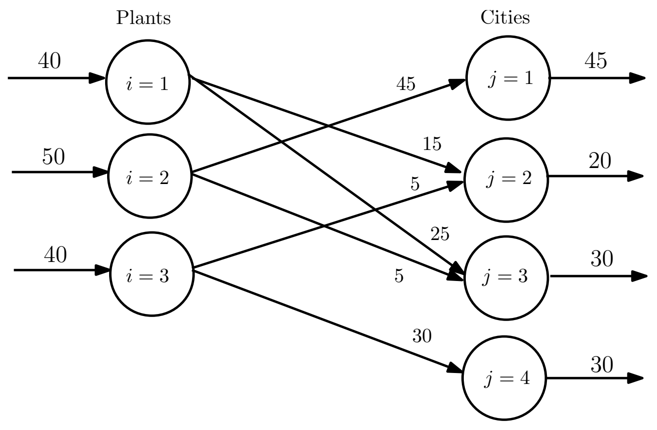


Figure 1.2: Solution of Example 1.2.

Here we have seen an example of double-index variables x_{ij}. We can also of course have three or more indices, depending on the problem, for instance x_{ijt} = the number of products transported from i to j in time period t.

1.3 Modeling problems that have a sequential aspect

Many problems have a natural time sequence embedded in its structure. At the start such problems might seem difficult to model, but again, it may help to use a network to illustrate the interaction between the time periods. We will give an example below.

Example 1.3. An event accessories company delivers napkins for a conference that lasts for five days. The demand for napkins for the five days is 900, 1440, 1260, 1620, 1296 respectively. For this event the napkins are delivered with a specific print, which renders the napkins worthless once the conference is over. The price for one napkin is \$2.50. The company has its own cleaning facility, where three different cleaning programs are used:

1. one-hour cleaning: the napkins are ready to use the next day,
2. fast cleaning: the napkins can be used again after two days,
3. standard cleaning: the napkins can be used again after three days.

The per unit costs for cleaning by the three different programs are \$0.50, \$0.25, and \$0.10 respectively. All the cleaning programs cause some discoloration of the special print on the napkins with the consequence that 40%, 20%, and 10%, respectively, of the napkins have to be discarded. The cleaning still has to be paid.

Model the problem of deciding how many new napkins should be bought, and how many napkins should be cleaned in the various cleaning programs each day, as a linear optimization problem. □

Input. Let

c_new	=	the per unit cost of buying a new napkin,
c_one	=	the per unit cost of the one-hour cleaning,
c_fast	=	the per unit cost of the fast cleaning,
c_stand	=	the per unit cost of the standard cleaning,
d_t	=	demand for napkins on day t.

In order to decide on a good set of decision variables we have to give it some thought. It is clear that we need to decide on how many new napkins we should buy and the number of napkins to be cleaned in the various cleaning programs each day:

Variable definition, part 1. Let

$xnew_t$	=	the number of new napkins bought for use on day t,
$xone_t$	=	the number of napkins cleaned by one-hour cleaning at the end of day t,
$xfast_t$	=	the number of napkins cleaned by fast cleaning at the end of day t,
$xstand_t$	=	the number of napkins cleaned by standard cleaning at the end of day t.

Moreover, it is clear that we can only buy and clean an integer number of napkins. So, the variables defined above should be restricted to take integer values only. What makes the modeling a bit more tricky than we have seen so far, is that a certain number of napkins are destroyed in each of the cleaning processes. The number of napkins that are lost, and the number of napkins that "survive" the cleaning should also be integer. Since the number sent to cleaning is restricted to be integer, it is sufficient to require either the number of lost napkins or the number of surviving napkins to be integer. Here we have decided on the number of lost napkins, but this choice is arbitrary. Either the surviving or the lost napkins therefore need to be introduced as decision variables that will be restricted to take integer values. Since the number of lost and clean napkins are required to be integer it might happen that we are forced to clean extra napkins just to achieve the integrality. We therefore allow for napkins to be wasted each day. We assume that we do not store wasted napkins to wash them at a later stage.

Variable definition, part 2. Let

$lossxone_t$	=	the number of napkins destroyed in one-hour cleaning at the end of day t,
$lossxfast_t$	=	the number of napkins destroyed in the fast cleaning at the end of day t,
$lossxstand_t$	=	the number of napkins destroyed in the standard cleaning at the end of day t,
$waste_t$	=	number of napkins thrown away after use on day t.

Before presenting the optimization model we again illustrate the problem by a graph, see Figure 1.3. The top node represents the buying of new napkins and has arcs to the nodes representing the five days. In these nodes we see the number of the day $(1, \ldots, 5)$ and the number of napkins required for that day.

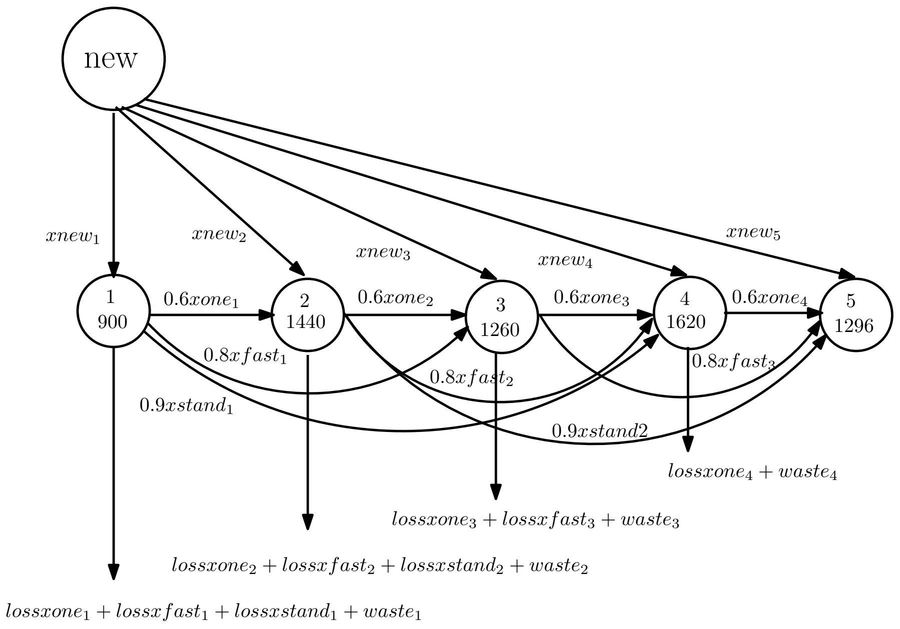


Figure 1.3: Graph representation of Example 1.3.

We now give the model for the general case of T days:

$$\begin{array}{rcl}
\min z = c\_new\sum_{t=1}^{T} xnew_t + c\_one\sum_{t=1}^{T-1} xone_t + c\_fast\sum_{t=1}^{T-2} xfast_t + c\_stand\sum_{t=1}^{T-3} xstand_t & & \\
\text{s.t. } lossxone_t - 0.4xone_t & = & 0,\ t = 1,\dots,T-1 \\
lossxfast_t - 0.2xfast_t & = & 0,\ t = 1,\dots,T-2 \\
lossxstand_t - 0.1xstand_t & = & 0,\ t = 1,\dots,T-3 \\
xnew_1 & \geq & d_1 \\
xnew_2 + 0.6xone_1 & \geq & d_2 \\
xnew_3 + 0.6xone_2 + 0.8xfast_1 & \geq & d_3 \\
xnew_t + 0.6xone_{t-1} + 0.8xfast_{t-2} + 0.9xstand_{t-3} & \geq & d_t,\ t = 4,\dots,T \\
xnew_1 - xone_1 - xfast_1 - xstand_1 - waste_1 & = & 0 \\
xnew_2 + 0.6xone_1 - xone_2 - xfast_2 - xstand_2 - waste_2 & = & 0 \\
xnew_3 + 0.6xone_2 + 0.8xfast_1 - xone_3 - xfast_3 - xstand_3 - waste_3 & = & 0 \\
xnew_t + 0.6xone_{t-1} + 0.8xfast_{t-2} + 0.9xstand_{t-3} - xone_t - xfast_t - xstand_t - waste_t & = & 0,\ t = 4,\dots,T-1 \\
xnew_t,\ xone_t,\ xfast_t,\ xstand_t,\ lossxone_t,\ lossxfast_t,\ lossxstand_t,\ waste_t & & \text{integer for relevant } t \\
\text{all variables} & \geq & 0.
\end{array}$$

The first three constraints just define the variables $lossxone_t$, $lossxfast_t$, and $lossxstand_t$. The following four constraints ensure that the inflow of napkins is large enough each day. The last four structural constraints guarantee that the inflow each day is equal to the outflow, so that no napkin is unaccounted for along the way. What we buy and what comes in from cleaning done in the previous days should be sent to cleaning at the end of the day or simply wasted if that turns out to be cheaper. As you can see we do not only have the standard non-negativity constraint, but again also a constraint saying that all variables have to take integer values.

If we solve the model for the given input we obtain the following result. We buy a total of 2,853 napkins divided over the days as follows.

	Day 1	Day 2	Day 3	Day 4	Day 5
New napkins	900	1,440	513	–	–

The choice of cleaning program is summarized in Table 1.2.

Cleaning Program	Day 1	Day 2	Day 3	Day 4
One hour	–	45	840	1,600
Fast	900	1,395	420	–
Standard	–	–	–	–
Total	900	1,440	1,260	1,600

Table 1.2: Choice of cleaning program, Example 1.3.

The cost of this solution is \$9,053.75 We illustrate the solution in the graph given in Figure 1.4. Notice that we only clean 1,600 napkins at the end of day 4. The remaining 20 are thrown away.

In the three models studied above either all or none of the variables were restricted to take integer values. We can easily imagine similar models in which a subset of the variables are restricted to

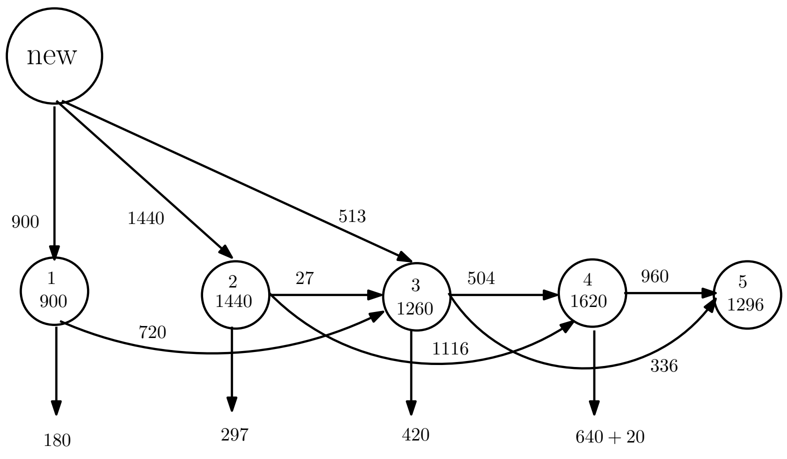


Figure 1.4: Solution of Example 1.3.

be integer whereas the remaining variables are continuous. We refer to such models as *mixed-integer models*, We will see examples of such models in the next chapter, and in particular we will see examples of so-called *combinatorial optimization problems.*

1.4 Exercises

Exercise 1.1. A company has 4 suppliers for a certain product and can deliver to 3 different customers. The table below indicates the demand of each customer, the supply of each supplier and the transportation costs from each supplier to each customer (per unit of the product). An x in the table indicates that a certain supplier is not able to deliver to a certain customer.

		customer			supply
		1	2	3	
	1	3	2.5	x	1200
supplier	2	4	x	3	500
	3	x	3	6	600
	4	2.5	3	x	1500
demand		1800	700	900	

The company aims at fulfilling all demand while minimizing the total transportation costs. The amounts of the product that are delivered do not need to be integer.

(a) What is the most logical choice for the decision variables?

(b) Formulate the problem as an LP.

(c) Solve the formulated LP using a tool of your choice, e.g. Simplex Method Tool, QSOpt or Matlab.

Exercise 1.2. A hospital needs to decide which surgeries will be executed the next day, such that the operating room (OR) is optimally utilized. More precisely, this means that the total time of all executed surgeries is maximized. There are two ORs available from 8am to 4pm. The next table gives the (expected) duration of each surgery on the waiting list (in hours).

surgery	1	2	3	4	5
duration	5	4	3.5	1.5	3

Give an ILP formulation of this problem.

Exercise 1.3. A group of 62 male and 34 female students wants to make homework exercises in pairs. Each student has made a list of preferred partners. Coincidentally, each male student has put only female students on his list, while each female student put only male students on her list.

All the lists are given to one student. Her task is to make as many pairs as possible, such that each pair consists of two students that are on each other's list.

Give an ILP formulation of this problem.

Exercise 1.4. In a certain region, there are p ambulances available and there are n locations where the ambulances could be placed (with $n \geq p$). We say that a house is *covered* if an ambulance is located within 10 minutes travel time. There are m houses in the region and the travel time between location i and house j is known and indicated as t_{ij}. Our goal is to locate the ambulances in such a way that as many houses are covered as possible. Give an ILP formulation of this problem.

Exercise 1.5. Give an ILP formulation of the problem of placing as many queens as possible on a chess board such that no two queens threaten each other. A chess board is an 8×8 matrix and two queens threaten each other when they are in the same row, in the same column, or in the same diagonal.

Exercise 1.6. Give an ILP formulation of the problem of placing as many knights as possible on a chess board such that no two knights threaten each other. A chess board is an 8×8 matrix and two knights threaten each other when they are either one row apart and two columns apart, or one column apart and two rows apart.

Exercise 1.7. A factory produces four products on two machines. The management wishes to increase the production rate by purchasing at most two additional machines. There are three machine types that they can choose from: type X, Y and Z. It is both possible two purchase two machines of the same type as well as to purchase two different machines. Both machines that are already present in the factory are of type X.

The following table indicates the processing time of each of the four products of each type of machine. A "_" indicates that it is not possible to produce this product on a certain machine.

	Product a	Product b	Product c	Product d
Machine of type X	2	_	5	4
Machine of type Y	1	3	2	_
Machine of type Z	_	4	4	2

The total processing time can be at most 24 hours per machine per day. The profit per produced product is indicated in the table below.

	Product a	Product b	Product c	Product d
Profit	600	300	500	900

The management needs to decide which machines it should purchase and how much of each product should be produced on each machine per day, as to maximize the total profit.

Give an ILP formulation of this problem.

Exercise 1.8. ([3], Exercise 1.9) The Candid Camera Company manufactures three lines of cameras: the Cub, the Quickiematic, and the VIP, whose contributions are \$3, \$9, and \$25, respectively. The distribution center requires that at least 250 Cubs, 275 Quickiematics, and 150 VIPs be produced each week.

Each camera requires a certain amount of time in order to: (1) manufacture the body parts; (2) assemble the parts (lenses are purchased from outside sources and can be ignored in the production planning decision); and (3) inspect, test, and package the final product. The Cub takes 0.1 hours to manufacture, 0.2 hours to assemble, and 0.1 hours to inspect, test and package. The Quickiematic needs 0.2 hours to manufacture, 0.35 hours to assemble, and 0.2 hours for the final set of operations. The VIP needs 0.7, 0.1, and 0.3 hours, respectively.

In addition, there are 250 hours per week of manufacturing time available, 350 hours of assembly, and 150 hours total to inspect, test, and package.

Formulate this production planning problem as a linear optimization problem that maximizes revenue.

Exercise 1.9. ([4], Exercise 1.6) A meat packing plant produces 480 hams, 400 pork bellies, and 230 picnic hams every day; each of these products can be sold either fresh or smoked. The total number of hams, bellies, and picnics that can be smoked during a normal working day is 420. In addition up to 250 products can be smoked on overtime at a higher cost. The net profits in \$ are as follows:

	Fresh	Smoked on regular time	Smoked on overtime
Hams	8	14	11
Bellies	4	12	7
Picnics	4	13	9

The objective is to find the schedule that maximizes the total net profit. Formulate as a linear optimization problem.

Chapter 2

Modeling Combinatorial Optimization Problems

Here we will focus on linear mixed-integer optimization models in which the integer variables are restricted to take the values 0 or 1. Another word for 0-1 variables is *binary variables*. Such models typically represent certain choices that are to be made, such as whether to include an element or not in a certain set.

One of the simplest 0-1 optimization problems is the knapsack problem. Here we are given a set $N = \{1, \ldots, n\}$ of items. Each item $j \in N$ has a weight and a "profit". We would like to decide on which items to bring in the "knapsack", which has a capacity of b, such that the total profit is maximized.

Input. Let

p_j	$=$	the profit of item j, $j \in N$,
a_j	$=$	the weight of item j, $j \in N$,
b	$=$	the capacity of the knapsack.

Variable definition. Let

$$x_j = \begin{cases} 1, & \text{if item } j \text{ is included in the knapsack} \\ 0, & \text{otherwise.} \end{cases}$$

We can now formulate the knapsack problem as follows:

$$\begin{aligned} \max z = & \sum_{j=1}^{n} p_j x_j \\ \text{s.t.} & \sum_{j=1}^{n} a_j x_j \leq b \\ & x_j \in \{0, 1\}, \; j = 1, \ldots, n \end{aligned}$$

We want to make three remarks:

1. In the definition of the input we used the notation $j \in N$ whereas we in the problem formulation used $j = 1, \ldots, n$. These variants mean the same and will be used interchangeably.

2. If we replace the $\leq$-sign in the knapsack constraint by the $=$-sign we obtain the so-called "subset sum" problem. Here we require the sum of the weights of the subset of chosen items to add up exactly to b.

3. If we replace $x_j \in \{0, 1\}$, $j = 1, \dots, n$ by x_j integer, $j = 1, \dots, n$, we obtain the integer knapsack problem, where x_j denotes the number of item i that is included in the knapsack.

The knapsack problem might not occur in practice as a stand-alone problem, but a vast number of 0-1 or integer optimization problems contain knapsack constraints, and it is therefore mathematically interesting to study the problem.

Typically, the 0-1 optimization problems fall into the category of *combinatorial optimization* problems. There is no unique definition of what a combinatorial optimization problem is, but it is quite usual to refer to a problem as a combinatorial optimization problem if we have a ground set N of items, such as in the knapsack problem, and we seek an optimal subset $S \subseteq N$ of this ground set, that is, we can write the problem as

$$\max \text{ (or min)}_{S\subseteq N} \{\sum_{j\in S} c_j \mid S \in \mathfrak{F}\},$$

where c_j is the objective coefficient of item j, and where $\mathfrak{F}$ is the set of feasible solutions. In the knapsack problem, the objective coefficients are denoted by p_j and the feasible set is defined as

$$\mathfrak{F} = \{x \in \{0, 1\}^n \mid \sum_{j\in N} a_j x_j \leq b\}\,.$$

2.1 Modeling fixed costs

In many applications we are dealing with a cost structure as follows:

$$c(x) = \begin{cases} ax + f, & \text{if } x > 0 \\ 0, & \text{otherwise.} \end{cases}$$

Such a cost structure is present whenever we have to pay a fixed cost for setting up an activity plus a per-unit cost for using it, see Figure 2.1.

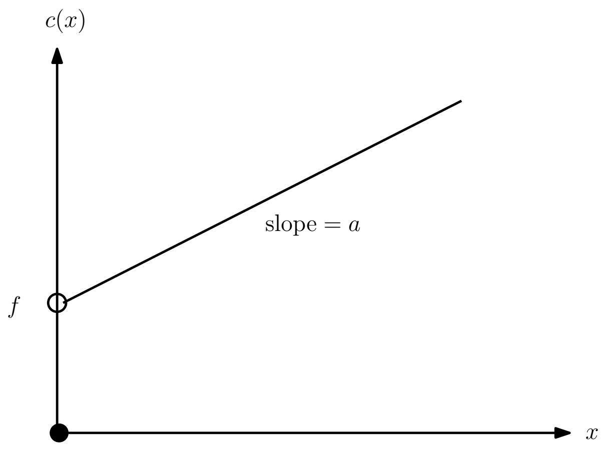


Figure 2.1: Discontinuous cost structure.

An example where such a cost structure is present is the uncapacitated facility location problem.

Example 2.1. We are given a set N of clients and a set M of possible sites where we can locate a facility. The cost of opening a facility at site $i \in M$ is f_i and the cost of supplying all of client j's demand from facility i is c_{ij}, $i \in M$, $j \in N$. We want to decide on at which sites we shall open a facility, and from which open facilities each client is served. The goal is to minimize the total cost under the restriction that the demand of each client has to be met. Formulate the problem as a mixed-integer linear optimization problem. An example of an instance is given in Figure 2.2. □

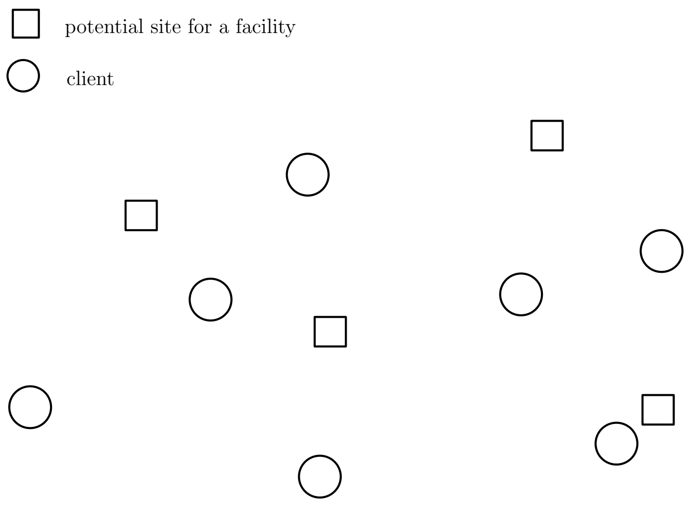


Figure 2.2: An example of an instance of the uncapacitated facility location problem.

Input. Let

f_i	$=$	the fixed cost of opening a facility at site i, $i \in M$,
c_{ij}	$=$	the cost of supplying all demand of client j from facility i, $i \in M$, j, $j \in N$.

The question now is which variables to introduce. It is natural to have one type of variable indicating the fraction of client j's demand supplied by facility i.

Variable definition, part 1. Let

x_{ij} $=$ fraction of client j's demand supplied from facility i, $i \in M$, $j \in N$.

Using this set of variables we can model the restriction that the demand of each client has to be met:

$$\sum_{i \in M} x_{ij} = 1, \quad j \in N\,.$$

Moreover, we can model part of the objective function, namely $\sum_{i\in M}\sum_{j\in N} c_{ij}x_{ij}$, but, how can we model the fixed cost of opening a facility at site $i \in M$ using a linear objective function and constraints? The trick is to introduce a binary variable y_i indicating whether or not we open a facility at site $i \in M$.

Variable definition, part 2. Let

$$y_i = \begin{cases} 1, & \text{if a facility is opened at site } i \in M \\ 0, & \text{otherwise.} \end{cases}$$

Using the y_i-variables we can model the other part of the objective function, namely the fixed costs: $\sum_{i \in M} f_i y_i$. If we leave it at this, all the y_i-variables will take the value 0 (assuming all fixed costs are positive), which is not what we want. What we need is a constraint that forces y_i to take value 1 if the $x_{ij} > 0$ for any $j \in N$. This is done by the constraints $y_i \geq x_{ij}$ for all $j \in N$. We are now ready to write down the complete model.

$$
\begin{aligned}
\min z = \sum_{i \in M} c_{ij} x_{ij} + \sum_{i \in M} f_i y_i & \\
\text{s.t.} \sum_{i \in M} x_{ij} = 1, &\qquad j \in N \\
y_i - x_{ij} \geq 0, &\qquad i \in M,\ j \in N \\
x_{ij} \geq 0, &\qquad i \in M,\ j \in N \\
y_i \in \{0,1\},\ i \in M\,. &
\end{aligned}
$$

An example of a feasible solution to the instance illustrated in Figure 2.2, can be found in Figure 2.3.

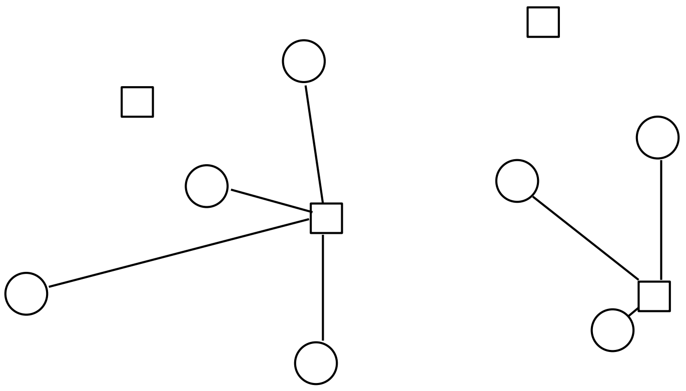


Figure 2.3: An illustration of a feasible solution to the instance given in Figure 2.2.

We see that two of the facility sites are not used and that, in this case, all clients are served by a single facility. In fact, it is easy to show that if a client is fractionally served by two or more facilities, then there exists a solution, with the same objective value, in which the client is served by a single facility. This is left as an exercise to the reader.

2.2 Modeling max/min problems

Example 2.2. A student wants to determine an optimal strategy for preparing for the next exam period in which he has n exams. For each course he can choose among m levels of preparation. If he prepares course j at level i he gets the grade c_{ij}, and in order to prepare course j at level i he needs t_{ij} hours. He has only T hours in total for his preparation. We assume that both c_{ij} and t_{ij} are integer for all i, j. His goal is to spend his hours in such a way that the lowest grade that he obtains is as high as possible. Formulate the problem as a *linear* mixed-integer optimization problem. $\square$

Input. Let

c_{ij}	$=$	the grade obtained after preparing course j at level i, $i = 1, \dots m$, $j = 1, \dots, n$,
t_{ij}	$=$	the number of hours needed to prepare course j at level i, $i = 1, \dots m$, $j = 1, \dots, n$,
T	$=$	the total number of hours available.

This problem is a so-called "max/min" problem, and we have to think carefully how to model this using only linear constraints and a linear objective function. Let us first think about the variable definition. It is clear that we need a 0-1 variable x_{ij} indicating whether or not he prepares a course j at level i. Using this set of variables we can model important parts of the problem namely:

- the student can prepare course j at exactly one level,
- the total time spent should not exceed T.

These two constraints can be written mathematically as:

$$\begin{aligned} \sum_{i=1}^{m} x_{ij} &= 1, \quad j = 1, \dots, n \,, \qquad (2.1) \\ \sum_{i=1}^{m}\sum_{j=1}^{n} t_{ij}x_{ij} &\leq T \,. \end{aligned}$$

The question now is how to link the study efforts to the resulting grade and how to enforce the "max/min" aspect. The grade is really just an outcome of the preparation and we can therefore just define a variable z_j saying which grade the student gets depending on the preparation. Constraint (2.1) ensures that each course is prepared at exactly one level.

$$z_j = \sum_{i=1}^{m} c_{ij}x_{ij}, \quad j = 1, \dots, n \,,$$

or, equivalently,

$$z_j - \sum_{i=1}^{m} c_{ij}x_{ij} = 0, \quad j = 1, \dots, n \,,$$

With all the grades in place we would like to determine $\max\{\min_j z_j\}$, but this is clearly not linear as it is written here. In order to obtain a linear problem we can, however, introduce an extra auxiliary variable z that we maximize with the constraint that $z \leq z_j, \; j = 1, \dots, n$. To summarize, we get the following set of variables:

Variable definition. Let

x_{ij}	$=$	1 if course j is prepared at level i and 0 otherwise, $i = 1, \dots m$, $j = 1, \dots, n$,
z_j	$=$	the grade obtained for course j, $i = 1, \dots m$, $j = 1, \dots, n$,
z	$=$	the lowest grade obtained.

The complete model now becomes:

$$\begin{array}{rcl} \max z & & \\ \text{s.t. } \sum_{i=1}^{m} x_{ij} & = & 1, \quad j = 1, \ldots, n\,, \\ \sum_{i=1}^{m}\sum_{j=1}^{n} t_{ij}x_{ij} & \leq & T\,, \\ z_j - \sum_{i=1}^{m} c_{ij}x_{ij} & = & 0, \quad j = 1, \ldots, n\,, \\ z - z_j & \leq & 0, \quad j = 1, \ldots, n\,, \\ z,\ z_j & \geq & 0, \quad j = 1, \ldots, n\,, \\ x_{ij} & \in & \{0,1\}, \quad i = 1, \ldots, m\ j = 1, \ldots, n\,. \end{array}$$

It is clear that min/max-problems can be modeled in a similar fashion.

2.3 Disjunctions

There are situations in which we want to model that only a subset of the constraints should be active. In a regular model we require all constraints to be active, but in some cases it is relevant to make a choice. Typical examples where this occurs are machine scheduling problems, in which we want to determine in which order jobs are processed on machines. Consider two jobs i and j. Then, either i precedes j on a certain machine, or j precedes i. The question is how to model this using linear expressions.

Suppose we have two constraints:

$$\begin{array}{rcl} a^1 x & \leq & b_1, \\ a^2 x & \leq & b_2\,, \end{array}$$

where a^1 and a^2 are n-dimensional row vectors, $x \in \mathbb{R}^n$ and $0 \leq x_j \leq u_j,\ j = 1, \ldots, n$. To model that only one of the two constraints is active, we introduce one 0-1 variable for each of the constraints.

Variable definition. Let

$$y_i = \begin{cases} 1, & \text{if constraint } i \text{ is active} \\ 0, & \text{otherwise.} \end{cases} \qquad i = 1, 2$$

We also need to introduce a constant M that should have a value that is "large enough". Having the constant M and the binary variables $y_i,\ i = 1, 2$, we can model the disjunction as follows:

$$\begin{array}{rcl} a^1 x - b_1 & \leq & M(1 - y_1) \\ a^2 x - b_2 & \leq & M(1 - y_2) \\ y_1 + y_2 & = & 1 \\ x_j & \leq & u_j,\ j = 1, \ldots, n \\ x_j & \geq & 0,\ j = 1, \ldots, n \\ y_i & \in & \{0,1\},\ i = 1, 2\,. \end{array} \tag{2.2}$$

Notice that in the case that we choose between two constraints like in the model above, one binary variable y suffices. We define y as follows:

Alternative variable definition. Let

$$y = \begin{cases} 1, & \text{if constraint (2.2) is active} \\ 0, & \text{otherwise.} \end{cases}$$

The resulting model is then:

$$\begin{array}{rcl} a^1x - b_1 & \leq & M(1-y) \\ a^2x - b_2 & \leq & My \\ x_j & \leq & u_j,\ j = 1, \ldots, n \\ x_j & \geq & 0,\ j = 1, \ldots, n \\ y & \in & \{0,1\}. \end{array}$$

2.4 Modeling connectivity

In several combinatorial optimization problems, especially graph-related problems, we need to choose subsets of the variables such that this subset has certain properties. An example is the Minimum Spanning Tree (MST) problem.

Given an undirected graph $G = (V, E)$, a tree $G' = (V', T)$, $V' \subseteq V$ in G is a connected graph such that $|T| = |V'| - 1$. If $V' = V$, then the tree is a *spanning tree* in G, i.e., the tree spans all the vertices in G, see Figure 2.4. An MST in G is a spanning tree in which the sum of the edge lengths is minimum.

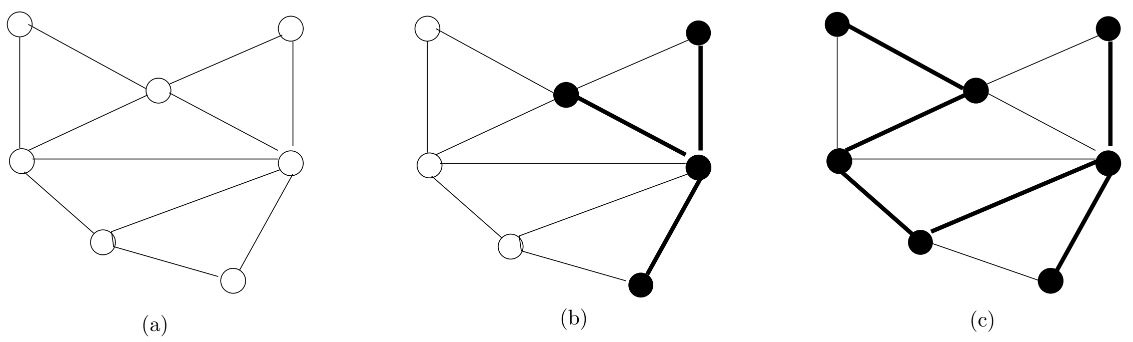


Figure 2.4: (a) An undirected graph G, (b) a tree in G, (c) a spanning tree in G.

Input. Let
$G = (V, E)$, and undirected graph
l_e = the length of edge $e \in E$.

To model the MST-problem as a binary linear optimization problem it is natural to use the following variables:

Variable definition. Let

$$x_e = \begin{cases} 1, & \text{if edge } e \text{ is part of the MST} \\ 0, & \text{otherwise.} \end{cases} \qquad e \in E$$

It is quite straightforward to model both the objective function and the requirement that the number of edges is equal to $|V|-1$:

$$\begin{aligned} \min \sum_{e\in E} l_e x_e & & \\ \text{s.t.} \sum_{e\in E} x_e &= |V|-1 & (2.3)\\ x_e &\in \{0,1\},\ e\in E\,. & (2.4) \end{aligned}$$

This model is, however, not complete since we have not yet included constraints that enforce the resulting subgraph to be connected. If we consider the graph in Figure 2.4(a), the subgraph in Figure 2.5(a) is feasible with respect to Constraints (2.3) and (2.4), but it is not a spanning tree.

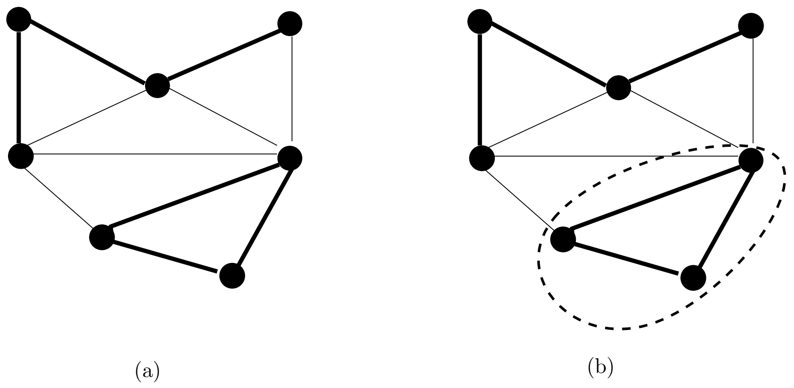


Figure 2.5: (a) A subgraph satisfying Constraints (2.3) and (2.4), (b) A subset of the vertices with no adjacent MST-edge.

The difficulty in modeling the MST-problem lies in the modeling of the connectivity requirements. We need to enforce that, for each proper subset S of the vertices, there should be an MST-edge between S and $V\setminus S$. By a proper subset we mean that $S\neq\emptyset$ and $S\neq V$. The encircled set of vertices in Figure 2.5(b) does not have an MST-edge connecting it to the complement set of vertices.

The complete MST-model then becomes:

$$\begin{aligned} \min \sum_{e\in E} l_e x_e & & \\ \text{s.t.} \sum_{e\in E} x_e &= |V|-1 & \\ \sum_{\{e:=\{i,j\}\in E|i\in S,\ j\in V\setminus S\}} x_e &\geq 1 \qquad \text{for all } S\subsetneq V, S\neq\emptyset & (2.5)\\ x_e &\in \{0,1\},\ e\in E\,. & \end{aligned}$$

What Constraints (2.5) are saying is that if we look at each proper subset S of the vertices and we sum over all the variables representing edges having one endpoint in S and the other endpoint in the complement of S, then at least one of these variables should take value 1. Since there are exponentially many subsets of V, constraint class (2.5) is exponentially large. This might seem prohibitive, but in fact it is not in this case. We will return to this issue in Section 3. Notice that we can also model the connectivity requirements as follows.

$$\sum_{e\in E(S)} x_e \leq |S|-1 \quad \text{for all } S\subsetneq V\,,$$

where $E(S)$ is the set of edges with both endpoints in S.

A second problem where similar constraints as (2.5) occur is the well-known Traveling Salesman Problem (TSP). Here we again have an undirected graph $G = (V, E)$ and we know the length of each edge $e \in E$. We assume that the graph is *complete*, i.e., all the edges are present, possibly with length equal to infinity.

Input. Let
$G = (V, E)$, a complete undirected graph,
l_e = the length of edge $e \in E$.

The problem is to determine a cycle on all vertices in G such that the length of the cycle is minimized. A different way of interpreting the problem is to view the vertices as "cities" and requiring a "salesman" to visit each city exactly once and then returning to the starting city in such a way that the length of the resulting "tour" is minimized. An example of a TSP-tour is given in Figure 2.6(b)

Variable definition. Let

$$x_e = \begin{cases} 1, & \text{if edge } e \text{ is part of the TSP-cycle} \\ 0, & \text{otherwise.} \end{cases} \qquad e \in E$$

It is again easy to model the objective function, which is the same as in the MST-problem. To say that each city should be visited exactly once is the same as requiring that the number of TSP-edges adjacent to each vertex should be exactly 2; one is used to enter the "city", and one to leave it. Let $\delta(v)$ be the set of edges adjacent to vertex v, i.e., $\delta(v) = \{e \in E \mid v \text{ is one of the endpoints of } e\}$. We would then obtain:

$$\begin{aligned} \min \sum_{e\in E} l_e x_e & \\ \text{s.t.} \sum_{e\in\delta(v)} x_e &= 2 \quad \text{for all } v \in V && (2.6) \\ x_e &\in \{0,1\},\ e \in E\,. && (2.7) \end{aligned}$$

All TSP-cycles are feasible in this model, but we also allow for several so-called "subtours" that only encompass a subset of the vertices, see Figure 2.6(a). To exclude the subtours we need to add so-

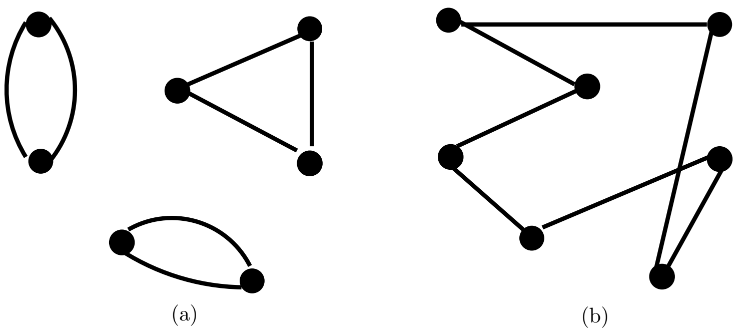


Figure 2.6: (a) Subtours satisfying Constraints (2.6) and (2.7), (b) A TSP-tour.

called "subtour elimination constraints". These are very similar to the connectivity constraints that we added in the MST-problem. The full model of the TSP-problem is as follows:

$$\begin{aligned} \min \sum_{e\in E} l_e x_e & & & \\ \text{s.t.} \sum_{e\in\delta(v)} x_e &= 2 & & \text{for all } v \in V \\ \sum_{\{\{i,j\}\in E \mid i\in S,\ j\in V\setminus S\}} x_e &\geq 2 & & \text{for all } S \subsetneq V, S \neq \emptyset \qquad (2.8) \\ x_e &\in \{0,1\},\ e \in E\,. \end{aligned}$$

Constraints (2.8) require that at least two edges have to be adjacent to any proper subset of vertices; the salesman has to enter the set at some city, and he has to leave the set at some city. He may also enter and leave the set more than one time, but this is allowed by Constraints (2.8).

2.5 Modeling integer variables by binary variables

Suppose we want to produce an item in certain batch sizes, say 0, 5, 15, 30, and 50. The natural thing to do is to introduce a variable x, representing the number of produced items, and require $x \in \{0, 5, 15, 30, 50\}$. But this does not immediately fit in our framework of linear integer optimization. The way to deal with this is to introduce one 0-1 variable for each of the four positive batch sizes:

Variable definition. Let

$$y_i = \begin{cases} 1, & \text{if we produce the } i\text{th positive batch size} \\ 0, & \text{otherwise,} \end{cases} \qquad i = 1, \ldots, 4\,,$$

where the batch sizes are 5, 15, 30, 50 respectively. We then get the following constraints:

$$\begin{aligned} x - 5y_1 - 15y_2 - 30y_3 - 50y_4 &= 0 \\ \sum_{i=1}^{4} y_i &\leq 1 \\ y_i &\in \{0,1\}, \quad i = 1, \ldots, 4\,. \end{aligned}$$

We may also want to model general integer variables by binary variables in the view of possible computational advantages. Suppose we have a variable x that can take any nonnegative integer value, i.e. $x \in \mathbb{Z}_{\geq 0}$. Then we can write x as

$$\begin{aligned} x &= \sum_{i=0}^{k} 2^i y_i \\ y_i &\in \{0,1\}, \quad i = 1, \ldots, k\,. \end{aligned}$$

An important question here is which value we should choose for k if x does not have an easily computable bound. An answer to this question is given in the proof of Theorem 13.6 in the book by Papadimitriou and Steiglitz [9].

2.6 Exercises

Exercise 2.1. Correct each of the following failed attempts at producing an (I)LP formulation. You may introduce new decision variables.

(a)

$$\begin{array}{ll} \max & \min\{x,y,z\} \\ s.t. & 2x+3y-z \geq 5 \\ & -3x-4y+2z \leq 3 \\ & x,y,z \geq 0 \end{array}$$

(b)

$$\begin{array}{ll} \min & 2x-3y \\ s.t. & 2x+3y \geq 5 \quad \text{or} \quad 3x-4y \leq 3 \\ & x,y \in \mathbb{N} \end{array}$$

(c)

$$\begin{array}{ll} \min & x+y \\ s.t. & 2x+3y \geq 5 \\ & -3x-4y \leq 3 \\ & x \in \{2,5,8,200\} \\ & y \in \mathbb{N} \end{array}$$

Exercise 2.2. Suppose you study n patients who suffer from a certain disease and you have selected m genes that may be related to this disease. You suspect that for each of these patients the disease is caused by a mutation in one of these genes. Your data is given as an $n \times m$ matrix M with $M_{i,j} = 1$ if gene j is mutated in patient i and $M_{i,j} = 0$ otherwise. To find out which genes are causing the disease, you want to find a smallest possible set J of genes, such that in each patient at least one gene from J is mutated. Give an ILP formulation of this problem.

Exercise 2.3. "Next-generation sequencing technology" is the general term for modern methods for producing DNA sequences. An important step in the analysis of the data produced by these methods is the assignment of obtained pieces of DNA, which are called *reads*, to locations on a reference genome. Given are reads $r_1, \ldots, r_n$ and locations $\ell_1, \ldots, \ell_m$ on a reference genome. For each read r_i and location ℓ_j the similarity s_{ij} between the read and the location on the reference genome are known. In addition, for each combination of two locations ℓ_j and ℓ_k is given whether they overlap ($o_{jk} = 1$) or not ($o_{jk} = 0$). We want to assign each read to exactly one location on the reference genome. To each location, we may assign zero, one, or multiple reads. However, the locations to which reads are assigned may not overlap. Hence, if two locations overlap than we may assign reads to at most one of the two locations. We want to maximize the sum of the similarities s_{ij} of all assignments. Give an ILP formulation of this problem.

Exercise 2.4. At an industrial design faculty, n students each need to do 3 projects. In total, there are p projects available. The students have given each of the projects a number from $\{1, \ldots, 10\}$ to indicate how much they want to do the project (1 indicating "not at all" and 10 indicating "very much"). Let c_{ij} denote the number given by student i to project j. There are p project rooms available, each with a certain capacity. Let m_k be the number of students that fits in project room k.

The faculty wants to assign students to projects and projects to project rooms, as to maximize the lowest number assigned by a student to a project that he/she has to do. Of course, each project needs to be assigned to a room with sufficient capacity. There can be at most one project in each room.

Give an ILP formulation of this problem.

Exercise 2.5. Give an ILP formulation of the following graph problem.

Minimum Steiner Tree
Given: a graph $G = (V, E)$, a length $\ell(e) > 0$ of each edge $e \in E$ and a set $T \subseteq V$ of *terminals*.
Find: a subgraph of G that is a tree and contains all terminals from T, such that the total length of all edges in the tree is minimized.

Exercise 2.6. Consider the following scheduling problem. There are n jobs and m machines on which the jobs can be processed. If job i is processed on machine j then the processing time is p_{ij}. We need to assign each job to exactly one of the machines.

(a) Give an ILP formulation for this problem, if we want to minimize the *makespan*, which is defined as $C_{max} = \max_j C_j$ with C_j the time that machine j is finished processing all its jobs.

(b) Now consider the situation that some of the jobs need additional resources to be processed. For each resource r, the set $N_r = \{i \mid \text{job } i \text{ needs resource } r\}$ is given. Since moving the resources takes too much time, jobs using the same resource need to be processed on the same machine. Our goal is now to minimize the total processing time (i.e. the sum of the processing times of the machines). Give an ILP formulation of this problem.

(c) Again consider the situation where jobs need additional resources but the resources can now be moved between the machines (this does not take time). However, at each point in time, each resource can be used on at most one machine. Hence, two jobs that need the same resource cannot be processed simultaneously. Give an ILP formulation of this problem, minimizing the makespan. To simplify the problem, you may assume that $p_{ij} = 1$ for all i and j.

Exercise 2.7. In a hospital, there are different types of surgeries need to be performed. Let n_s denote the number of surgeries of type s that are performed per year. For each surgery, certain instruments are need. Let r_{is} denote the number of instruments of type i that are required at a surgery of type s. The instruments are collected on metal trays. Let f_i denote the fraction of a tray that is used by an instrument of type i. Before each surgery, trays carrying the necessary instruments are transported to the operation room. After the surgery, all instruments on all used trays need to be sterilized. The costs of sterilizing a tray with its instruments are S. In addition, there are fixed costs F per tray.

The hospital wants to compose the trays in such a way that the total costs (fixed costs plus sterilization costs) are minimized. Therefore, the hospital needs to decide how many instruments of each type will be assigned to each tray and which trays will be used at which type of surgery. The hospital can also decide not to assing any instruments to a certain tray, in which case no fixed costs need to be paid for that tray. Of course, they have to make sure that at each surgery all required instruments are available. In addition, the instruments assigned to a tray must all fit on that tray.

Give an ILP formulation of this problem.

Exercise 2.8. (a) A company has two products $k = 1, 2$, one factory, two distribution centres $i = 1, 2$ and five important customers $j = 1, \ldots, 5$. The demand of each customer j for product k is known, $d_{jk} > 0$. The company must decide which products are handled by which distribution centre and which distribution centre handles the orders of each customer. For each combination of customer and product, the product can only be handled by **one** distribution centre.

Formulate this problem as an ILP in which the total cost must be minimized. The cost is given as follows:

i. A fixed cost f_{ik} if product k is handled by distribution centre i.
ii. A fixed cost f_{ijk} if the demand of customer j for product k is handled by distribution centre i.

iii. A cost of c_{ijk} per unit for transporting product k from distribution centre i to customer j.

(b) How does the model change if a customer can split he handling of a product over multiple distribution centres?

Part II

Linear Programming

Chapter 3

Polytopes and Polyhedra

In this chapter we present the basic definitions and results regarding polytopes and polyhedra in order to explain and relate results in linear optimization that will be presented in the following chapters. Section 3.2 contains more advanced material and forms a useful background for Chapters 4 and 12.

3.1 Basic definitions and results

As we have seen in the previous chapters, the set of feasible solutions to an optimization problem is formed by finitely many constraints. If we have a linear (integer) optimization problem, the constraints are linear, i.e., they can be written in the form

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b\,.$$

Other ways of writing such a constraint is

$$\sum_{j=1}^{n} a_jx_j \leq b\,,$$

or,

$$\boldsymbol{ax} \leq b\,,$$

where we view $\boldsymbol{a}$ as a row vector.

Definition 3.1. A set of points that satisfies a linear inequality $\boldsymbol{ax} \leq b$, $HS = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{ax} \leq b\}$, is called a *half-space*.

Definition 3.2. A set of points $H = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{ax} = b\}$ is called a *hyperplane*.

If we take the set of points satisfying finitely many constraints, which is precisely the form of the set of feasible solutions to a linearly constrained optimization problem, we obtain a polyhedron.

Definition 3.3. A set $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a}_i\boldsymbol{x} \leq b_i,\ i = 1, \ldots, m\}$ is called a *polyhedron*. If the polyhedron is bounded it is called a *polytope*.

An illustration of a half-space, a polyhedron, and a polytope is given in Figure 3.1. The set P in Definition 3.3 can be written as $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$, where b_i, the ith element of the vector $\boldsymbol{b}$, and $\boldsymbol{a}_i$ is the ith row of the matrix A, which explains why we view $\boldsymbol{a}$ as a row vector.

Both in linear and nonlinear optimization, *convexity* will play an important role. In this chapter we introduce convex sets and convex combinations. We introduce convex functions in Chapter 18. If the feasible set is convex, and/or the function we optimize is convex, it typically means that the associated optimization problem is "nicer" to deal with.

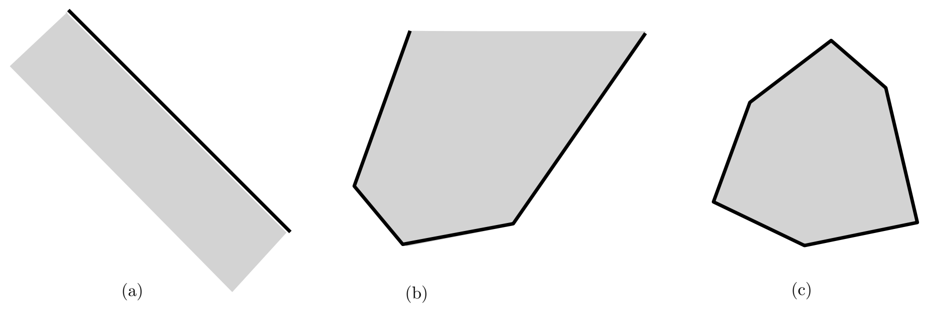


Figure 3.1: (a) A halfspace generated by a linear inequality $\boldsymbol{ax} \leq b$. (b) A polyhedron. (c) A polytope.

Definition 3.4. A set S is *convex* if for any two points $\boldsymbol{x}^1,\ \boldsymbol{x}^2 \in S$ all points $\boldsymbol{x}$ that can be written as $\boldsymbol{x} = \lambda\boldsymbol{x}^1 + (1-\lambda)\boldsymbol{x}^2$ for $0 \leq \lambda \leq 1$ belong to S.

So, what the definition says, is that a set S is convex if we can take any two points in S, draw a line between them, and all points on that line segment belong to S as well.

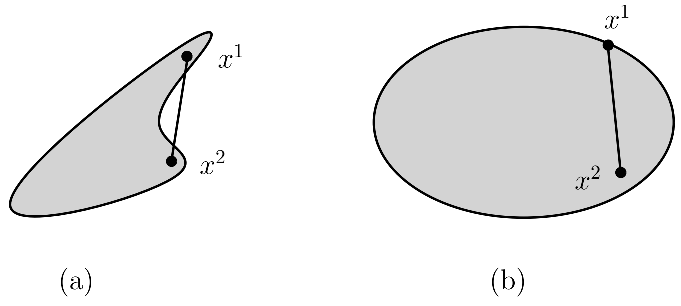


Figure 3.2: (a) A non-convex set. The line connecting $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ contains points that do not belong to the set. (b) A convex set. Regardless of how we choose $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$, all points on the line connecting these points belong to the set.

Example 3.1. A halfspace is a convex set. To see that, let $HS = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{ax} \leq b\}$, and take $\boldsymbol{x}^1, \boldsymbol{x}^2 \in HS$. Let $\boldsymbol{x} = \lambda\boldsymbol{x}^1 + (1-\lambda)\boldsymbol{x}^2$ for $0 \leq \lambda \leq 1$. We now have to show that $\boldsymbol{x} \in HS$.

$$\boldsymbol{ax} = \lambda\boldsymbol{ax}^1 + (1-\lambda)\boldsymbol{ax}^2 \leq \lambda b + (1-\lambda)b = b\,,$$

where the inequality is a consequence of $\boldsymbol{x}^1, \boldsymbol{x}^2 \in HS$. □

A very useful result is that the intersection of finitely many convex sets is a convex set. We will prove this result next.

Theorem 3.1. Let $X_1, \ldots, X_k$ be convex sets. Then, the set

$$S = \cap_{i=1}^k X_i$$

is a convex set.

Proof. Take any $\boldsymbol{x}^1, \boldsymbol{x}^2 \in \cap_{i=1}^k X_i$, and let $0 \leq \lambda \leq 1$. As a consequence of both $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ belonging to the intersection of $X_1, \ldots, X_k$, it holds that for $i = 1, \ldots, k$, $\boldsymbol{x}^1 \in X_i$ and $\boldsymbol{x}^2 \in X_i$. Therefore $\boldsymbol{x} = \lambda\boldsymbol{x}^1 + (1-\lambda)\boldsymbol{x}^2 \in X_i$ due to the convexity of X_i. Hence, $\boldsymbol{x} \in \cap_{i=1}^k X_i$.

□

Observation 3.1. A consequence of halfspaces being convex sets, and Theorem 3.1, is that a polyhedron is a convex set. Notice that a ball is also a convex set, but it is *not* a polyhedron as it cannot be written as a set of points that is defined by finitely many constraints.

Definition 3.5. If $\boldsymbol{x}^1, \dots, \boldsymbol{x}^k \in \mathbb{R}^n$, and $\lambda_1 \cdots \lambda_k$ are real scalars, then a vector $\boldsymbol{x} \in \mathbb{R}^n$ is called a *convex combination* if

$$\boldsymbol{x} = \sum_{i=1}^{k} \lambda_i \boldsymbol{x}^i, \quad \sum_{i=1}^{k} \lambda_i = 1, \quad \lambda_i \geq 0 \text{ for all } i = 1, \dots, k\,.$$

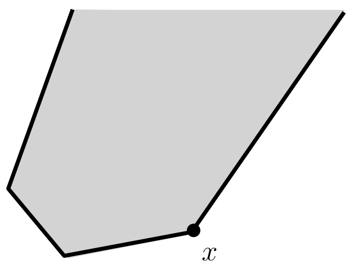


Figure 3.3: An extreme point $\boldsymbol{x}$ of a polyhedron.

Definition 3.6. Given is a polyhedron $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$. A point $\boldsymbol{x} \in P$ is an *extreme point* of P if $\boldsymbol{x}$ cannot be written as a convex combination of two distinct points in P.

So, geometrically, an extreme point of P is a "corner point" of P, see Figure 3.3. In the literature, an extreme point of a polyhedron P is often called a *vertex* of P.

Theorem 3.2. Given is a linear optimization problem

$$\max z = \boldsymbol{c}\boldsymbol{x} \text{ subject to } \boldsymbol{x} \in P = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}.$$

Assume that $P \neq \emptyset$. If the objective function value of the problem is bounded, then there is an optimal solution at an extreme point of P.

Extreme points of polyhedra play a center role in the Simplex algorithm for solving linear optimization problems. The Simplex method will be presented in Chapter 4.

Notice that an optimal solution does not have to be unique. If more than one vertex is optimal, say vertices $\boldsymbol{x}^1, \dots, \boldsymbol{x}^k$, then we can write the set of all optimal solutions as the convex combination of $\boldsymbol{x}^1, \dots, \boldsymbol{x}^k$, i.e. any vector $\boldsymbol{x}$ that can be written as

$$\boldsymbol{x} = \lambda_1 \boldsymbol{x}^1 + \cdots + \lambda_k \boldsymbol{x}^k,$$

with $\lambda_i \geq 0,\ i = 1, \dots, k,\ \sum_{i=1}^{k} \lambda_i = 1$, is optimal see also Chapter 4.3.

Example 3.2. Consider the polytope in Figure 3.4. The vertices $(\boldsymbol{x}^1)^\mathsf{T} = (1,\ 3/2)$, $(\boldsymbol{x}^2)^\mathsf{T} = (2, 1)$ are optimal since the objective function z is parallel to the constraint that is common to both $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$. So any vector

$$\boldsymbol{x} = \lambda \begin{pmatrix} 1 \\ 3/2 \end{pmatrix} + (1 - \lambda) \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

with $0 \leq \lambda \leq 1$ is an optimal solution. Choosing for instance $\lambda = 1/4$ yields the optimal solution $\boldsymbol{x}^\mathsf{T} = (7/4,\ 9/8)$.

□

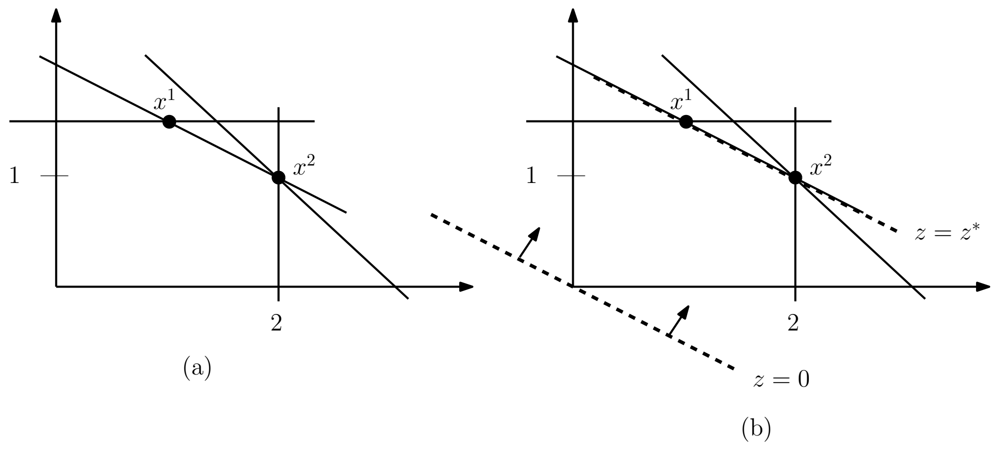


Figure 3.4: (a) The problem in Example 3.2. (b) The vertices $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ are optimal. z^* is the optimal objective value.

Definition 3.7. Given is a set of points $X = \{x^k\}_{k \in K}$, $\boldsymbol{x}^k \in \mathbb{R}^n$ for all $k \in K$. The set of all convex combinations of the points in X is called the *convex hull* of X.

Another way of thinking about the convex hull of X is as the smallest convex set containing all points in X. For an illustration of the convex hull, see Figure 3.5.

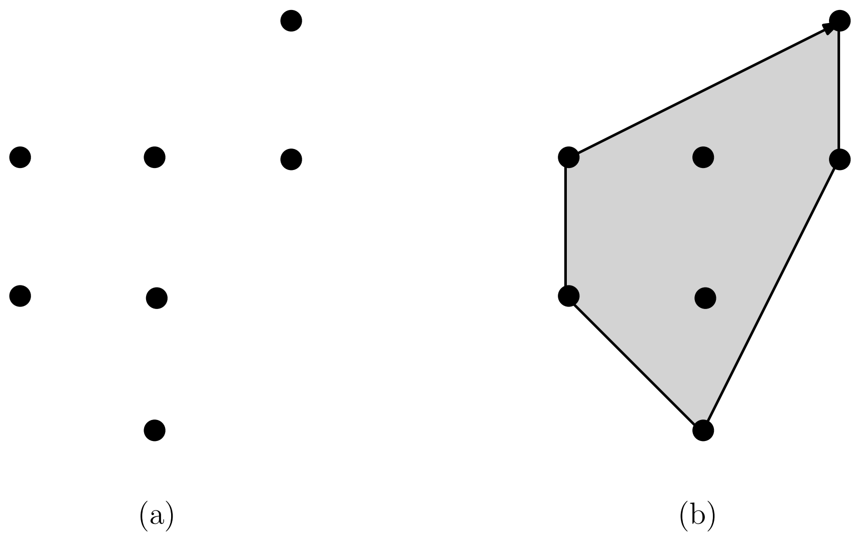


Figure 3.5: (a) A set of points. (b) The convex hull of the points.

Theorem 3.3. Every polytope is the convex hull of its extreme points. If X is a finite set of points, then the convex hull of X is a polytope P. The set of vertices of P is a subset of the points in X.

A vector $\boldsymbol{r} \in \mathbb{R}^n$ is a *ray* of the polyhedron $P \subset \mathbb{R}^n$ if and only if for any point $\bar{\boldsymbol{x}} \in P$, the set of vectors $\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x} = \bar{\boldsymbol{x}} + \lambda \boldsymbol{r} \text{ for } \lambda \geq 0\}$ also belongs to P. Informally, a ray of P is a "direction" that we can follow infinitely far, and still be within P. Another more formal definition is as follows:

Definition 3.8. Let $P^0 = \{\boldsymbol{r} \in \mathbb{R}^n \mid A\boldsymbol{r} \leq \boldsymbol{0}\}$. If $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{b}\} \neq \emptyset$, then $\boldsymbol{r} \in P^0 \setminus \{\boldsymbol{0}\}$ is called a *ray* of P.

Definition 3.9. A ray $\boldsymbol{r} \in \mathbb{R}^n$ of P is an *extreme ray* of P if $\boldsymbol{r}$ cannot be expressed as a convex combination of other rays of P.

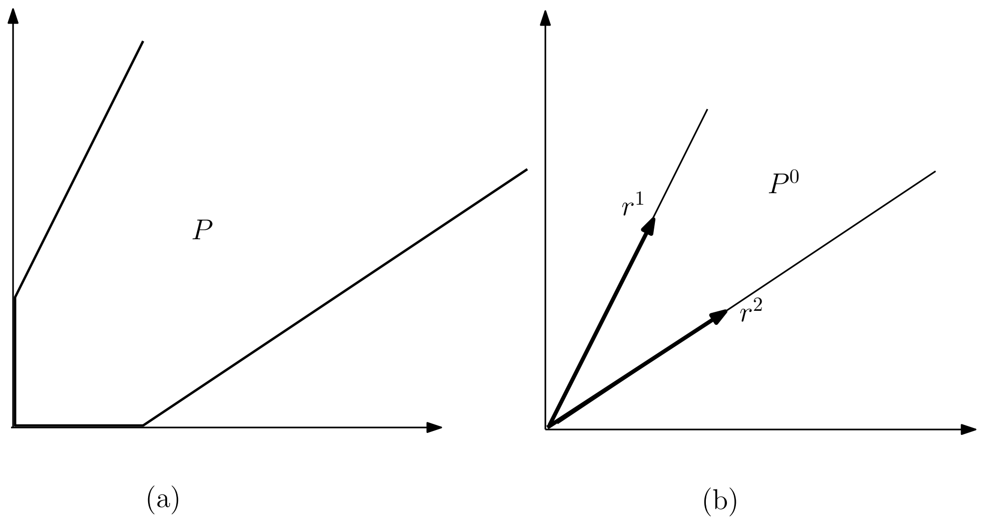


Figure 3.6: (a): A polyhedron P, (b): the corresponding P^0, and the two extreme rays, $\boldsymbol{r}^1$ and $\boldsymbol{r}^2$ of P.

An example of a polyhedron P, extreme rays of P, and the associated set P^0 are illustrated in Figure 3.6. Each polyhedron can be written as the sum of the convex combination of its extreme points and a nonnegative linear combination of its extreme rays, as stated in Minkowski's theorem below.

Theorem 3.4 (Minkowski)**.** Given is a polyhedron $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$. Assume that $P \neq \emptyset$ and that $\operatorname{rank}(A) = n$. Let $\{\boldsymbol{x}^k\}_{k\in K}$ and $\{\boldsymbol{r}^l\}_{l\in L}$ be the set of extreme points and extreme rays of P respectively. Then, we can write P as

$$\begin{aligned} P = \; & \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x} = \textstyle\sum_{k\in K} \lambda_k \boldsymbol{x}^k + \sum_{l\in L} \mu_l \boldsymbol{r}^l\,, \\ & \textstyle\sum_{k\in K} \lambda_k = 1,\ \lambda_k \geq 0,\ \text{for all } k \in K,\ \mu_l \geq 0,\ \text{for all } l \in L\}\,. \end{aligned}$$

3.2 Dimension, faces, and facets

Here we will introduce concepts of independence, and define dimension of a polyhedron and a face of a polyhedron. Even though we have defined convex combination and convex set in the previous subsection, it is repeated here for comparison with the new concepts.

If $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k \in \mathbb{R}^n$, and $\lambda_1, \ldots, \lambda_k$ are real scalars, then a vector $\boldsymbol{x} \in \mathbb{R}^n$ is called a *linear combination* of $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k$ if

$$\boldsymbol{x} = \sum_{i=1}^{k} \lambda_i \boldsymbol{x}^i\,;$$

an *affine combination* if

$$\boldsymbol{x} = \sum_{i=1}^{k} \lambda_i \boldsymbol{x}^i, \quad \sum_{i=1}^{k} \lambda_i = 1\,;$$

and a *convex combination* if

$$\boldsymbol{x} = \sum_{i=1}^{k} \lambda_i \boldsymbol{x}^i, \quad \sum_{i=1}^{k} \lambda_i = 1, \quad \lambda_i \geq 0 \text{ for all } i = 1, \ldots, k\,.$$

A linear subspace (affine subspace, convex set) is a set S closed under linear (affine, convex) combinations of finitely many vectors in S. For an arbitrary set S:

- lin $S = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x}$ is a linear combination of finitely many elements in $S\}$.
- aff $S = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x}$ is an affine combination of finitely many elements in $S\}$.
- conv $S = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x}$ is a convex combination of finitely many elements in $S\}$.

The sets lin S, aff S, conv S are called the linear, affine, and convex hull of S, respectively.

By convention, lin $\emptyset = \{0\}$, aff $\emptyset =$ conv $\emptyset = \emptyset$. Clearly, S is a linear subspace (affine subspace, convex set) if and only if $S =$ lin S ($S =$ aff S, $S =$ conv S). It can be shown that lin S (aff S, conv S) is the smallest linear space (affine space, convex set) containing S, i.e.,

$$\text{lin } S = \bigcap_{S \subseteq T \subseteq \mathbb{R}^n,\ T \text{ linear space}} T\,,$$

$$\text{aff } S = \bigcap_{S \subseteq T \subseteq \mathbb{R}^n,\ T \text{ affine space}} T\,,$$

$$\text{conv } S = \bigcap_{S \subseteq T \subseteq \mathbb{R}^n,\ T \text{ convex set}} T\,.$$

Example 3.3. $S = \{(0,2),\ (1,1)\}$.
We obtain the following sets: lin $S = \mathbb{R}^2$, aff $S = \{(x_1,x_2) \mid x_1 + x_2 = 2\}$,
conv $S = \{(x_1,x_2) \mid x_1 + x_2 = 2,\ 0 \le x_1 \le 1\}$. See also Figure 3.7.

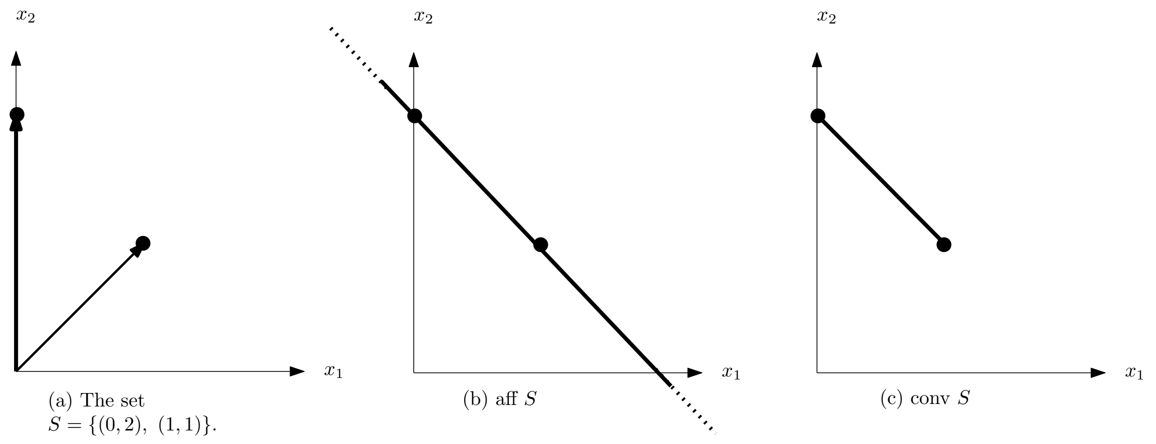


Figure 3.7: The sets S, aff S, and conv S.

□

A subset $S \subseteq \mathbb{R}^n$ is *linearly independent* if for all finite subsets $\{\boldsymbol{x}^1, \dots, \boldsymbol{x}^k\} \subseteq S$

$$\sum_{i=1}^{k} \lambda_i \boldsymbol{x}^i = 0 \text{ implies } \lambda_i = 0 \text{ for all } i = 1, \dots, k\,;$$

and *affinely independent* if

$$\sum_{i=1}^{k} \lambda_i \boldsymbol{x}^i = 0,\ \sum_{i=1}^{k} \lambda_i = 0 \text{ implies } \lambda_i = 0 \text{ for all } i = 1, \dots, k\,.$$

The following lemmas show the similarities, and contrast the differences, between linear and affine independence.

Lemma 3.1. Let $X = \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k\} \subseteq \mathbb{R}^n$. The following are equivalent:

(a) X is linearly independent.

(b) No vector of X can be expressed as a linear combination of the other vectors in X.

(c) Each vector $\boldsymbol{y} \in \text{lin } X$ can be expressed as a linear combination of the vectors in X is a unique way, i.e., there are scalars $\lambda_1, \ldots, \lambda_k$ such that $\boldsymbol{y} = \sum_{i=1}^k \lambda_i \boldsymbol{x}^i$, and if there are scalars $\mu_1, \ldots, \mu_k$ with $\boldsymbol{y} = \sum_{i=1}^k \mu_i \boldsymbol{x}^i$, then $\mu_i = \lambda_i$ for all i.

Lemma 3.2. Let $X = \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k\} \subseteq \mathbb{R}^n$. The following are equivalent:

(a) X is affinely independent.

(b) No vector of X can be expressed as an affine combination of the other vectors in X.

(c) Each vector $\boldsymbol{y} \in \text{aff } X$ can be expressed as an affine combination of the vectors in X is a unique way.

(d) The set $\{\boldsymbol{x}^2 - \boldsymbol{x}^1,\ \boldsymbol{x}^3 - \boldsymbol{x}^1,\ \boldsymbol{x}^k - \boldsymbol{x}^1\}$ is linearly independent.

Suppose we have linearly independent vectors $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n \in \mathbb{R}^n$. Then $\{\boldsymbol{0}, \boldsymbol{x}^1, \ldots, \boldsymbol{x}^n\}$ is linearly *dependent*, but affinely *independent*. All sets $\{\boldsymbol{x}\}$ with $\boldsymbol{x} \neq \boldsymbol{0}$ are both linearly and affinely independent, $\{\boldsymbol{0}\}$ is linearly dependent but affinely independent (verify this!). By convention $\emptyset$ is linearly *and* affinely independent.

The *linear dimension* of a set $S \subseteq \mathbb{R}^n$, lin.dim S, is the cardinality of the largest linearly independent subset of S. The *affine dimension*, aff.dim S, is the cardinality of the largest affinely independent subset of S. The *dimension* of S, dim S, is defined by

$$\dim S = \text{aff.dim } S - 1\,.$$

A set $S \subseteq \mathbb{R}^n$ is called *full-dimensional* if dim $S = n$. Our definition of dimension coincides with our geometric intuition, i.e., points have dimension 0, lines or line segments have dimension 1, and so on. Note that dim $\mathbb{R}^n$ = lin.dim $\mathbb{R}^n = n$, while aff.dim $\mathbb{R}^n = n + 1$. The following lemma establishes the relation between aff.dim S and lin.dim S.

Lemma 3.3. For any $S \subseteq \mathbb{R}^n$

(a) if $\boldsymbol{0} \in \text{aff } S$, then aff.dim S = lin.dim $S + 1$,

(b) if $\boldsymbol{0} \notin \text{aff } S$, then aff.dim S = lin.dim S.

Example 3.4. Consider the polytope P in Figure 3.8. Determine the dimension of P.

We can exhibit three affinely independent points in P:

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \end{pmatrix},$$

which indicates that aff.dim $P \geq 3$. Since the polytope $P \subset \mathbb{R}^2$ we know that aff.dim $P \leq 3$ and we can conclude that aff.dim $P = 3$. Since dim P = aff.dim $P - 1$ we have

$$\dim P = 2\,.$$

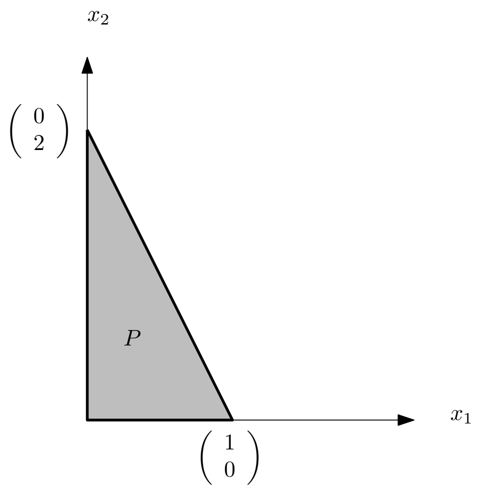


Figure 3.8: A full-dimensional polytope.

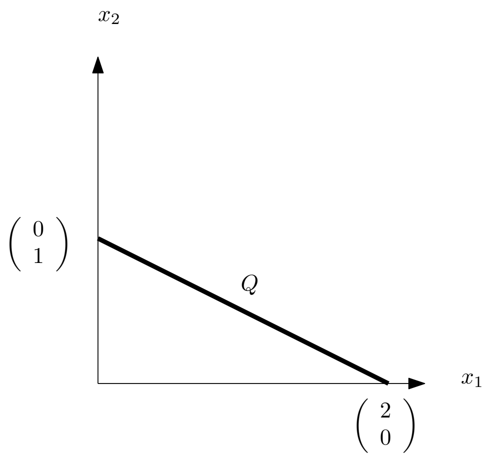


Figure 3.9: A polytope that consists of a line segment.

Next, consider the polytope $Q = \{\boldsymbol{x} \in \mathbb{R}^2 \mid x_1 + 2x_2 = 2,\ \boldsymbol{x} \geq 0\}$, that consists of the line segment illustrated in Figure 3.9. Since $Q \subset \mathbb{R}^2$ and since there is an equation in the description of Q, we know that $\dim Q \leq 1$.

Since $\mathbf{0} \notin \text{aff } Q$, $\text{aff.dim } Q = \text{lin.dim } Q$. We can exhibit two linearly independent points:

$$\begin{pmatrix} 2 \\ 0 \end{pmatrix}, \quad \begin{pmatrix} 0 \\ 1 \end{pmatrix},$$

which shows that $\text{aff.dim } Q \geq 2$. Using $\dim Q = \text{aff.dim } Q - 1$ we obtain $\dim Q \geq 1$, and combining with $\dim Q \leq 1$ gives as result

$$\dim Q = 1\,.$$

□

Definition 3.10. An inequality $\boldsymbol{\alpha x} \leq \beta$ is *valid* for a polyhedron P, if $\boldsymbol{\alpha x} \leq \beta$ for all $\boldsymbol{x} \in P$.

Definition 3.11. Assume that $\boldsymbol{\alpha x} \leq \beta$ is valid for P. The set $F = \{\boldsymbol{x} \in P \mid \boldsymbol{\alpha x} = \beta\}$ is a *face* of P. F is a proper face of P if $F \neq \emptyset$ and $F \neq P$.

Definition 3.12. A face F of P is a *facet* of P if dim $F =$ dim $P - 1$.

For an illustration of an improper face, a face and a facet, see Figure 3.10.

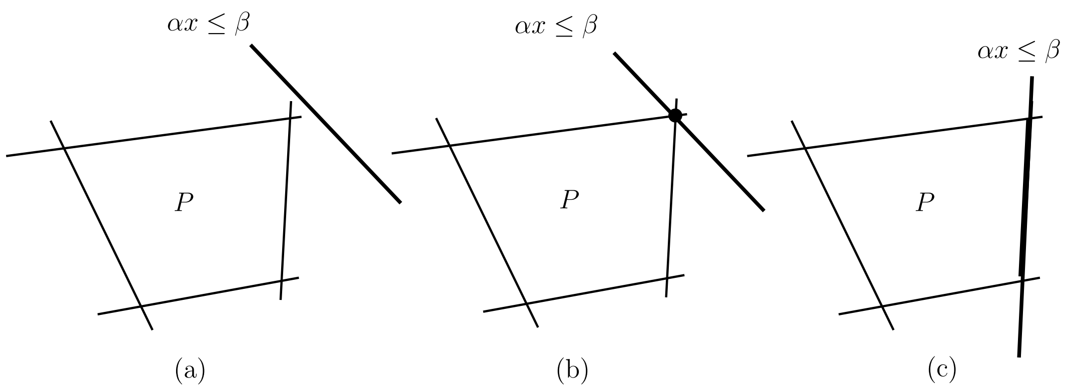


Figure 3.10: A polytope P and a valid inequality $\boldsymbol{\alpha x} \le \beta$. (a): The valid inequality induces an improper face as $F = \emptyset$. (b): The valid inequality induces a zero-dimensional face. (c): The valid inequality induces a facet.

Proposition 3.1. The point $\boldsymbol{x}$ is an extreme point of P if and only if $\boldsymbol{x}$ is a zero-dimensional face of P. The ray $\boldsymbol{r}$ is an extreme ray of P if and only if it is a one-dimensional face of P^0.

3.3 Exercises

Exercise 3.1. Show that constraints on the form $\boldsymbol{ax} = b$ and $\boldsymbol{ax} \ge b$ can be written in the form $\boldsymbol{ax} \le b$.

Exercise 3.2. Prove the following. If $A, B \subseteq \mathbb{R}^n$ are convex sets, then $A \cap B$ is also a convex set.

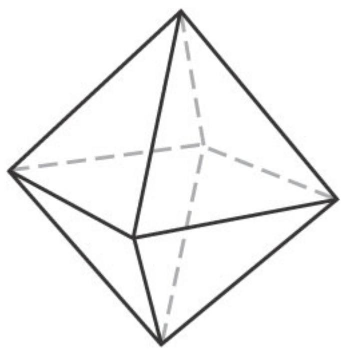
Exercise 3.3. Prove Observation 3.1.

Exercise 3.4. Prove that the collection of optimal solutions of an LP forms a convex set.

Exercise 3.5. The n-dimensional *orthoplex* can be defined as

$$O_n = \text{conv}\,\{e_1, -e_1, e_2, -e_2, \ldots, e_n, -e_n\}$$

with $e_1, \ldots, e_n$ the standard basic vectors. For example, the 3-dimensional orthoplex is the octahedron, see the figure below. Describe O_n as a polyhedron $\{x \in \mathbb{R}^n \mid Ax \le b\}$.



Chapter 4

The Simplex Method

Given a linear programming problem (LP), precisely one of the following three situations occur:

1. The problem is *feasible*, and the optimal objective function value is *bounded*.

2. The problem is *feasible*, and the optimal objective function value is *unbounded*.

3. The problem is *infeasible*, i.e., the associated polyhedron is the empty set.

In this chapter, we describe the Simplex method [5] for solving LPs. By “solving”, we mean that we find an *optimal solution*, which is a feasible solution with largest possible objective function value if the problem is a maximization problem or with smallest possible objective function value if the problem is a minimization problem, if such an optimal solution exists. If no optimal solution exists, we report either that the problem is *infeasible* (no feasible solution exists) or *unbounded* (there exist feasible solutions with arbitrarily large/small objective function values). It is also possible that there exist multiple optimal solutions, in which case we may either be happy with finding a single optimal solution, or want to find all optimal solutions.

In Chapter 3, Theorem 3.2, we saw that if we minimize or maximize a linear function over a polyhedron, a minimum, or maximum will be attained in an extreme point of the polyhedron if the optimum is bounded. We may have infinitely many feasible solutions to a linear optimization problem, but only finitely many extreme points. Notice that “finitely many” may still be a huge number.

If a given problem has feasible solutions, then the Simplex method systematically checks extreme points of the associated polyhedron in order to find a best one, or give us a certificate that proves that the objective value is unbounded. A best extreme point is an optimal solution to the problem.

For now, we assume that we are dealing with a *feasible problem* with *bounded* objective value. We will return to case 2 in Section 4.3, where we also look at problems with multiple optimal solutions. Case 3 will be discussed, among other things, in Section 4.4.

Before we look at how the Simplex method works we need to answer three questions:

1. How do we represent an extreme point algebraically?

2. How do we know whether an extreme point is optimal?

3. If an extreme point is not optimal, how can we move to another extreme point?

4.1 Representing an extreme point algebraically

We start by a small example.

Example 4.1.

$$\begin{array}{llrcrcrl}
\max & z = & 50x_1 & + & 45x_2 & & & \\
\text{s.t.} & & 6x_1 & + & 5x_2 & \leq & 60 & (1) \\
 & & x_1 & + & 2x_2 & \leq & 15 & (2) \\
 & & x_1 & & & \leq & 8 & (3) \\
 & & \multicolumn{3}{l}{x_1,\ x_2 \geq 0} & & &
\end{array}$$

We can study this example graphically: The set of feasible solutions forms a polytope, and has

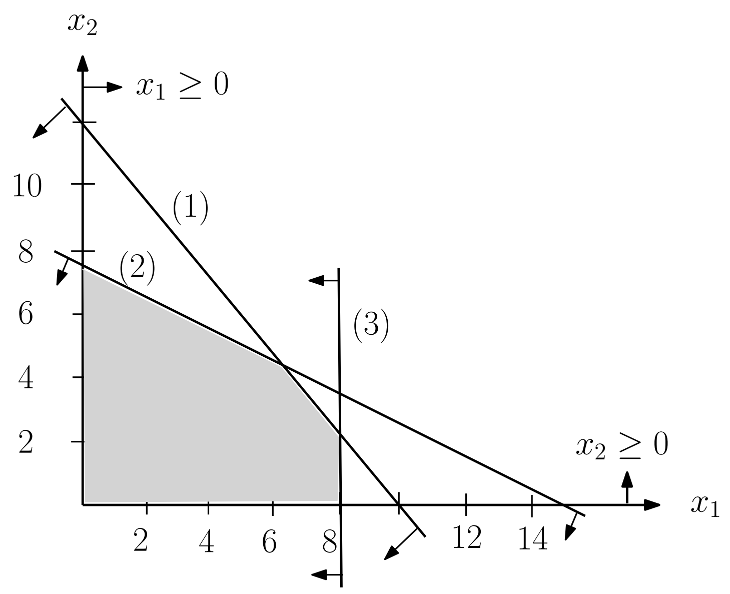


Figure 4.1: The set of feasible solutions to the example.

five extreme points:

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 8 \\ 0 \end{pmatrix}, \begin{pmatrix} 8 \\ 2\frac{2}{5} \end{pmatrix}, \begin{pmatrix} 6\frac{3}{7} \\ 4\frac{2}{7} \end{pmatrix}, \begin{pmatrix} 0 \\ 7\frac{1}{2} \end{pmatrix}.$$

Each of the extreme points is obtained by taking the intersection between two of the lines defining the associated constraints. In general, an extreme point is defined as an intersection of hyperplanes. Consider for instance the extreme point $(8, 2\frac{2}{5})^\mathsf{T}$. This point is obtained by taking the intersection of the lines defining Constraints (1) and (3):

$$\begin{array}{rcrcr}
6x_1 & + & 5x_2 & = & 60 \\
x_1 & & & = & 8
\end{array}$$

□

The first step we make in order to be able to represent extreme points, is to put the constraints in "equality form". When we have a constraint in the form $\boldsymbol{a}_i\boldsymbol{x} \leq b_i$, we add a so-called *slack variable* $s_i \geq 0$ to the left-hand side of the constraint. For any feasible solution $\bar{\boldsymbol{x}}$, the value of s_i tells us precisely the difference between the right-hand side b_i and the left-hand side $\boldsymbol{a}_i\bar{\boldsymbol{x}}$. Notice that for all feasible solutions, this difference should be nonnegative!

Depending on which extreme point we consider, some slack variables will take value zero, namely the slack variables that are associated with the binding constraints in that point. Slack variables associated with non-binding constraints will take a positive value. We will observe that each extreme point has an algebraic counterpart, and that there is a correspondence between these representations.

Example 4.1, cont.
We put all constraints in equality form by adding one slack variable for each constraint:

$$\begin{array}{rcrcrcrcrcl} 6x_1 & + & 5x_2 & + & s_1 & & & & & = & 60 \quad (1') \\ x_1 & + & 2x_2 & & & + & s_2 & & & = & 15 \quad (2') \\ x_1 & & & & & & & + & s_3 & = & 8 \quad (3') \\ & & & & \multicolumn{5}{c}{x_1,\ x_2,\ s_1,\ s_2,\ s_3 \geq 0} & & \end{array}$$

If we again consider the extreme point $(8, 2\frac{2}{5})^\mathsf{T}$, we see that to obtain this point we set s_1 and s_3 equal to zero and solve for the remaining variables. These slack variables are equal to zero since they are associated with the binding constraints. The value of slack variable s_2 is equal to $15 - 8 - 2(2\frac{2}{5}) = 2\frac{1}{5}$.

The extreme point $(0, 0)^\mathsf{T}$ (the origin) is obtained by setting $x_1 = x_2 = 0$ and we can easily read out the values of s_1, s_2 and s_3, *since each of these variables occur in precisely one constraint with a coefficient* $+1$. We leave it to the reader to compute the other extreme points by setting two variables equal to zero and solve for the remaining variables.

This way of computing extreme points algebraically will be used by the Simplex method. Finally, we observe that this approach works, since we have three linearly independent columns in our system of constraints. □

Suppose we have a system of m constraints in equality form and n variables, $A\boldsymbol{x} = \boldsymbol{b}$.

Assumption 4.1. There are m linearly independent columns $\boldsymbol{A}_j$ of A.

As we say in our example, the fact that we added one slack variable for each constraint ensures that Assumption 4.1 holds. For the purpose of linear optimization, Assumption 4.1 is non-restrictive.

Definition 4.1. A set of constraints $F = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}\}$ to a linear optimization problem is said to be in *standard form* if the following holds:

1. Each constraint is written in equality form
2. Each element b_i of the right-hand side vector $\boldsymbol{b}$ is nonnegative.
3. All variables are restricted to be nonnegative.

Definition 4.2. Given is an $m \times n$ matrix A. A *basis* B of A is an $m \times m$ non-singular submatrix of A, i.e., B contains m linearly independent columns of A, $B = [\boldsymbol{A}_{j_1} \cdots \boldsymbol{A}_{j_m}]$.

Definition 4.3. Given a system of constraints in standard form, $A\boldsymbol{x} = \boldsymbol{b}$ and a basis B of A, a *basic solution* is obtained by setting the variables corresponding to the non-basic columns to zero, and solve for the remaining variables. The variables corresponding to the columns in the basis are called *basic variables* and the variables that we set to zero are called *non-basic variables*.

Example 4.1, cont.
Set the variables x_2 and s_2 equal to zero and solve for the remaining variables. This gives the basic solution $x_1 = 15,\ s_1 = -30,\ s_3 = -7,\ x_2 = s_2 = 0$. Since s_1 and s_3 have negative values, this basic solution is infeasible .

If we put $x_1 = s_2 = 0$ we obtain the basic solution $x_2 = 7\frac{1}{2},\ s_1 = 15,\ s_3 = 8,\ x_1 = s_2 = 0$. In this solution, all variables take nonnegative values, so this solution is feasible. □

Definition 4.4. If, in a basic solution, all variables take nonnegative values, we call the solution a *basic feasible solution* (bfs).

Theorem 4.1. Given a linear programming problem with the constraints in standard form, $F = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}\}$, the following statements hold.

(i) If there exists a vector $\boldsymbol{x} \in F$, then there exists a basic feasible solution $\bar{\boldsymbol{x}} \in F$.

(ii) If there exists a finite optimal solution $\boldsymbol{x} \in F$, then there exists an optimal basic feasible solution $\boldsymbol{x}^* \in F$.

For a proof, we refer to the book by Padberg [8].

The way the Simplex method represents a bfs, will make it trivial to determine the value of the basic variables.

Example 4.1, cont.
Consider again equations $(1') - (3')$ and recall that the extreme point $(0,0)^\mathsf{T}$ gives a particularly easy way of determining the value of the basic variables, $s_1 = 60,\ s_2 = 15,\ s_3 = 8$, since each of the basic variables only occur, with a coefficient $+1$, in precisely one row. Since all right-hand side values are nonnegative, we can immediately conclude that the basic solution is indeed feasible. □

So far we have ignored the objective function $z = \boldsymbol{c}^\mathsf{T}\boldsymbol{x}$, but it will of course have to be added to our formulation. In our presentation we choose to write the objective row as follows:

$$-z + \boldsymbol{c}^\mathsf{T}\boldsymbol{x} = -\bar{z}\,, \tag{4.1}$$

where $\bar{z}$ is the value of the objective function evaluated in the solution $\boldsymbol{x}$. We could also have chosen to write $z - \boldsymbol{c}^\mathsf{T}\boldsymbol{x} = \bar{z}$, but as we will see in the following section, intuitively it may be an advantage to work with the original signs of the objective coefficients $\boldsymbol{c}$, which in (4.1) is the case.

In each iteration of the Simplex method we will have a *system of equations*, which includes the objective function in the form given in (4.1), in which the following holds:

(R1) The system of constraints is in standard form.

(R2) In each constraint row, there is a variable with coefficient $+1$. In all other rows, including the objective row, this variable has coefficient 0.

Example 4.1, cont.
Our problem, written in the following form, satisfies requirements R1 and R2 given above.

$$\begin{array}{rcrcrcrcrcrcrl}
-z & + & 50x_1 & + & 45x_2 & & & & & & & = & 0 & (0') \\
 & & 6x_1 & + & 5x_2 & + & s_1 & & & & & = & 60 & (1') \\
 & & x_1 & + & 2x_2 & & & + & s_2 & & & = & 15 & (2') \\
 & & x_1 & & & & & & & + & s_3 & = & 8 & (3') \\
 & & & & \multicolumn{7}{c}{x_1,\ x_2,\ s_1,\ s_2,\ s_3 \geq 0} & & &
\end{array}$$

From this representation we can quickly read the bfs $(x_1, x_2, s_1, s_2, s_3)^\mathsf{T} = (0, 0, 60, 15, 8)$ and its objective value 0. □

Before we start explaining the Simplex method in more detail, we formalize the relation between the feasible region F of an LP in standard form:

$$F = \{x \in \mathbb{R}^n \mid A\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}\} \tag{4.2}$$

and a polyhedron P.

Since rank$(A) = m$ by Assumption 4.1, we can write the equations in F (4.2) in the following form, possibly after row operations:

$$x_{n-m+i} = \bar{b}_i - \sum_{j=1}^{n-m} \bar{a}_{ij} x_j, \quad i = 1, \ldots, m\,.$$

Here we use the notation $\bar{b}_i$, $\bar{a}_{ij}$ to indicate that the coefficients are possibly modified by the row operations. Since $x_j \geq 0$ for all $j = 1, \ldots, n$ in F (4.2), we now obtain

$$\begin{aligned} x_j &\geq 0, \quad j = 1, \ldots, n-m\,, \\ \bar{b}_i - \sum_{j=1}^{n-m} \bar{a}_{ij} x_j &\geq 0, \quad i = 1, \ldots, m\,. \end{aligned}$$

This set of n constraints defines a polyhedron in $\mathbb{R}^{n-m}$.

Next, we start with a polyhedron $P \in \mathbb{R}^{n-m}$ defined by n halfspaces. These halfspaces can be expressed as follows:

$$d_{i1}x_1 + d_{i2}x_2 + \cdots + d_{i,n-m}x_{n-m} \leq g_i, \quad i = 1, \ldots, n\,. \tag{4.3}$$

Without loss of generality, we can assume that $n - m$ of the halfspaces are nonnegativity constraints, and we can assume that they are the first $n - m$ constraints, i.e., $x_i \geq 0,\ i = 1, \ldots, n - m$.

Let D be the constraint matrix for the remaining constraints. We can now introduce slack variables $x_{n-m+1}, \ldots, x_n$ in (4.3) to obtain

$$\begin{aligned} A\boldsymbol{x} &= \boldsymbol{b}\,, \\ \boldsymbol{x} &\geq \boldsymbol{0}\,, \end{aligned}$$

where $A = (D, I)$ and $b_i = g_i$ for $i = n - m + 1, \ldots, n$.

So, there is a translation between a polyhedron P and a set constraints F in standard form. Moreover, any point $\hat{\boldsymbol{x}} = (x_1, \ldots, x_{n-m}) \in P$ can be transformed to a point $x = (x_1, \ldots, x_n) \in F$ by defining

$$x_i = g_i - \sum_{j=1}^{n-m} d_{ij} x_j, \quad i = n - m + 1, \ldots, n\,. \tag{4.4}$$

Any point $\boldsymbol{x} = (x_1, \ldots, x_n) \in F$ can be transformed to $\hat{\boldsymbol{x}} = (x_1, \ldots, x_{n-m}) \in P$ by dropping the last coordinates of $\boldsymbol{x}$.

Theorem 4.2. The point $\hat{\boldsymbol{x}} \in P$ is an extreme point (vertex) of P if and only if the corresponding vector $\boldsymbol{x} \in F$, as defined in (4.4), is a bfs of F.

It is important to notice that an extreme point of a polyhedron P can correspond to multiple basic feasible solutions. Recall that a bfs is obtained by setting $n - m$ variables equal to zero and then solve for the remaining m variables. If one or more basic variables take value zero in a bfs, one could interchange them in the basis by equally many non-basic variables and obtain exactly the same extreme point of the polyhedron. We illustrate this in Example 4.2 below.

Definition 4.5. A basic solution is called *degenerate* if one or more basic variables take value zero.

Example 4.2. Consider the polytope P defined by following constraints:

$$\begin{array}{ccccc} x_1 & & & \leq & 1 \\ & & x_2 & \leq & 1 \\ x_1 & + & x_2 & \leq & 2 \\ x_1, & & x_2 & \geq & 0 \end{array}$$

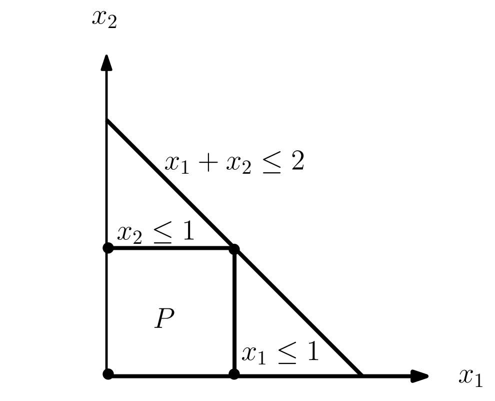


Figure 4.2: The polyhedron corresponding to the constraints of Example 4.2.

The polytope P and its four extreme points are illustrated in Figure 4.2.
If we introduce slack variables, we obtain the set F of feasible solutions in $\mathbb{R}^5$:

$$\begin{array}{ccccccccccc} x_1 & & & + & s_1 & & & & & = & 1 \\ & & x_2 & & & + & s_2 & & & = & 1 \\ x_1 & + & x_2 & & & & & + & s_3 & = & 2 \\ x_1, & & x_2, & & s_1, & & s_2, & & s_3 & \geq & 0 \end{array}$$

Setting $s_1 = s_2 = 0$ and solving for the rest yields the bfs:
basic variables: $x_1 = x_2 = 1,\ s_3 = 0$
non-basic variables: $s_1 = s_2 = 0$,
corresponding to the extreme point $(1, 1)^{\mathsf{T}}$. This extreme point, however, also corresponds to the basic feasible solutions that are obtained by setting any two of the three slack variables to zero. □

Degeneracy is something that can cause the Simplex method to get stuck by cycling among degenerate basic feasible solutions corresponding to a certain extreme point of the feasible set. In Subsection 4.5 we briefly mention how cycling can be avoided.

4.2 Checking optimality and moving to another basic feasible solution

Informally, the Simplex method checks in each iteration whether the current bfs is optimal, and if not, it moves to a "neighboring" bfs that has an objective value that is at least as good as the current bfs. By "at least as good", we mean a value that is a least as large, if our problem is a maximization problem, or at least as small, if we consider a minimization problem. By "neighboring" bfs we mean a new bfs in which one basic variable is made non-basic, and one of the current non-basic variables is made basic.

Example 4.1, cont.
Consider the bfs $(0, 0, 60, 15, 8)^{\mathsf{T}}$ given in the previous section. The objective value of that solution is $z = 0$. We recall that our problem is a maximization problem. Each unit of increase in the value of variable x_1 gives in increase in the objective value of 50, and a unit of increase in the value of variable x_2 gives an increase of 45, so it makes sense to increase the value of any of these variables.

Let us start with x_1. An important question is: how much can we increase x_1 before we create an infeasible solution? If we just look at Figure 4.1 we see that we can increase x_1 up to the value of 8 and still remain feasible. Suppose we do that, then Constraint (3) will be binding and the associated slack variable, which currently takes value 8, will take value 0. We will now preform this step algebraically by considering the current solution and letting the value of x_1 increase while maintaining feasibility. Notice that the value of x_2 remains zero. This gives rise to the following:

$$\begin{array}{rclcrcl} s_1 & = & 60 - 6x_1 & \geq 0 \Rightarrow & x_1 & \leq & 10, \\ s_2 & = & 15 - x_1 & \geq 0 \Rightarrow & x_1 & \leq & 15, \\ s_3 & = & 8 - x_1 & \geq 0 \Rightarrow & x_1 & \leq & 8\,. \end{array}$$

Since the last of the three inequalities gives the smallest upper bound on the increase of x_1, this bound is also going to determine the new value of x_1, i.e., $x_1 = 8$, which implies $s_3 = 0$. This is exactly what we observed graphically. The value of the new solution is $z = 50x_1 = 50 \cdot 8 = 400$.

We now want to express the new solution in the same form as (0′)–(3′), i.e., being in standard form and satisfying requirements **(R1)** and **(R2)**. This is easily done by performing row operations on (0′)–(3′). The row operations that have been applied to (0′)–(3′) to get the new system (0″)–(3″) below are given next to the new equations.

$$\begin{array}{rrrrrcrl} -z & + 45x_2 & & & - 50s_3 & = & -400 & (0''):\ (0') - 50\cdot(3') \\ & 5x_2 & + s_1 & & - 6s_3 & = & 12 & (1''):\ (1') - 6\cdot(3') \\ & 2x_2 & & + s_2 & - s_3 & = & 7 & (2''):\ (2') - (3') \\ x_1 & & & & + s_3 & = & 8 & (3''):\ (3') \\ & \multicolumn{4}{l}{x_1,\ x_2,\ s_1,\ s_2,\ s_3 \geq 0.} & & & \end{array}$$

The new bfs is $(x_1, x_2, s_1, s_2, s_3)^\mathsf{T} = (8, 0, 12, 7, 0)$ with objective value $z = 400$. We notice that the new basic variables are x_1, s_1, s_2 and the non-basic variables are x_2, s_3, so the new basis differs from the previous one by one element. We also notice that the objective value increased from 0 to 400.

We summarize the steps we just did to go from one bfs to a new bfs with better objective value below.

1. Since we have a maximization problem and there are variables in the objective function with positive coefficient, we choose the one with largest coefficient, x_1, and increase its value. Variable x_1 then becomes a basic variable, and we call x_1 the *entering basic variable*.

2. We check how much we can increase the value of x_1. This is determined by which of the current basic variables takes value zero first. Notice that this is determined by checking, for each of the constraints, the ratio between the current right-hand side value and the constraint coefficient of the entering basic variable. The minimum ratio then determines which variable takes zero first. That is, we compute $\min\{60/6,\ 15/1,\ 8/1\} = 8$, which is referred to as the *min ratio test.* Since the smallest ratio occurs in the third constraint, the variable that was basic for that constraint leaves the basis. In our case it was variable s_3, which is then *leaving basic variable*.

3. Let $\bar{a}_{ij}$ denote the current values of the constraint coefficients. Once we know the entering variable, x_1, and the leaving variable, s_3 that was basic variable in the third row, we know our so-called *pivot element*, $\bar{a}_{13}$. Perform row operations such that the coefficient $\bar{a}_{13}$ takes value $+1$, and such that all other coefficient in column 1 take value 0. Transitioning between two basic feasible solutions is referred to as a *pivot.*

Is the current bfs optimal? The answer is ‘no’, since the current objective coefficient of x_2 is positive, so increasing the value of x_2 will give us a solution with better objective value. We perform the steps described above once more to move to a new bfs. Variable x_2 is the entering basic variable. From our current system of equations we determine the leaving basic variable:

$$\begin{array}{rclcrcl} s_1 & = & 12 - 5x_2 & \geq 0 \Rightarrow & x_2 & \leq & \frac{12}{5} = 2.4, \\ s_2 & = & 7 - 2x_2 & \geq 0 \Rightarrow & x_2 & \leq & \frac{7}{2} = 3.5\,. \end{array}$$

Variable s_1 will reach value 0 first, when x_2 increases to the value 2.4, so s_1 is the leaving basic variable. Notice that we cannot perform the ratio test for the third constraint, since the constraint coefficient $\bar{a}_{32} = 0$.

After performing row operations on $(0'')$–$(3'')$, we obtain the following system:

$$\begin{array}{rrrrrrrrrrrl}
-z & & & - & 9s_1 & & & + & 4s_3 & = & -508 & (0''') : \ (0'') - 9 \cdot (1'') \\
 & & x_2 & + & \frac{1}{5}s_1 & & & - & \frac{6}{5}s_3 & = & \frac{12}{5} & (1''') : \ \frac{1}{5} \cdot (1'') \\
 & & & - & \frac{2}{5}s_1 & + & s_2 & + & \frac{7}{5}s_3 & = & \frac{11}{5} & (2''') : \ (2'') - \frac{2}{5} \cdot (1'') \\
 & x_1 & & & & & & + & s_3 & = & 8 & (3''') : \ (3'') \\
 & & \multicolumn{9}{l}{x_1, \ x_2, \ s_1, \ s_2, \ s_3 \geq 0.}
\end{array}$$

The new bfs is $(x_1, x_2, s_1, s_2, s_3)^\mathsf{T} = (8, 2.4, 0, 2.2, 0)$ with objective value $z = 508$. Since s_3 has a positive objective coefficient, we let s_3 enter the basis and determine which variable will leave.

$$\begin{array}{rcllcl}
x_2 & = & \frac{12}{5} + \frac{6}{5}s_3 & \geq 0 & \Rightarrow & \text{no upper bound on } s_3, \\
s_2 & = & \frac{11}{5} - \frac{7}{5}s_3 & \geq 0 & \Rightarrow & s_3 \ \leq \ \frac{11}{7} \\
x_1 & = & 8 - s_3 & \geq 0 & \Rightarrow & s_3 \ \leq \ 8\,.
\end{array}$$

Here we again see that performing a ratio test on the first constraint is meaningless, since the coefficient $\bar{a}_{1,s_3} < 0$, which means that no upper bound on the increase in the value of the entering variable can be obtained. We therefore only perform the min ratio test on *positive* $\bar{a}_{ij}$-coefficients.

Since variable s_2 reaches zero first, the increase in the value of s_3 is bounded by 11/7, and s_2 is the leaving basic variable. The new system of equations, with associated row operations, is given below:

$$\begin{array}{rrrrrrrrrrrl}
-z & & & - & 7\frac{6}{7}s_1 & - & \frac{20}{7}s_2 & & & = & -514\frac{2}{7} & (0''') - \frac{20}{7} \cdot (2''') \\
 & & x_2 & - & \frac{1}{7}s_1 & + & \frac{6}{7}s_2 & & & = & 4\frac{2}{7} & (1''') + \frac{6}{7} \cdot (2''') \\
 & & & - & \frac{2}{7}s_1 & + & \frac{5}{7}s_2 & + & s_3 & = & \frac{11}{7} & \frac{5}{7}(2''') \\
 & x_1 & & + & \frac{2}{7}s_1 & - & \frac{5}{7}s_2 & & & = & 6\frac{3}{7} & (3''') - \frac{5}{7}(2''') \\
 & & \multicolumn{9}{l}{x_1, \ x_2, \ s_1, \ s_2, \ s_3 \geq 0.}
\end{array}$$

Now we observe that all coefficients in the current objective row have nonpositive values, so we cannot increase any of the current non-basic variable in order to find a solution with a better objective value. Hence, the current bfs is optimal.

We write the optimal solution as $\boldsymbol{x}^*$ and the optimal objective function value as z^*. In the solution $\boldsymbol{x}^*$ we typically include only the original variables, and not the slack variables.

$$\begin{pmatrix} x_1^* \\ x_2^* \end{pmatrix} = \begin{pmatrix} 6\frac{3}{7} \\ 4\frac{2}{7} \end{pmatrix}, \quad z^* = 514\frac{2}{7}\,.$$

Graphically, we illustrate how we transitioned between the extreme points corresponding to the three basic feasible solutions in Figure 4.3.

In our description of the Simplex method we referred to the "current objective coefficients" when deciding whether or not a solution is optimal. The common name for "current objective coefficient" of variable x_j is the *reduced cost* of x_j, denoted by $\bar{c}_j$. If we have a maximization problem, we have reached an optimal solution if all reduced costs are nonpositive, i.e., $\bar{c}_j \leq 0$ for all j. In a minimization problem, we have reached optimum is all reduced costs are nonnegative, i.e., $\bar{c}_j \geq 0$ for all j. A minimization problem is considered in Example 4.3 in the following section.

□

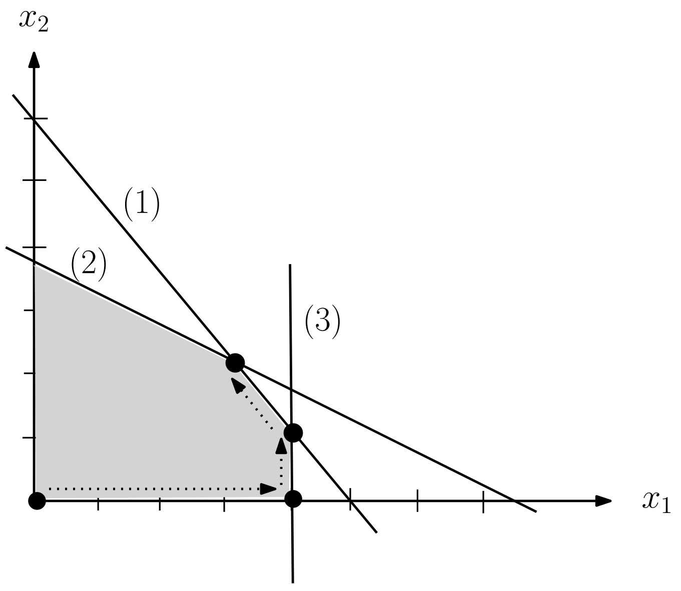


Figure 4.3: The four extreme points corresponding to the basic feasible solutions.

In Figure 4.3 we see that it would have taken one iteration less if we had started with x_2 as entering variable. This raises the question of efficiency of the Simplex method. In Chapter 15 we will discuss how to analyse algorithms, but as a remark here we can say that no-one has so far been able to develop a variant of the Simplex method that is always "efficient".

4.3 Problems with unbounded objective value or multiple optimal solutions

Example 4.1 that we considered in the previous section had a unique bounded optimum. Relevant questions are how the Simplex method detects that the optimal value is unbounded or that we have multiple optimal solutions.

4.3.1 Unbounded optimal objective value

We have seen that the min-ratio test determines the leaving basic variable, i.e., the current basic variable that takes value zero first when the value of the new basic variable is increased. In the case of an unbounded objective value, we can increase the value of an entering basic variable along an extreme ray of a polyhedron, and we therefore have no upper bound on the increase. In the Simplex method this is reflected in a failed min-ratio test, i.e., all constraint coefficients in the column of the entering basic variable take nonpositive values. We illustrate this in the following example.

Example 4.3.

$$\begin{array}{llrcrcrl} \min & z = & -x_1 & + & 2x_2 & & & \\ \text{s.t.} & & -x_1 & + & x_2 & \leq & 2 & \quad (1) \\ & & \multicolumn{5}{l}{x_1,\ x_2 \geq 0.} & \end{array}$$

Graphically, the problem looks as follows.

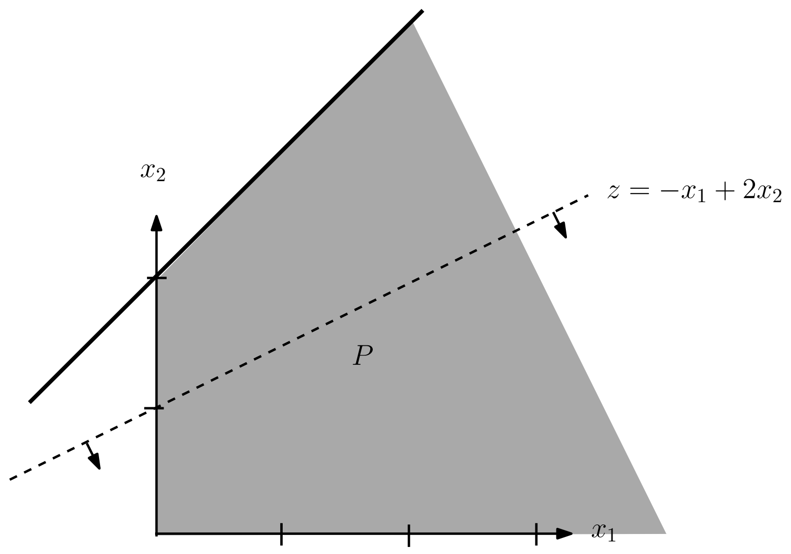


Figure 4.4: An LP with unbounded optimum.

To put the problem in standard form, we introduce a slack variable in Constraint (1).

$$\begin{array}{llrcrcrcll} \min & z = & -x_1 & + & 2x_2 & & & & & \\ \text{s.t.} & & -x_1 & + & x_2 & + & s_1 = & 2 & & (1') \\ & & \multicolumn{8}{l}{x_1,\ x_2,\ s_1 \geq 0.} \end{array}$$

The initial bfs has basic variable $s_1 = 2$ and non-basic variables $x_1 = x_2 = 0$, corresponding to the extreme point $(x_1, x_2)^\mathsf{T} = (0, 0)$. Since we want to minimize, and the reduced cost of x_1 is negative, we want to increase the value of x_1, so x_1 becomes the entering basic variable.

To determine how much x_1 can increase, we need to perform the min-ratio test, but we immediately notice that the coefficient of x_1 in Constraint (1') is negative, so we can increase the value of x_1 infinitely: $s_1 = 2 + x_1 \geq 0 \Rightarrow x_1 \geq -2$, which trivially holds, since x_1 is required to be nonnegative. $\square$

To conclude:
If, for a possible entering basic variable x_k, the current constraint coefficients $\bar{a}_{ik}$ are less than or equal to zero *for all constraint rows* i, then the optimal solution is unbounded, i.e., in a maximization problem we have $z^* = +\infty$, and in a minimization problem we have $z^* = -\infty$. It is important to notice, that the problem is not unbounded just because the polyhedron is not bounded. If the objective function would have been different, a bounded optimum could have been obtained. Take for instance the objective function $\min z = 2x_1 - x_2$, which yields the optimal extreme point $(0, 2)^\mathsf{T}$.

4.3.2 Multiple optimal solutions

In Example 4.1, the problem had a unique optimal solution. This is often not the case, and the questions are then:

1. How do we detect that there are multiple optimal solutions?
2. How do we describe all optimal solutions?

We illustrate multiple optima in the following example, which is a modification of Example 4.1.

Example 4.4.

$$\begin{array}{llrcrcrl}
\max & z = & 54x_1 & + & 45x_2 & & & \\
\text{s.t.} & & 6x_1 & + & 5x_2 & \leq & 60 & (1) \\
 & & x_1 & + & 2x_2 & \leq & 15 & (2) \\
 & & x_1 & & & \leq & 8 & (3) \\
 & & \multicolumn{5}{l}{x_1,\ x_2 \geq 0.} &
\end{array}$$

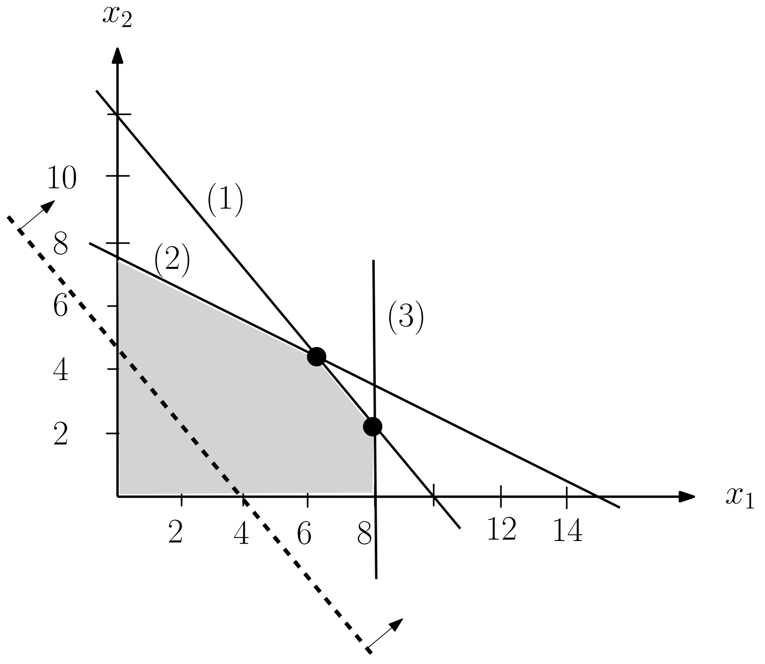


Figure 4.5: The objective function is parallel to Constraint (1), which yields two optimal extreme points.

The objective function is now parallel to Constraint (1). After two iterations of the Simplex method we obtain the following system:

$$\begin{array}{rcrcrcrcrcr}
-z & + & & - & 9s_1 & & & & & = & -540 \\
 & & x_2 & + & \frac{1}{5}s_1 & & & - & \frac{6}{5}s_3 & = & \frac{12}{5} \\
 & & & - & \frac{2}{5}s_1 & + & s_2 & + & \frac{7}{5}s_3 & = & \frac{11}{5} \\
x_1 & & & & & & & + & s_3 & = & 8 \\
 & & \multicolumn{7}{l}{x_1,\ x_2,\ s_1,\ s_2,\ s_3 \geq 0.} & &
\end{array}$$

Since all reduced costs are nonpositive, an optimal solution has been obtained, namely $(x_1^*, x_2^*)^\mathsf{T} = (8, 12/5)$ with $z^* = 540$. By definition, all basic variables have reduced cost equal to zero. Here, we also notice that the non-basic variable s_3 has reduced cost zero. We can, without changing the objective value, make s_3 entering basic variable. As usual, the min-ratio test determines the leaving basic variable:

$$\min_i \{ \frac{\bar{b}_i}{\bar{a}_{i,s_3}} \mid \bar{a}_{i,s_3} > 0 \} = \min\{ \frac{11/5}{7/5},\ 8 \} = 11/7\,,$$

which implies that variable s_2 becomes leaving basic variable. Making the pivot results in the following system:

$$\begin{array}{rcrcrcrcrcl}
-z & + & & & - & 9s_1 & & & & = & -540 \\
& & x_2 & & - & \frac{1}{7}s_1 & + & \frac{6}{7}s_2 & & = & 4\frac{2}{7} \\
& & & & - & \frac{2}{7}s_1 & + & \frac{5}{7}s_2 & + \; s_3 & = & \frac{11}{7} \\
x_1 & & & & + & \frac{2}{7}s_1 & - & \frac{5}{7}s_2 & & = & 6\frac{3}{7} \\
\end{array}$$

$$x_1,\ x_2,\ s_1,\ s_2,\ s_3 \geq 0.$$

The second optimal extreme point is $(x_1^*, x_2^*)^\mathsf{T} = (6\frac{3}{7},\ 4\frac{2}{7})$. The two optimal extreme points are indicated in Figure 4.5. In fact, not only the two extreme points are optimal, but *every* point on the line segment between these points are optimal, i.e., the set of *all optimal points* is the convex combination of the optimal extreme points:

$$\boldsymbol{x}^* = \lambda \begin{pmatrix} 8 \\ \frac{12}{5} \end{pmatrix} + (1-\lambda) \begin{pmatrix} 6\frac{3}{7} \\ 4\frac{2}{7} \end{pmatrix},$$

for $0 \leq \lambda \leq 1$.

□

In general, suppose $\boldsymbol{x}^1, \dots, \boldsymbol{x}^k$ are optimal extreme points, then the set of all optimal points can be expressed as

$$\boldsymbol{x}^* = \lambda_1 \boldsymbol{x}^1 + \cdots + \lambda_k \boldsymbol{x}^k \quad \text{for } \lambda_1, \dots, \lambda_k \geq 0 \text{ and } \sum_{i=1}^{k} \lambda_i = 1\,.$$

4.4 Finding an initial basic feasible solution

In the examples we have seen so far, we could determine a starting bfs simply by adding a slack variable s_i to each of the constraints. In this way we have a solution satisfying requirements **(R1)** and **(R2)** (see Section 4.1). To obtain a starting basic solution in general, we always start by putting the set of constraints in standard form, see Definition 4.1. Once we have the problem in standard form we make sure that there is one variable in each constraint with a $+1$ coefficient in that constraint and with a coefficient 0 in all other constraints and in the objective function. Regardless of the initial formulation of the LP, we can reformulate it such that it satisfies requirements **(R1)** and **(R2)**. We will show how in this section. First we describe how to obtain the standard form in general, i.e., how to satisfy requirement **(R1)**.

4.4.1 Satisfying requirement (R1): Obtaining the standard form

Case 1: We have a constraint with $b_i < 0$. In this case, we simply multiply the whole constraint by -1. Suppose we have a constraint $x_1 - x_2 = -5$, then we multiply by -1 to obtain $-x_1 + x_2 = 5$.

Case 2: We have a constraint $\boldsymbol{a}_i\boldsymbol{x} \leq b_i,\ b_i \geq 0$. This case we have seen before. We simply add a slack variable $s_i \geq 0$ to the left-hand side to obtain a constraint in equality form, see Example 4.1.

Case 3: We have a constraint $\boldsymbol{a}_i\boldsymbol{x} \geq b_i,\ b_i \geq 0$. In this case the left-hand side is at least as large a the right-hand side. Here we need to *subtract* a nonnegative variable in the left-hand side to obtain equality form. The variable we subtract is in this case referred to as a *surplus variable*:

$$\boldsymbol{a}_i\boldsymbol{x} \geq b_i \text{ becomes } \boldsymbol{a}_i\boldsymbol{x} - s_i = b_i \text{ with } s_i \geq 0\,.$$

Case 4: We have a variable that is restricted to be nonpositive, i.e., $x_k \le 0$. Here, we replace x_k by its negative counterpart in the whole problem formulation, i.e., substitute x_k by $x_k^{'} = -x_k \ge 0$ in the problem formulation. As an example, suppose we have

$$\begin{aligned} \max z = x_1 + x_2 & & \\ \text{s.t. } 2x_1 - 3x_2 &= 5 \\ x_1 \ge 0, x_2 &\le 0\,, \end{aligned}$$

then, after substitution, we obtain

$$\begin{aligned} \max z = x_1 - x_2^{'} & & \\ \text{s.t. } 2x_1 + 3x_2^{'} &= 5 \\ x_1, x_2^{'} &\ge 0\,. \end{aligned}$$

Case 5: We have a variable that is unrestricted in sign, i.e., x_k is "free". In this case we substitute x_k by the difference between two variables that we denote x_k^+ and x_k^- that are both nonnegative, i.e., $x_k = x_k^+ - x_k^-$, $x_k^+, x_k^- \ge 0$. As an example, suppose we have

$$\begin{aligned} \max z = x_1 + x_2 & & \\ \text{s.t. } 2x_1 - 3x_2 &= 5 \\ x_1 \ge 0,\ x_2 &\in \mathbb{R}\,, \end{aligned}$$

then, after substitution, we obtain

$$\begin{aligned} \max z = x_1 + x_2^+ - x_2^- & & \\ \text{s.t. } 2x_1 - 3x_2^+ + 3x_2^- &= 5 \\ x_1, x_2^+, x_2^- &\ge 0\,. \end{aligned}$$

4.4.2 Satisfying requirement (R2): Determining a starting bfs, and the Simplex 2-phase method

In Section 4.1, we determined a starting bfs in the case that all constraints are in $\le$-form. We simply added a slack variable for each constraint, and the slack variables formed the first set of basis variables. All of the original variables are then non-basic, and therefore equal to zero, i.e., *our starting extreme point is the origin.*

In many cases the origin is not feasible, which typically happens when there are constraints of $\ge$- or $=$-form in our formulation. What the Simplex method then does, is still to start in the *infeasible* origin, and then pivot through *infeasible* basic solutions satisfying requirements **(R1)** and **(R2)**, until it reaches a bfs, or until we can conclude that no bfs exists. This version of Simplex is called the 2-phase Simplex method. In Phase 1, we pivot on infeasible basic solutions, and minimize the "infeasibility". Before we describe the 2-phase Simplex method, we describe how we can make the origin the starting basic solution in case our problem contains constraints of $\ge$- or $=$-form. In the discussion that follows, we assume that the problem is in standard form.

Case 1: We have a constraint $\boldsymbol{a}_i\boldsymbol{x} - s_i = b_i$. This means that our original constraint was $\boldsymbol{a}_i\boldsymbol{x} \ge b_i$, and we subtracted a surplus variable to obtain standard form. In this case we cannot choose $s_i \ge 0$ as the basic variable for this constraint, since we then would get $s_i = -b_i \le 0$, which is infeasible. To overcome this situation we introduce a so-called artificial variable $x_i^a \ge 0$ that will be basic variable for constraint i. In some sense we "trick" Simplex to accept this variable as a starting basic variable, as Simplex works under the assumption that all variables are nonnegative. The artificial variable does not add any information to the problem itself, it is simply the negative of the surplus. Our constraint now becomes:

$$\boldsymbol{a}_i\boldsymbol{x} - s_i + x_i^a = b_i\,.$$

Case 2: We have a constraint $\boldsymbol{a}_i\boldsymbol{x} = b_i$. In this case we do not have an obvious candidate as starting basic variable, unless of course there happens to be one variable with a positive coefficient in the equation that has coefficient zero in all other rows. Again, the solution is to introduce an artificial variable to obtain:

$$\boldsymbol{a}_i\boldsymbol{x} + x_i^a = b_i\,,$$

and let x_i^a be the starting basic variable for that constraint. This is simply an effective way of finding a starting basis.

Notice that all artificial variables have to take value zero in any feasible solution.

Example 4.5. Consider the following LP:

$$\begin{array}{llrcrcll}
\max & z = & x_1 & + & 2x_2 & & & \\
\text{s.t.} & & x_1 & + & x_2 & = & 4 & (1) \\
 & & x_1 & & & \leq & 3 & (2) \\
 & & & & x_2 & \geq & 1 & (3) \\
 & & \multicolumn{5}{l}{x_1,\ x_2 \geq 0.} &
\end{array}$$

The problem in standard form is:

$$\begin{array}{llrcrcrcrcll}
\max & z = & x_1 & + & 2x_2 & & & & & & & \\
\text{s.t.} & & x_1 & + & x_2 & & & & & = & 4 & (1') \\
 & & x_1 & & & + & s_2 & & & = & 3 & (2') \\
 & & & & x_2 & & & - & s_3 & = & 1 & (3') \\
 & & \multicolumn{9}{l}{x_1,\ x_2,\ s_2,\ s_3 \geq 0.} &
\end{array}$$

Finally, to obtain a starting bfs we need to introduce an artificial variable in Constraints (1') and (3') to obtain:

$$\begin{array}{llrcrcrcrcrcrcll}
\max & z = & x_1 & + & 2x_2 & & & & & & & & & & & \\
\text{s.t.} & & x_1 & + & x_2 & & & & & + & x_1^a & & & = & 4 & (1'') \\
 & & x_1 & & & + & s_2 & & & & & & & = & 3 & (2'') \\
 & & & & x_2 & & & - & s_3 & & & + & x_3^a & = & 1 & (3'') \\
 & & \multicolumn{13}{l}{x_1,\ x_2,\ s_2,\ s_3,\ x_1^a,\ x_3^a \geq 0.} &
\end{array}$$

The starting basis will now be

$$\begin{pmatrix} x_1^a \\ s_2 \\ x_3^a \end{pmatrix} = \begin{pmatrix} 4 \\ 3 \\ 1 \end{pmatrix},$$

with non-basic variables $x_1 = x_2 = s_3 = 0$. In the eyes of Simplex, this is a "feasible" solution since it satisfies requirements **(R1)** and **(R2)**, but with respect to the problem we want to solve, the solution is infeasible, since it represents the origin, which is not in the feasible set. □

If we have artificial variables in our basis we have represented our problem in a form that is acceptable to Simplex, but we actually are outside the set of feasible solutions to our original problem. Simplex Phase 1 is then used to iterate until we obtain a basic solution that is feasible for our original problem, or until we conclude that the problem at hand is infeasible.

Recall that all artificial variables take value zero in any feasible solution. It is therefore natural to introduce the objective function

$$w^* = \min w, \quad \text{with } w = \sum_i^l x_i^a \,, \tag{4.5}$$

to our Simplex Phase 1 constraints. If a feasible solution exists, we will obtain $w^* = 0$. Objective function (4.5) is called the *Simplex Phase 1 objective function.* We solve the Simplex Phase 1 problem until an optimal solution is found. Since we minimize, we have an optimal solution if all reduced costs are nonnegative. We are then in one of the following three cases:

Case 1: $w^* = 0$ and all artificial variables are non-basic.

Case 2: $w^* = 0$ and one or more artificial variables are basic with value 0. Take any of the basic zero-valued artificial variables, say x_i^a, and pivot it out of the basis by choosing any arbitrary non-zero element in the row of x_i^a as pivot element. Repeat until all artificial variables are non-basic. We are then in the Case 1 situation.

Case 3: $w^* > 0$. In this case, the original problem is infeasible, and we can stop.

If we have reached the Case 1 situation, we drop all artificial variables from the formulation, and use the Simplex Phase 1 optimal basis as a starting point to iterate further with the original objective function. This is then Simplex Phase 2. When reintroducing the original objective function, we first express it in terms of the current non-basic variables.

Example 4.5, cont. The Simplex Phase 1 problem is:

$$\begin{array}{llcccccccccl}
\min & w = & & & & & & x_1^a & + & x_3^a & & \\
\text{s.t.} & & x_1 & + & x_2 & & & + \; x_1^a & & & = 4 & (1'') \\
& & x_1 & & & + & s_2 & & & & = 3 & (2'') \\
& & & & x_2 & & - \; s_3 & & + & x_3^a & = 1 & (3'') \\
& & \multicolumn{8}{l}{x_1, \; x_2, \; s_2, \; s_3, \; x_1^a, \; x_3^a \geq 0.} & &
\end{array}$$

Notice that the basic variables x_1^a and x_3^a occur in the objective function, so we need to formulate the Phase 1 objective function in terms of non-basic variables before we can start with Simplex Phase 1, otherwise the problem does not satisfy requirement (**R2**). This is easily done by just using Constraints (1”) and (3”) to express the variables x_1^a and x_3^a in non-basic variables., i.e.,

$$\begin{aligned}
x_1^a &= 4 - x_1 - x_2 \\
x_3^a &= 1 - x_2 + s_3 \,.
\end{aligned}$$

So, we obtain $w = x_1^a + x_3^a = 5 - x_1 - 2x_2 + s_3$. We have now obtained the following Simplex Phase 1 problem that satisfies requirements **(R1)** and **(R2)**.

$$\begin{array}{cccccccccccl}
-w & -x_1 & - & 2x_2 & & & + \; s_3 & & & & = -5 & (0) \\
& x_1 & + & x_2 & & & & + \; x_1^a & & & = 4 & (1) \\
& x_1 & & & + & s_2 & & & & & = 3 & (2) \\
& & & x_2 & & & - \; s_3 & & + & x_3^a & = 1 & (3) \\
& \multicolumn{9}{l}{x_1, \; x_2, \; s_2, \; s_3, \; x_1^a, \; x_3^a \geq 0.} & &
\end{array}$$

The iterations of Simplex Phase 1 are as follows:

$$\begin{array}{rrrrrrrrrrrrl}
-w & -x_1 & & & & & - & s_3 & & & + & 2x_3^a & = & -3 \\
& x_1 & & & & & + & s_3 & + & x_1^a & - & x_3^a & = & 3 \\
& x_1 & & & + & s_2 & & & & & & & = & 3 \\
& & & x_2 & & & - & s_3 & & & + & x_3^a & = & 1 \\
\multicolumn{14}{c}{x_1,\ x_2,\ s_2,\ s_3,\ x_1^a,\ x_3^a \geq 0,}
\end{array}$$

and

$$\begin{array}{rrrrrrrrrrrrl}
-w & & & & & & & & + & x_1^a & + & x_3^a & = & 0 \\
& x_1 & & & & & + & s_3 & + & x_1^a & - & x_3^a & = & 3 \\
& x_1 & & & + & s_2 & & & & & & & = & 3 \\
& x_1 & + & x_2 & & & & & + & x_1^a & & & = & 4 \\
\multicolumn{14}{c}{x_1,\ x_2,\ s_2,\ s_3,\ x_1^a,\ x_3^a \geq 0\,.}
\end{array}$$

Since all the reduced costs are nonnegative, we have reached an optimal Phase 1 solution. Since $w^* = 0$ and both artificial variables are non-basic, we are in Case 1, so we have reached a feasible solution of the original problem. We illustrate in Figure 4.6 how we iterated to obtain the optimal Phase 1 solution.

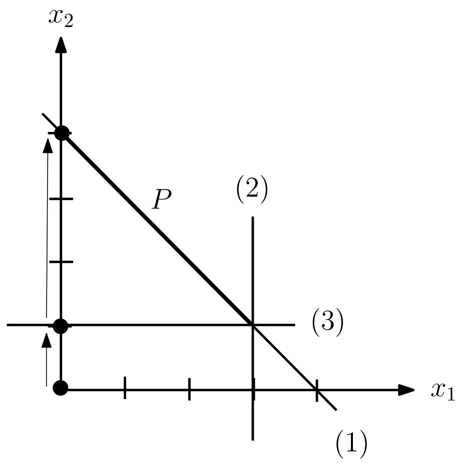


Figure 4.6: The polytope is just the line segment between the points $(0,4)^\mathsf{T}$ and $(1,1)^\mathsf{T}$. The dots show the points visited by Simplex Phase 1. The optimal Phase 1 solution is the first solution that is feasible for the original problem.

We are now ready to start Phase 2, and we then remove all artificial variables and reintroduce the original objective function $\max z = x_1 + 2x_2$. Before we start Phase 2, we express the objective function in terms of the non-basic variables so we obtain a solution that satisfies requirement **(R2)**. To do that, we use the last row of the last system of equations to express the basic variable x_2 as a function of non-basic variables, $x_2 = 4 - x_1$. The original objective function now becomes $\max z = x_1 + 2x_2 = x_1 + 8 - 2x_1 = 8 - x_1$. Our Simplex Phase 2 system is as follows:

$$\begin{array}{rrrrrrrrrl}
-z & -x_1 & & & & & & & = & -8 \\
& x_1 & & & & & + & s_3 & = & 3 \\
& x_1 & & & + & s_2 & & & = & 3 \\
& x_1 & + & x_2 & & & & & = & 4 \\
\multicolumn{10}{c}{x_1,\ x_2,\ s_2,\ s_3 \geq 0\,.}
\end{array}$$

We observe that all reduced costs are nonpositive, and since we maximize this means that the optimal solution is found. If the solution had not been optimal, we had continued with Simplex as usual. The optimal solution is

$$\begin{pmatrix} x_1^* \\ x_2^* \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \end{pmatrix} \text{ with } z^* = 8\,.$$

□

4.5 The Simplex method formally

We are now ready to introduce the Simplex method more formally. This description covers both Phase 1 and Phase 2. Phase 1 is a minimization problem. If Phase 1 terminates with optimal objective value $w^* > 0$, the original problem is infeasible and we can stop.

Initialize: Write the problem such that it satisfies requirements **(R1)** and **(R2)**.

Step 1: Choose *entering basic variable*. In a minimization problem, choose a variable x_k with $\bar{c}_k < 0$, and in a maximization problem choose a variable x_k with $\bar{c}_k > 0$ and go to Step 2. If all $c_j \geq 0$ (minimization) or all $c_j \leq 0$ (maximization), the current bfs is optimal, stop.

Step 2: Choose *leaving basic variable* x_l. Perform the min ratio test:

$$l = \text{argmin}_i \{ \frac{\bar{b}_i}{\bar{a}_{ik}} \mid \bar{a}_{ik} > 0 \}$$

If $\bar{a}_{ik} \leq 0$ for all i, then the problem is unbounded, stop. Otherwise, pivot on the element $\bar{a}_{lk}$, i.e., apply row operations to the current system such that element $\{l, k\}$ takes value $+1$, and such that all other elements in column k take value zero. Go to Step 1.

We have referred to Simplex so far as the "Simplex method". To really make it an algorithm, we need to specify a so-called pivot rule, that is, a specific way of choosing an entering and a leaving variable in the case we have a choice. In the examples we chose the entering variable as the variable with the largest positive reduced cost (max) or the smallest negative reduced cost (min). In the case of leaving variable we always have to choose a variable that gives the smallest ratio in the min ratio rule, but we have not specified how to choose in case of equal values. In this text we just break ties arbitrarily. It would be nice if we could specify a pivot rule that is provably efficient in the sense of analyzing algorithms as described in Chapter 15, but so far no pivot rule is known to be provably efficient.

One remark concerns the way we write every pivot step. So far we have written it in equation form since what we really do in Simplex is row operations on a set of equations. One can represent the steps more compactly using a so-called "Simplex tableau". This is a more compact way of representing the equations, avoiding to write down all variable names in each iteration. This tableau form will be used in Chapter 5 and introduced there more formally, but we include an iteration of the Simplex method here, in tableau form, to illustrate the layout.

Example 4.1, cont.
Here we illustrate the starting tableau and the first iteration of the problem introduced in Example 4.1.
Starting tableau:

basis	$\bar{b}$	x_1	x_2	s_1	s_2	s_3
s_1	60	6	5	1		
s_2	15	1	2		1	
s_3	8	**1**				1
$-z$	0	50	45			

The entering basic variable is x_1 and the leaving basic variable is s_3. The pivot element (in column x_1 and row s_3) is marked in bold.
The first iteration:
Next to the rows we indicate the row operations that were performed to obtain this tableau. Here r_i denotes constraint row i, $i = 1, 2, 3$ and r_z indicates the objective row.

basis	$\bar{b}$	x_1	x_2	s_1	s_2	s_3	
s_1	12		**5**	1		-6	$r_1 - 6r_3$
s_2	7		2		1	-1	$r_2 - r_3$
x_1	8	1				1	r_3
$-z$	-400		45			-50	$r_z - 50r_3$

In the next iteration (left as an exercise), variable x_2 enters the basis, and s_1 leaves it. The element in column x_2 and row s_1 is then the pivot element, marked in bold in the tableau.

□

Before we finish the chapter we return to the issue of degeneracy, introduced in Section 4.1. Definition 4.5. Recall that one extreme point of a polyhedron can correspond to multiple degenerate basic feasible solutions. This can cause the Simplex method to cycle between these degenerate solutions so that we do not move on to a new extreme point. Bland [2] developed an *anti-cycling pivot rule* that guarantees that the Simplex algorithm terminates:

- If more than one variable can be chosen as entering basic variable, choose the one with the smallest index.
- If more than one variable can be chosen as leaving basic variables, choose the one with the smallest index.

Bland proved the following result.

Theorem 4.3. The Simplex algorithm with Bland's anti-cycling pivot rule terminates in a finite number of steps.

4.6 Exercises

Exercise 4.1. **a)** Put the following two LP formulations in standard form:

(i)

$$\begin{array}{llrcrcr} \min & z = & -3x_1 & - & 4x_2 & & \\ \text{s.t.} & & 2x_1 & + & 3x_2 & \leq & 8 \\ & & & & x_2 & \geq & -2 \\ & & & & \multicolumn{3}{r}{x_1 \geq 0,\ x_2 \in \mathbb{R}} \end{array}$$

(ii)

$$\begin{array}{llrcrcr} \max & z = & -5x_1 & + & 7x_2 & & \\ \text{s.t.} & & 2x_1 & + & 9x_2 & = & 13 \\ & & -5x_1 & + & 3x_2 & \leq & 20 \\ & & & & \multicolumn{3}{r}{x_1 \leq 0,\ x_2 \geq 0} \end{array}$$

b) Give three basic solutions of the standard form of a)(ii). Are they feasible?

c) Draw the feasible region of a)(ii) and indicate which extreme point corresponds to which basic solution from b).

Exercise 4.2. At the lecture, we have seen the following. If two different bases give the same basic feasible solution $\boldsymbol{x}$, then $\boldsymbol{x}$ is degenerate (i.e. more than $n-m$ variables in the standard form have value 0). Show that the converse is not true in general by giving an example LP and a basic feasible solution that is degenerate but has only one corresponding basis.

Exercise 4.3. Suppose an LP has n free variables (variables that may take all values in $\mathbb{R}$). Show that these n free variables can be replaced by $n+1$ nonnegative variables.

Exercise 4.4. Solve the LP below with the Simplex method, draw the feasible region and determine for each basic feasible solution the corresponding extreme point. Is the obtained optimal solution unique? If not, give all optimal solutions.

$$\begin{array}{lcrcrcr}
\min & z = & 3x_1 & - & 6x_2 & & \\
\text{s.t.} & & 5x_1 & + & 7x_2 & \leq & 35 \\
 & & -x_1 & + & 2x_2 & \leq & 2 \\
 & & & & \multicolumn{3}{l}{x_1 \geq 0,\ x_2 \geq 0\,.}
\end{array}$$

Exercise 4.5. Suppose that for a certain LP no basic feasible solution is degenerate. Does it follow that this LP has a unique optimal solution? Give a proof or a counterexample.

Exercise 4.6. Solve the LP below using the Simplex method, draw the feasible region and determine for each basic feasible solution the corresponding extreme point.

$$\begin{array}{lcrcrcr}
\min & z = & 2x_1 & + & x_2 & & \\
\text{s.t.} & & x_1 & + & x_2 & \geq & 1 \\
 & & 3x_1 & + & 2x_2 & \leq & 6 \\
 & & & & \multicolumn{3}{l}{x_1 \geq 0,\ x_2 \geq 0\,.}
\end{array}$$

Exercise 4.7. Solve the LP below using the Simplex method, draw the feasible region and determine for each basic feasible solution the corresponding extreme point.

$$\begin{array}{lcrcrcr}
\min & z = & -3x_1 & + & x_2 & -20 & \\
\text{s.t.} & & -3x_1 & + & 3x_2 & \leq & 6 \\
 & & -8x_1 & + & 4x_2 & \leq & 4 \\
 & & & & \multicolumn{3}{l}{x_1 \geq 0,\ x_2 \geq 0\,.}
\end{array}$$

Exercise 4.8. Solve the LP below using the Simplex method, draw the feasible region and determine for each basic feasible solution the corresponding extreme point.

$$\begin{array}{lrcrcrcl} \max & z & = & 240x_1 & + & 60x_2 & & \\ \text{s.t.} & & & 2x_1 & + & 2x_2 & \leq & 100 \\ & & & 6x_1 & + & x_2 & \leq & 100 \\ & & & 10x_1 & & & \leq & 100 \\ & & & & & \multicolumn{3}{l}{x_1 \geq 0,\ x_2 \geq 0\,.} \end{array}$$

In the second iteration, choose s_2 as leaving basic variable. What changes to the solution in the last iteration?

Exercise 4.9. Find **all** optimal solutions of the LP below, using the Simplex method.

$$\begin{array}{ll} \min & -4\,x_1 + x_2 - x_3 \\ \text{s.t.} & \quad x_1 + x_2 + x_3 \leq 5 \\ & \qquad\quad x_2 + x_3 \geq 2 \\ & \quad 2\,x_1 \qquad + x_3 \leq 1 \\ & x_1, x_2, x_3 \geq 0 \end{array}$$

Exercise 4.10. Prove that cycling cannot occur in the Simplex method when in each iteration no basic variable has value zero.

Exercise 4.11. Prove that cycling cannot occur in the Simplex method when in each iteration at most one basic variable has value zero.

Exercise 4.12. Consider the following LP and assume that $b,\ a_1, \ldots, a_n \neq 0$.

$$\begin{array}{lrcl} \min & \sum_{j=1}^{n} c_j x_j & & \\ \text{s.t.} & \sum_{j=1}^{n} a_j x_j & = & b \\ & x_j & \geq & 0 \quad \text{for } j = 1, \ldots, n\,. \end{array}$$

(a) Design a test to check if the LP has a *feasible* solution.

(b) Design a test to check if the LP is *unbounded.*

(c) Design a simple method (not Simplex) to find an *optimal solution.* Argue also why the obtained solution is optimal.

Chapter 5

Linear Programming Duality

Each linear programming problem has a certain *dual* problem. The dual of a minimization problem is a maximization problem, while the dual of a maximization problem is a minimization problem. This dual problem has many nice properties and forms the basis for most theory and practical algorithms for solving linear and integer linear programming problems.

We will see in this chapter that the variables of the dual problem correspond to the constraints of the original problem, which we call the *primal* problem, while the constraints of the dual problem correspond to the variables of the primal problem. Moreover, it will be shown that solutions to the dual problem give us bounds on the optimal value of the primal problem. As a result, duality can be used to verify that a given solution is optimal.

5.1 The diet problem

We will first explain the ideas behind the dual problem by looking at a simple but insightful example. The next section will explain how the dual can be formulated in general.

Suppose you wish to compose an optimal diet consisting of different types of food. Each type of food contains certain nutrients. Of course, you have to make sure that you eat enough of each nutrient (i.e., at least the reference daily intake (RDI)). However, you do not want to spend more money on food than necessary. Therefore, you minimize the total price of your diet (per day).

The following is a simple toy example of the diet problem.

	chips	muesli	sausage	required (RDI)
costs	3	3	4	
carbohydrates	2	1	0	3
protein	0	1	4	2
fat	4	0	8	9

The question is how much we should eat from each of the food types (chips, muesli, sausage) in order to minimize the total costs while having sufficient intake from each nutrient. Introducing variables x_i indicating how much we eat from the ith food type, this problem can be formulated as a linear programming problem as follows.

$$\begin{array}{llrcrcrcrl}
\min & z = & 3x_1 & + & 3x_2 & + & 4x_3 & & & \\
\text{s.t.} & & 2x_1 & + & x_2 & & & \geq & 3 & (1) \\
 & & & & x_2 & + & 4x_3 & \geq & 2 & (2) \\
 & & 4x_1 & & & + & 8x_3 & \geq & 9 & (3) \\
 & & \multicolumn{5}{l}{x_1,\ x_2,\ x_3 \geq 0} & & &
\end{array}$$

5.1.1 A first lower bound

Recall that the dual problem is about finding bounds on the optimal value. Since the diet problem is a minimization problem, we are especially interested in lower bounds (each feasible solution to the primal problem already gives an upper bound on the optimum). How much do we certainly need to spend at least? First look what happens when you add up the first two constraints:

$$\begin{array}{ccccccl|c} 2x_1 & + & x_2 & & & \geq 3 & & (1) \\ & & x_2 & + & 4x_3 & \geq 2 & & (2) \\ \hline 2x_1 & + & 2x_2 & + & 4x_3 & \geq 5 & & (1)+(2) \end{array}$$

Now compair the result to the objective function. Since the variables x_1, x_2 are nonnegative, we find that

$$z = 3x_1 + 3x_2 + 4x_3 \geq 2x_1 + 2x_2 + 4x_3 \geq 5.$$

Hence, 5 is a lower bound on our objective function. We know for sure that we will need to spend at least 5 Euros on our diet.

5.1.2 A better lower bound

We can find a better lower bound by taking $1\frac{1}{2}$ times the first constraint plus the second constraint:

$$\begin{array}{cccccll} 3x_1 & + & 1\frac{1}{2}x_2 & & & \geq 4\frac{1}{2} & 1\frac{1}{2} \times (1) \\ & & x_2 & + & 4x_3 & \geq 2 & (2) \\ \hline 3x_1 & + & 2\frac{1}{2}x_2 & + & 4x_3 & \geq 6\frac{1}{2} & 1\frac{1}{2} \times (1) + (2) \end{array}$$

This way, we find that

$$z = 3x_1 + 3x_2 + 4x_3 \geq 3x_1 + 2\frac{1}{2}x_2 + 4x_3 \geq 6\frac{1}{2}.$$

Hence, we know that we will need to spend at least 6.50 Euros on our diet. Are there even better (i.e. bigger) lower bounds? To answer that question we need a more systematic approach.

5.1.3 Dual of the diet problem

To find lower bounds systematically, notice that we can try each possible linear combination of the constraints. So, we add up π_1 times the first constraint, plus π_2 times the second constraint, plus π_3 times the third constraint, with $\pi_1, \pi_2, \pi_3 \geq 0$ new variables, called the *dual variables*. We get the following.

$$\begin{array}{cccccccc} 2\pi_1 x_1 & + & \pi_1 x_2 & & & \geq & 3\pi_1 \\ & & \pi_2 x_2 & + & 4\pi_2 x_3 & \geq & 2\pi_2 \\ 4\pi_3 x_1 & & & + & 8\pi_3 x_3 & \geq & 9\pi_3 \\ \hline (2\pi_1 + 4\pi_3)x_1 & + & (\pi_1 + \pi_2)x_2 & + & (4\pi_2 + 8\pi_3)x_3 & \geq & 3\pi_1 + 2\pi_2 + 9\pi_3 \end{array}$$

Note that we need $\pi_1, \pi_2, \pi_3 \geq 0$ because otherwise the signs of the inequalities will flip. We will get back to this in the next subsection.

When does this linear combination of constraints give us a lower bound on the optimal value? Well, since the objective function is $z = 3x_1 + 3x_2 + 4x_3$, we just need that

$$\begin{array}{cccccccc} 2\pi_1 & & & + & 4\pi_3 & \leq & 3 \\ \pi_1 & + & \pi_2 & & & \leq & 3 \\ & & 4\pi_2 & + & 8\pi_3 & \leq & 4 \end{array} \tag{5.1}$$

and then we have

$$z = 3x_1 + 3x_2 + 4x_3 \geq (2\pi_1 + 4\pi_3)x_1 + (\pi_1 + \pi_2)x_2 + (4\pi_2 + 8\pi_3)x_3 \geq 3\pi_1 + 2\pi_2 + 9\pi_3$$

and hence that $3\pi_1 + 2\pi_2 + 9\pi_3$ is a lower bound on the optimal value.

The best possible lower bound we get by maximizing this value, $3\pi_1 + 2\pi_2 + 9\pi_3$, under the constraints (5.1).

To summarize the discussion above, we get the following dual for the toy example of the diet problem.

$$\begin{array}{llrcrcrcl} \max & w = & 3\pi_1 & + & 2\pi_2 & + & 9\pi_3 & & \\ \text{s.t.} & & 2\pi_1 & & & + & 4\pi_3 & \leq & 3 \\ & & \pi_1 & + & \pi_2 & & & \leq & 3 \\ & & & & 4\pi_2 & + & 8\pi_3 & \leq & 4 \\ & & \multicolumn{7}{l}{\pi_1, \pi_2, \pi_3 \geq 0} \end{array} \tag{5.2}$$

In the case of the diet problem, there is a nice economic interpretation of the dual. Suppose that a company wanted to sell pills for the various nutrients. What would the optimal prices for those pills be? Of course, in order to maximize their profit, the company would want to make the price of your diet as high as possible. Hence, if π_1, π_2, π_3 are the prices of carbohydrate-pills, protein-pills and fat-pills, respectively, the company wants to maximize $3\pi_1 + 2\pi_2 + 9\pi_3$, which is indeed the objective of the dual problem (5.2). However, if it is cheaper to eat chips than to take the corresponding pills, nobody would buy the pills. Hence, the company needs to make sure that $2\pi_1 + 4\pi_3 \leq 3$, which is precisely the first dual constraint. Applying the same reasoning for muesli and suasages leads to the second and third dual constraint.

5.2 Dual of linear programming problems in general form

In the diet problem, we only had $\geq$-constraints and nonnegative variables. Now consider the LP below, which has a $\leq$-constraint, a $=$-constraint, a free variable and a nonpositive variable.

$$\begin{array}{llrcrcrl} \min & z = & 3x_1 & + & 8x_2 & & & \\ \text{s.t.} & & x_1 & + & 3x_2 & \leq & 5 & \quad (1) \\ & & x_1 & + & 2x_2 & = & -2 & \quad (2) \\ & & \multicolumn{5}{l}{x_1 \in \mathbb{R}, x_2 \leq 0} & \end{array}$$

As before, we will take linear combinations of the constraints, by taking π_1 times the first constraint plus π_2 times the second constraint. We need that $\pi_1 \leq 0$ because the sign of the first constrained needs to be flipped in order to get a lower bound. We have $\pi_2 \in \mathbb{R}$ because the second constraint is an equality, so we do not need to worry about the sign to flip. We get the following:

$$\begin{array}{rcrcr} \pi_1 x_1 & + & 3\pi_1 x_2 & \geq & 5\pi_1 \\ \pi_2 x_1 & + & 2\pi_2 x_2 & = & -2\pi_2 \\ \hline (\pi_1 + \pi_2)x_1 & + & (3\pi_1 + 2\pi_2)x_2 & \geq & 5\pi_1 - 2\pi_2. \end{array}$$

The question is now, when does $5\pi_1 - 2\pi_2$ form a lower bound on $z = 3x_1 + 8x_2$? Because x_1 is a free variable, we need that $\pi_1 + \pi_2 = 3$ in order to be sure that $(\pi_1 + \pi_2)x_1 \geq 3x_1$. Since $x_2 \leq 0$, we need that $3\pi_1 + 2\pi_2 \geq 8$ in order to make sure that $(3\pi_1 + 2\pi_2)x_2 \leq 8x_2$. We get the best possible lower bound by maximizing the value of the obtained lower bound, $5\pi_1 - 2\pi_2$, under the mentioned constraints. To summarize the above discussion, the complete dual is as follows:

$$\begin{array}{llrcrcl} \max & w = & 5\pi_1 & - & 2\pi_2 & & \\ \text{s.t.} & & \pi_1 & + & \pi_2 & = & 3 \\ & & 3\pi_1 & + & 2\pi_2 & \geq & 8 \\ & & \multicolumn{5}{l}{\pi_1 \leq 0, \pi_2 \in \mathbb{R}.} \end{array}$$

Applying exactly the same reasoning as for the example, it is now easy to see that the dual of a linear program in general form can be found as follows:

the **minimization** problem: the dual **maximization** problem:

$$\begin{array}{llll} \min & z = \boldsymbol{c}^\mathsf{T}\boldsymbol{x} & & \\ \text{s.t.} & \boldsymbol{a_i x} = b_i & i \in M \\ & \boldsymbol{a_i x} \geq b_i & i \in \overline{M} \\ & \boldsymbol{a_i x} \leq b_i & i \in \tilde{M} \\ & x_j \geq 0 & j \in N \\ & x_j \leq 0 & j \in \tilde{N} \\ & x_j \in \mathbb{R} & j \in \overline{N} \end{array} \qquad \begin{array}{llll} \max & w = \boldsymbol{b}^\mathsf{T}\boldsymbol{\pi} & & \\ \text{s.t.} & \pi_i \in \mathbb{R} & i \in M \\ & \pi_i \geq 0 & i \in \overline{M} \\ & \pi_i \leq 0 & i \in \tilde{M} \\ & \boldsymbol{\pi}^\mathsf{T}\boldsymbol{A_j} \leq c_j & j \in N \\ & \boldsymbol{\pi}^\mathsf{T}\boldsymbol{A_j} \geq c_j & j \in \tilde{N} \\ & \boldsymbol{\pi}^\mathsf{T}\boldsymbol{A_j} = c_j & j \in \overline{N}, \end{array} \tag{5.3}$$

where $\boldsymbol{a_i}$ denotes the ith row and $\boldsymbol{A_j}$ denotes the jth column of matrix A, where $\boldsymbol{a_i}$ is a row vector and $\boldsymbol{A_j}$ a column vector.

We will often assume without loss of generality that $\tilde{M} = \tilde{N} = \emptyset$ since the corresponding constraints/variables can be easily transformed into other constraints/variables by multiplying corresponding rows/columns by -1.

The following theorem can be easily verified.

Theorem 5.1. The dual of the dual problem is the primal problem.

Hence, the dual of a maximization problem can also be found from (5.3) by taking the maximization problem as primal, making the minimization problem the dual.

5.3 Duality in matrix form

In this section, we will assume (without loss of generality) that the primal problem has the following form, i.e. it is the minimization problem from (5.3) with $\tilde{M} = \tilde{N} = \emptyset$:

$$\begin{array}{lllcll} \min & z = & \boldsymbol{c}^\mathsf{T}\boldsymbol{x} & & & \\ \text{s.t.} & & \boldsymbol{a_i x} & = & b_i & \forall i \in M \\ & & \boldsymbol{a_i x} & \geq & b_i & \forall i \in \overline{M} \\ & & x_j \geq 0 & & & \forall j \in N \\ & & x_j \in \mathbb{R} & & & \forall j \in \overline{N}. \end{array} \tag{5.4}$$

First, we write (5.4) in standard form by substituting $x_j = x_j^+ - x_j^-$ for all $j \in \overline{N}$, introducing surplus variables $s_i \geq 0$ for each $i \in \overline{M}$ and replacing the corresponding constraints by

$$\boldsymbol{a_i x} - s_i = b_i \quad \forall j \in \overline{M}.$$

Let $\hat{\boldsymbol{x}}$ be the obtained vector of variables, $\hat{\boldsymbol{c}}$ the obtained vector of objective function coefficients, $\hat{A}$ the obtained left-hand-coefficient matrix and $\boldsymbol{b}$ the (unchanged) vector of righthand coefficients. The obtained primal problem in standard form is then as follows.

$$\begin{array}{lllcl} \min & z = & \hat{\boldsymbol{c}}^\mathsf{T}\hat{\boldsymbol{x}} & & \\ \text{s.t.} & & \hat{A}\hat{\boldsymbol{x}} & = & \boldsymbol{b} \\ & & \hat{\boldsymbol{x}} \geq \boldsymbol{0} & & \end{array} \tag{5.5}$$

We have seen in Section 4.2 that in an optimal solution the objective function row of the Simplex tableau is nonnegative. If $\hat{B}$ is the basis matrix corresponding to the optimal solution, we can write this as follows in matrix form:

$$\bar{\hat{\boldsymbol{c}}}^{\mathsf{T}} = \hat{\boldsymbol{c}}^{\mathsf{T}} - \hat{\boldsymbol{c}}_B^{\mathsf{T}} \hat{B}^{-1} \hat{A} \geq 0.$$

Therefore, we have

$$\hat{\boldsymbol{c}}_B^{\mathsf{T}} \hat{B}^{-1} \hat{A} \leq \hat{\boldsymbol{c}}^{\mathsf{T}}.$$

We now define $\boldsymbol{\pi}^{\mathsf{T}} = \hat{\boldsymbol{c}}_B^{\mathsf{T}} \hat{B}^{-1}$ as the *dual variables*. Substituting this into the above inequality, we get

$$\boldsymbol{\pi}^{\mathsf{T}} \hat{A} \leq \hat{\boldsymbol{c}}^{\mathsf{T}},$$

which are the dual constraints of the problem in standard form.

Finally, we translate the dual constraints back to the original problem (5.4). To do so, we distinguish three groups of constraints:

1. for $j \in N$, the constraints $\boldsymbol{\pi}^{\mathsf{T}} \boldsymbol{A_j} \leq c_j$ corresponding to nonnegative primal variables;

2. for $j \in \overline{N}$, the constraints $\boldsymbol{\pi}^{\mathsf{T}} \boldsymbol{A_j} \leq c_j$ and $-\boldsymbol{\pi}^{\mathsf{T}} \boldsymbol{A_j} \leq -c_j$ corresponding to free variables of the primal problem; these constraints can be combined to $\boldsymbol{\pi}^{\mathsf{T}} \boldsymbol{A_j} = c_j$; and

3. for $i \in \overline{M}$, the constraints $-\pi_i \leq 0$, corresponding to surplus variables of the primal problem; these constrains are equivalent to $\pi_i \geq 0$.

To summarize, we obtain the following dual of (5.4):

$$\begin{array}{llllll} \max & w = & \boldsymbol{b}^{\mathsf{T}} \boldsymbol{\pi} & & & \\ \text{s.t.} & & \boldsymbol{\pi}^{\mathsf{T}} \boldsymbol{A_j} & \leq & c_j & \forall j \in N \\ & & \boldsymbol{\pi}^{\mathsf{T}} \boldsymbol{A_j} & = & c_j & \forall j \in \overline{N} \\ & & \pi_i \in \mathbb{R} & & & \forall i \in M \\ & & \pi_i \geq 0 & & & \forall i \in \overline{M}. \end{array} \tag{5.6}$$

5.4 Duality theorems

We now prove the weak duality theorem, which formalizes the intuition that each feasible solution to the dual problem provides a lower bound for the primal problem (if the primal problem is a minimization problem). We again assume that the primal problem is in the following form:

$$\begin{array}{llllll} \min & z = & \boldsymbol{c}^{\mathsf{T}} \boldsymbol{x} & & & \\ \text{s.t.} & & \boldsymbol{a_i x} & = & b_i & \forall i \in M \\ & & \boldsymbol{a_i x} & \geq & b_i & \forall i \in \overline{M} \\ & & x_j \geq 0 & & & \forall j \in N \\ & & x_j \in \mathbb{R} & & & \forall j \in \overline{N}. \end{array} \tag{5.7}$$

Theorem 5.2. If $\boldsymbol{x}$ is a feasible solution to the primal problem (5.7) and $\boldsymbol{\pi}$ is a feasible solution to the dual problem (5.6), then

$$z(\boldsymbol{x}) \geq w(\boldsymbol{\pi}).$$

Proof.

$$\begin{aligned}
z(\boldsymbol{x}) &= \sum_{j\in N} c_j x_j + \sum_{j\in \overline{N}} c_j x_j \\
&\geq \sum_{j\in N} (\boldsymbol{\pi}^\mathsf{T} \boldsymbol{A_j}) x_j + \sum_{j\in \overline{N}} (\boldsymbol{\pi}^\mathsf{T} \boldsymbol{A_j}) x_j \\
&= (\boldsymbol{\pi}^\mathsf{T} A)\boldsymbol{x} \\
&= \boldsymbol{\pi}^\mathsf{T} (A\boldsymbol{x}) \\
&= \sum_{i\in M} \pi_i (\boldsymbol{a_i}\boldsymbol{x}) + \sum_{i\in \overline{M}} \pi_i (\boldsymbol{a_i}\boldsymbol{x}) \\
&\geq \sum_{i\in M} \pi_i b_i + \sum_{i\in \overline{M}} \pi_i b_i \\
&= \boldsymbol{\pi}^\mathsf{T} \boldsymbol{b} \\
&= w(\boldsymbol{\pi}).
\end{aligned}$$

The first inequality follows from the dual constraints and the fact that $x_j \geq 0$ for $j \in N$. The second inequality follows from the primal constraints and the fact that $\pi_i \geq 0$ for $i \in \overline{M}$. □

We now continue to prove the strong duality theorem, which states that the primal and dual problems have the same optimal value, assuming that they are both feasible.

Theorem 5.3. If $\boldsymbol{x}^*$ is an optimal solution to the primal problem (5.7), then the dual problem (5.6) has an optimal solution $\boldsymbol{\pi}^*$ and

$$z(\boldsymbol{x}^*) = w(\boldsymbol{\pi}^*).$$

Proof. First note that when the primal problem has a feasible solution, it follows from Theorems 5.1 and 5.2 that the dual problem is bounded.

Now suppose that the primal problem has an optimal solution $\boldsymbol{x}^*$. Let $\hat{\boldsymbol{x}}^*$ be the corresponding optimal solution of the primal problem in standard form (5.5) and let $\hat{B}$ be the corresponding basis matrix. Then

$$\boldsymbol{\pi}^\mathsf{T} = \hat{\boldsymbol{c}}_B^\mathsf{T} \hat{B}^{-1}$$

is a feasible solution to the dual of (5.5) and

$$w(\boldsymbol{\pi}) = \boldsymbol{\pi}^\mathsf{T}\boldsymbol{b} = \hat{\boldsymbol{c}}_B^\mathsf{T} \hat{B}^{-1}\boldsymbol{b} = \hat{\boldsymbol{c}}_B^\mathsf{T} \hat{\boldsymbol{x}}_B = z(\boldsymbol{x}^*).$$

Since, by Theorem 5.2, no feasible solution to the dual problem can have a bigger objective function value than $z(\boldsymbol{x}^*)$, it follows that $\boldsymbol{\pi}$ is an optimal solution to the dual problem. □

We now summarize the possibilities for a primal problem and its dual. The combinations marked with a $\times$ in the table below are not possible, while the combinations marked with a ✓ are possible, see Exercises 5.3 and 5.4.

		Dual		
		Bounded optimum	Unbounded	Infeasible
Primal	Bounded optimum	✓	×	×
	Unbounded	×	×	✓
	Infeasible	×	✓	✓

5.5 Dual solutions in the primal Simplex tableau

In this section, we will see that, after solving the primal problem with the simplex method, we can simply find an optimal solution to the dual problem from the optimal primal Simplex tableau. At first, we will assume that the primal problem has the following form.

$$\begin{array}{llrcl} \min & z = & \boldsymbol{c}^\mathsf{T}\boldsymbol{x} & & \\ \text{s.t.} & & A\boldsymbol{x} & \leq & \boldsymbol{b} \\ & & \boldsymbol{x} \geq \boldsymbol{0} & & \end{array}$$

To solve such a problem with the Simplex method, we first introduce slack variables:

$$\begin{array}{llrcl} \min & z = & \boldsymbol{c}^\mathsf{T}\boldsymbol{x} & & \\ \text{s.t.} & & A\boldsymbol{x} + I\boldsymbol{s} & = & \boldsymbol{b} \\ & & \boldsymbol{x}, \boldsymbol{s} \geq \boldsymbol{0}. & & \end{array}$$

Assume that the primal problem has some optimal solution $\boldsymbol{x}^*$, let B be the corresponding basis matrix, $\boldsymbol{x}_B$ the basic variables and $\boldsymbol{c}_N$ the non-basic variables. Assume furthermore, for simplicity, that $A = \begin{bmatrix} B & N \end{bmatrix}$ and $\boldsymbol{c}^\mathsf{T} = \begin{bmatrix} \boldsymbol{c}_B^\mathsf{T} & \boldsymbol{c}_N^\mathsf{T} \end{bmatrix}$. Then the starting Simplex tableau can be written as follows (with possibly some duplicate columns):

basis	$\bar{\boldsymbol{b}}$	$\boldsymbol{x}_B$	$\boldsymbol{x}_N$	$\boldsymbol{s}$
$\boldsymbol{s}$	$\boldsymbol{b}$	B	N	I
$-z$	0	$\boldsymbol{c}_B^\mathsf{T}$	$\boldsymbol{c}_N^\mathsf{T}$	$\boldsymbol{0}^\mathsf{T}$

The final Simplex tableau has, in the columns of the basic variables, zeros in the objective function row and the identity matrix in the remaining rows. Hence, the final tableau can be obtained from the starting tableau by premultiplying the rows of the basic variables by B^{-1} and subtracting these rows $\boldsymbol{c}_B^\mathsf{T}$ times from the objective function row. This gives the following final tableau:

basis	$\bar{\boldsymbol{b}}$	$\boldsymbol{x}_B$	$\boldsymbol{x}_N$	$\boldsymbol{s}$
$\boldsymbol{x}_B$	$B^{-1}\boldsymbol{b}$	I	$B^{-1}N$	B^{-1}
$-z$	$-\boldsymbol{c}_B^\mathsf{T}B^{-1}\boldsymbol{b}$	$\boldsymbol{0}$	$\boldsymbol{c}_N^\mathsf{T} - \boldsymbol{c}_B^\mathsf{T}B^{-1}N$	$\boldsymbol{c}_B^\mathsf{T}B^{-1}$

In this tableau, we can find an optimal solution solution to the dual problem in the columns of the slack variables, in the objective function row:

$$\boldsymbol{\pi}^{*\mathsf{T}} = \boldsymbol{c}_B^\mathsf{T}B^{-1}.$$

It remains to consider the case that not all constraints are $\leq$-constraints, and hence not all rows get a slack variable. Note that in those cases, the two-phase Simplex method introduces artificial variables for those rows, and these can be used in exactly the same way as the slack variables above. If we keep the columns for the artificial variables until completing the Simplex method (phase two), we can read an optimal solution to the dual problem in the objective function row, in the columns of the artificial and slack variables.

5.6 Exercises

Exercise 5.1. Find the dual of each of the problems below.

(a)

$$\begin{array}{llrcrcr} \min & z = & 3x_1 & - & 6x_2 & & \\ \text{s.t.} & & 5x_1 & + & 7x_2 & \leq & 35 \\ & & -x_1 & + & 2x_2 & \leq & 2 \\ & & & & \multicolumn{3}{r}{x_1 \geq 0,\ x_2 \geq 0} \end{array}$$

(b)

$$\begin{array}{llrcrcr} \min & z = & 2x_1 & + & x_2 & & \\ \text{s.t.} & & x_1 & + & x_2 & \geq & 1 \\ & & 3x_1 & + & 2x_2 & \leq & 6 \\ & & & & \multicolumn{3}{r}{x_1 \geq 0,\ x_2 \geq 0} \end{array}$$

(c)

$$\begin{array}{llrcrcr} \min & z = & -3x_1 & + & x_2 & -20 & \\ \text{s.t.} & & -3x_1 & + & 3x_2 & \leq & 6 \\ & & -8x_1 & + & 4x_2 & \leq & 4 \\ & & & & \multicolumn{3}{r}{x_1 \geq 0,\ x_2 \geq 0} \end{array}$$

(d)

$$\begin{array}{llrcrcr} \max & z = & 240x_1 & + & 60x_2 & & \\ \text{s.t.} & & 2x_1 & + & 2x_2 & \leq & 100 \\ & & 6x_1 & + & x_2 & \leq & 100 \\ & & 10x_1 & & & \leq & 100 \\ & & & & \multicolumn{3}{r}{x_1 \geq 0,\ x_2 \geq 0} \end{array}$$

(e)

$$\begin{array}{llrcrcr} \min & z = & -3x_1 & - & 4x_2 & & \\ \text{s.t.} & & 2x_1 & + & 3x_2 & \leq & 8 \\ & & & & x_2 & \geq & -2 \\ & & & & \multicolumn{3}{r}{x_1 \geq 0,\ x_2 \in \mathbb{R}} \end{array}$$

(f)

$$\begin{array}{llrcrcr} \max & z = & -5x_1 & + & 7x_2 & & \\ \text{s.t.} & & 2x_1 & + & 9x_2 & = & 13 \\ & & -5x_1 & + & 3x_2 & \leq & 20 \\ & & & & \multicolumn{3}{r}{x_1 \leq 0,\ x_2 \geq 0} \end{array}$$

Exercise 5.2. Indicate for each of the following two statements whether it is true or false.

(a) If an LP problem has no feasible solutions, then its dual is unbounded.

(b) If an LP problem is unbounded, then its dual has no feasible solutions.

Exercise 5.3. Find an example of a primal-dual pair for each of the following combinations:

(a) The primal and dual both have a bounded optimum.

(b) The primal is unbounded and the dual infeasible.

(c) The primal and dual are both infeasible.

Exercise 5.4. Prove that the following primal-dual pairs are not possible:

(a) The dual has a bounded optimum and the primal is unbounded.

(b) The dual has a bounded optimum and the primal is infeasible.

(c) The primal and dual are both unbounded.

Exercise 5.5. Consider an LP problem in standard form and its dual. Give, for each of the following two statements, either a proof or a counter example.

(a) If the primal problem has a unique optimal solution, then the dual problem has a non-degenerate optimal solution.

(b) If the dual problem has a non-degenerate optimal solution, then the primal problem has a unique optimal solution.

Exercise 5.6. Prove Theorem 5.1.

Exercise 5.7. Consider the following linear programming problem:

$$\begin{array}{ll} \min & z = \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} \\ \text{s.t.} & A\boldsymbol{x} \geq \boldsymbol{c} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{array}$$

with A a symmetric $n \times n$ matrix (i.e. $a_{ij} = a_{ji}$ for all $1 \leq i, j \leq n$). Show that if $\boldsymbol{x}^*$ satisfies $A\boldsymbol{x}^* = \boldsymbol{c}$ and $\boldsymbol{x}^* \geq \boldsymbol{0}$ then $\boldsymbol{x}^*$ is an optimal solution of the above linear programming problem.

Chapter 6

Complementary Slackness

Complementary slackness is a way to use duality to characterize optimal solutions. If you have feasible solutions to a linear programming problem and its dual, then complementary slackness basically says that both solutions are optimal if and only if for each inequality constraint holds that either there is no slack/surplus in the constraint or there is no surplus in the nonnegativity constraint of the corresponding variable. Hence the name *complementary slackness.* We now formalize this.

6.1 Theorem and proof

As in the previous section, consider the following primal-dual pair.

The primal problem (P): The dual problem (D):

$$\begin{array}{llll} \min & z = \boldsymbol{c}^\mathsf{T}\boldsymbol{x} & & \\ \text{s.t.} & \boldsymbol{a_i}\boldsymbol{x} = b_i & i \in M \\ & \boldsymbol{a_i}\boldsymbol{x} \geq b_i & i \in \overline{M} \\ & x_j \geq 0 & j \in N \\ & x_j \in \mathbb{R} & j \in \overline{N} \end{array} \qquad \begin{array}{llll} \max & w = \boldsymbol{b}^\mathsf{T}\boldsymbol{\pi} & & \\ \text{s.t.} & \pi_i \in \mathbb{R} & i \in M \\ & \pi_i \geq 0 & i \in \overline{M} \\ & \boldsymbol{\pi}^\mathsf{T}\boldsymbol{A_j} \leq c_j & j \in N \\ & \boldsymbol{\pi}^\mathsf{T}\boldsymbol{A_j} = c_j & j \in \overline{N}. \end{array} \tag{6.1}$$

Theorem 6.1 (Complementary Slackness)**.** If $\boldsymbol{x}$ is a feasible solution of (P) and $\boldsymbol{\pi}$ a feasible solution of (D), then $\boldsymbol{x}$ and $\boldsymbol{\pi}$ are both optimal if and only if:

$$\begin{array}{rll} \pi_i(\boldsymbol{a_i}\boldsymbol{x} - b_i) = 0 & \forall i \in \overline{M} & (CS1) \\ x_j(c_j - \boldsymbol{\pi}^\mathsf{T}\boldsymbol{A_j}) = 0 & \forall j \in N & (CS2) \end{array}$$

Before proving the theorem, we remark the following. Note that condition (CS1) is equivalent to saying that either $\pi_i = 0$ or $\boldsymbol{a_i}\boldsymbol{x} = b_i$ (or both). In words, if a dual variable is positive, then the corresponding primal constraint must be satisfied with equality. Similarly, (CS2) is equivalent to saying that either $x_j = 0$ or $\boldsymbol{\pi}^\mathsf{T}\boldsymbol{A_j} = c_j$ (or both). In words, if a primal variable is positive, then the corresponding dual constraint must be satisfied with equality.

Also note that condition (CS1) trivially holds for $i \in M$ and (CS2) for $j \in \overline{N}$. For this reason it is not necessary to include those in the statement of the theorem.

We will now prove Theorem 6.1.

Proof. First we define

$$u_i := \pi_i(\boldsymbol{a_i}\boldsymbol{x} - b_i)$$
$$v_j := x_j(c_j - \boldsymbol{\pi}^\mathsf{T}\boldsymbol{A_j}),$$

and observe that $u_i \geq 0$ for $1 \leq i \leq m$ and $v_j \geq 0$ for $1 \leq j \leq n$. Next, we define

$$u := \sum_{i=1}^{m} u_i$$
$$v := \sum_{j=1}^{n} v_j.$$

Note that u_i is the left-hand side of (CS1) and v_j the left-hand side of (CS2). Hence, (CS1) is equivalent to stating that $u_i = 0$ for all i and (CS2) is equivalent to stating that $v_j = 0$ for all j. Since all u_i and all v_j are nonnegative, (CS1) and (CS2) both hold if and only if $u + v = 0$. To prove the theorem it remains to show that $u + v = 0$ if and only if $\boldsymbol{x}$ and $\boldsymbol{\pi}$ are optimal solutions (to the primal and dual problem, respectively). We do so as follows.

$$\begin{aligned} u + v &= \sum_{i=1}^{m} [\pi_i(\boldsymbol{a_i x}) - \pi_i b_i] &&+ \sum_{j=1}^{n} \left[x_j c_j - x_j(\boldsymbol{\pi}^\mathsf{T} \boldsymbol{A_j})\right] \\ &= \sum_{i=1}^{m} \pi_i \sum_{j=1}^{n} a_{ij} x_j - \sum_{i=1}^{m} \pi_i b_i &&+ \sum_{j=1}^{n} x_j c_j - \sum_{j=1}^{n} x_j \sum_{i=1}^{m} \pi_i a_{ij} \\ &= -\sum_{i=1}^{m} \pi_i b_i &&+ \sum_{j=1}^{n} x_j c_j \\ &= -w(\boldsymbol{\pi}) &&+ z(\boldsymbol{x}) \end{aligned}$$

Hence, the complementary slackness conditions (CS1) and (CS2) hold if and only if $u + v = 0$, which is the case precisely when $w(\boldsymbol{\pi}) = z(\boldsymbol{x})$, which holds if and only if $\boldsymbol{\pi}$ and $\boldsymbol{x}$ are both optimal (by duality, Theorems 5.2 and 5.3). □

6.2 Applying complementary slackness

In this section, we show how complementary slackness, Theorem 6.1, can be used to quickly find an optimal solution to the dual problem, given an optimal solution to the primal problem, without having to apply the Simplex method. The converse is of course also possible, given an optimal solution to the dual problem, we can find an optimal solution to the primal problem by applying complementary slackness.

Example 6.1. Consider the toy example for the diet problem:

$$\begin{array}{llrcrcrcrl} \min & z = & 3x_1 & + & 3x_2 & + & 4x_3 & & & \\ \text{s.t.} & & 2x_1 & + & x_2 & & & \geq & 3 & (1) \\ & & & & x_2 & + & 4x_3 & \geq & 2 & (2) \\ & & 4x_1 & & & + & 8x_3 & \geq & 9 & (3) \\ & & \multicolumn{5}{l}{x_1,\ x_2,\ x_3 \geq 0} & & & \end{array}$$

and its dual

$$\begin{array}{llrcrcrcr} \max & w = & 3\pi_1 & + & 2\pi_2 & + & 9\pi_3 & & \\ \text{s.t.} & & 2\pi_1 & & & + & 4\pi_3 & \leq & 3 \\ & & \pi_1 & + & \pi_2 & & & \leq & 3 \\ & & & & 4\pi_2 & + & 8\pi_3 & \leq & 4 \\ & & \multicolumn{5}{l}{\pi_1, \pi_2, \pi_3 \geq 0} & & \end{array}$$

Suppose we are given an optimal solution $\boldsymbol{\pi}^* = (1\frac{1}{2}, 1, 0)$ to the dual problem, with optimal value $w^* = 6\frac{1}{2}$. Then we can find an optimal solution to the primal problem as follows.

By Theorem 6.1, (CS2), we know that in an optimal solution to the primal problem $x_j^* = 0$ whenever the jth dual constraint has positive slack. Hence, we substitute the given dual solution into the dual constraints.

The slack in the first dual constraint is

$$3 - 2\pi_1^* - 4\pi_3^* = 3 - 3 - 0 = 0,$$

which does not imply anything for x_1^*.

The slack in the second dual constraint is

$$3 - \pi_1^* - \pi_2^* = 3 - 1\frac{1}{2} - 1 = \frac{1}{2} > 0,$$

which implies that $x_2^* = 0$.

The slack in the third dual constraint is

$$4 - 4\pi_2^* - 8\pi_3^* = 4 - 4 - 0 = 0,$$

which does not imply anything for x_3^*.

Now we use (CS1) to find the values of x_1^* and x_3^*. This part of Theorem 6.1 tells us that when $\pi_i^* > 0$, the ith primal constraint needs to be satisfied with equality.

Since $\pi_1^* = 1\frac{1}{2} > 0$, the first primal constraint is satisfied with equality:

$$2x_1^* + x_2^* = 3$$

and since we already knew that $x_2^* = 0$, we now also know that $x_1^* = 1\frac{1}{2}$.

Finally, since $\pi_2^* = 1 > 0$, the second primal constraint is satisfied with equality:

$$x_2^* + 4x_3^* = 2$$

and using that $x_2^* = 0$ we now find that $x_3^* = \frac{1}{2}$. Hence, we have found an optimal solution $\boldsymbol{x}^* = (1\frac{1}{2}, 0, \frac{1}{2})$ to the primal problem. □

6.3 Geometrical interpretation

Consider the following form of an LP, which is more suitable for geometrical interpretations.

$$\begin{array}{ll} \max & z = \boldsymbol{c}^\mathsf{T}\boldsymbol{x} \\ \text{s.t.} & A\boldsymbol{x} \le \boldsymbol{b} \\ & \boldsymbol{x} \in \mathbb{R}^n \end{array}$$

As we have seen in Chapter 3, if the LP has an optimal solution, it has some optimal solution $\boldsymbol{x}^*$ that is a vertex of the polyhedron

$$P = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \le \boldsymbol{b}\}.$$

Let z^* denote the optimal value $z(\boldsymbol{x}^*)$. The hyperplane $H_{\boldsymbol{c},z^*} = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{c}^\mathsf{T}\boldsymbol{x} = z^*\}$ is a supporting hyperplane of P and $H_{\boldsymbol{c},z^*} \cap P = \boldsymbol{x}$. Hence, the equality $\boldsymbol{c}^\mathsf{T}\boldsymbol{x} = z^*$ is a nonnegative linear combination of constraints of the form $\boldsymbol{a_i}\boldsymbol{x} = b_i$. I.e., there exist $\pi_1^*, \ldots, \pi_m^* \ge 0$ such that

$$\begin{array}{ccccccc} \pi_1^*\boldsymbol{a}_1 & + & \ldots & + & \pi_m^*\boldsymbol{a}_m & = & \boldsymbol{c}^\mathsf{T} \\ \pi_1^* b_1 & + & \ldots & + & \pi_m^* b_m & = & z^*. \end{array}$$

Hence, $\boldsymbol{\pi}^*$ is an optimal solution to the dual problem:

$$\begin{array}{llll} \max \boldsymbol{c}^\mathsf{T}\boldsymbol{x} & = z^* = & \min \boldsymbol{\pi}^\mathsf{T}\boldsymbol{b} & \\ \text{s.t.} \quad A\boldsymbol{x} \le \boldsymbol{b} & & \text{s.t.} \ \boldsymbol{\pi}^\mathsf{T} A = \boldsymbol{c}^\mathsf{T} & \\ \qquad \boldsymbol{x} \in \mathbb{R}^n & & \qquad \boldsymbol{\pi} \ge \boldsymbol{0} & . \end{array}$$

By complementary slackness, Theorem 6.1, $\pi_i^* = 0$ whenever $\boldsymbol{a_i x} < b_i$. In other words, the hyperplane $H_{\boldsymbol{c},z^*}$ is a nonnegative linear combination of hyperplanes corresponding to inequalities that are satisfied with equality by $\boldsymbol{x}^*$, see Figure 6.1.

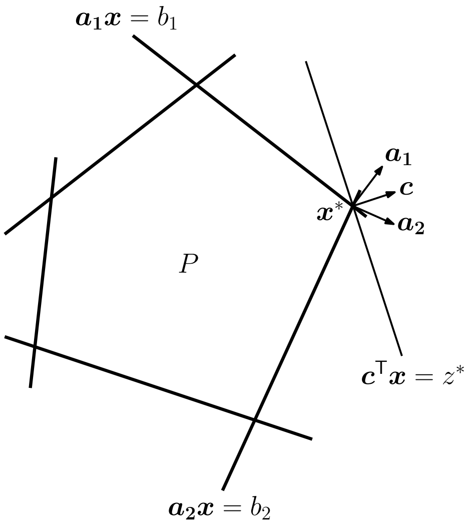

$\boldsymbol{a}_1\boldsymbol{x} = b_1$
$\boldsymbol{a}_1$
$\boldsymbol{c}$
$\boldsymbol{x}^*$
$\boldsymbol{a}_2$
P
$\boldsymbol{c}^\mathsf{T}\boldsymbol{x} = z^*$
$\boldsymbol{a}_2\boldsymbol{x} = b_2$

Figure 6.1: Illustration of the geometrical interpretation of complementary slackness. Hyperplane $\boldsymbol{c}^\mathsf{T}\boldsymbol{x} = z^*$ is a nonnegative linear combination of hyperplanes $\boldsymbol{a}_1\boldsymbol{x} = b_1$ and $\boldsymbol{a}_2\boldsymbol{x} = b_2$, which correspond to the constraints that are satisfied with equality in an optimal solution $\boldsymbol{x}^*$.

6.4 Application to shortest paths

In this section, we consider the following optimization problem.

Shortest Path
Given: directed graph $D = (V, A)$ with a length $\ell(a)$ for each arc $a \in A$ and two special vertices $s, t \in V$.
Find: a path P from s to t in D that minimizes

$$\ell(P) = \sum_{a \in A(P)} \ell(a),$$

with $A(P)$ the set of arcs that lie on path P.

To formulate this problem as an ILP problem, we describe the directed graph $D = (V, A)$ using its *incidence matrix* M, which is the matrix with a row for each vertex $v \in V$, a column for each arc $a \in A$, and the element in the row for v and column for a equal to:

$$M_{va} = \begin{cases} 1 & \text{if vertex } v \text{ is the tail of arc } a \\ -1 & \text{if vertex } v \text{ is the head of arc } a \\ 0 & \text{otherwise.} \end{cases}$$

We assume that the first row of A corresponds to s and the last row to t. In addition, we define the parameters $n = |V|$, $m = |A|$ and the vector $\boldsymbol{c}$ containing the length $\ell(a)$ of each arc $a \in A$.

The next step is to introduce the decision variables. We introduce a binary variable f_a for each arc $a \in A$, which indicates whether the arc is used by the path (in which case $f_a = 1$) or not ($f_a = 0$). Together, these binary variables form a vector $\boldsymbol{f}$. The ILP formulation of the Shortest Path problem is now as follows.

$$\begin{array}{ll} \min & z = \boldsymbol{c}^\mathsf{T}\boldsymbol{f} \\ \text{s.t.} & M\boldsymbol{f} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ -1 \end{bmatrix} \\ & \boldsymbol{f} \in \{0,1\}^m \end{array} \tag{6.2}$$

Moreover, we claim that we may relax the constraint $\boldsymbol{f} \in \{0,1\}^m$ to $\boldsymbol{f} \geq \boldsymbol{0}$, thus obtaining the following LP formulation of the Shortest Path problem.

$$\begin{array}{ll} \min & z = \boldsymbol{c}^\mathsf{T}\boldsymbol{f} \\ \text{s.t.} & M\boldsymbol{f} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ -1 \end{bmatrix} \\ & \boldsymbol{f} \geq \boldsymbol{0} \end{array} \tag{6.3}$$

The equivalence of the LP and ILP formulations follow from the next observation (see Exercise 6.5).

Observation 6.1. At least one optimal solution to (6.3) is binary.

In addition, we observe that the equality constraints are not independent. Therefore, we may omit one constraint, for example the last row corresponding to vertex t. Indeed, a solution is a path that leaves s and whenever it enters a vertex from $V \setminus \{s,t\}$ it must also leave that vertex again. Hence, it must eventually arrive at t. On the other hand, we may also keep all constraints, which we will do here.

The next step is to formulate the dual problem, which has a dual variable π_v for each vertex $v \in V$, together forming a vector $\boldsymbol{\pi}$, and a constraint for each arc $a \in A$. The variables are free variables since the primal problem has equality constrains. In addition, the constrains are $\leq$-constraints since the primal problem is a minimization problem with nonnegative variables.

$$\begin{array}{ll} \max & w = \pi_s - \pi_t \\ \text{s.t.} & \boldsymbol{\pi}^\mathsf{T} M \leq \boldsymbol{c} \\ & \boldsymbol{\pi} \in \mathbb{R}^m \end{array} \tag{6.4}$$

Noting that each column of matrix M has exactly two non-zero values (a 1 for the tail and a -1 for the head of the arc), we can reformulate the dual problem as follows.

$$\begin{array}{ll} \max & w = \pi_s - \pi_t \\ \text{s.t.} & \pi_u - \pi_v \leq \ell(a) \quad \forall a = (u,v) \in A \\ & \boldsymbol{\pi} \in \mathbb{R}^m \end{array} \tag{6.5}$$

We are now ready to use complementary slackness to characterize optimal solutions of the Shortest Path problem. Theorem 6.1 tells us that a solution $\boldsymbol{f}$ to (6.3) and $\boldsymbol{\pi}$ to (6.5) are both optimal if and only if $f_a = 0$ for each arc $a = (u, v) \in A$ with $\pi_u - \pi_v < \ell(a)$. Equivalently, whenever $f_a = 1$ we need to have $\pi_u - \pi_v = \ell(a)$ for this arc $a = (u, v)$. Note that the first part of Theorem 6.1 (CS1) does not give us anything in this case because the primary constrains are all equality constraints and can hence never have positive slack.

In other words, complementary slackness tells us that a given s-t path P is optimal if and only if one can assign a value π_v to each vertex v such that

1. $\pi_u - \pi_v \le \ell(a)$ for each arc $a = (u, v)$ (the dual constraints), and
2. $\pi_u - \pi_v = \ell(a)$ for each arc $a = (u, v)$ on P (complementary slackness).

Finally, note that complementary slackness does *not* tell us anything for arcs that are not on the path, so for such arcs we have have $\pi_u - \pi_v = \ell(a)$ or $\pi_u - \pi_v < \ell(a)$.

Example 6.2. Consider the directed graph in Figure 6.2.

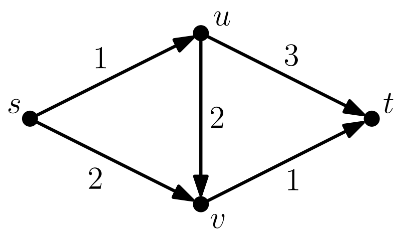


Figure 6.2: An instance of the SHORTEST PATH problem.

Its incidence matrix is

$$M = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 \\ 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & -1 \end{bmatrix},$$

assuming we order the vertices as s, u, v, t. The LP formulation of the SHORTEST PATH problem is then as follows, using the notation f_{uv} for f_a when $a = (u, v)$,

$$\begin{array}{llrrrrrcrl}
\min & z = & f_{su} & +\ 2f_{sv} & +\ 2f_{uv} & +\ 3f_{ut} & +\ f_{vt} & & & \\
\text{s.t.} & & f_{su} & +\ f_{sv} & & & & = & 1 & (s) \\
& & -f_{su} & & +\ f_{uv} & +\ f_{ut} & & = & 0 & (u) \\
& & & -f_{sv} & -f_{uv} & & +\ f_{vt} & = & 0 & (v) \\
& & & & & -f_{ut} & -f_{vt} & = & -1 & (t) \\
& & \multicolumn{5}{l}{f_{su},\ f_{sv},\ f_{uv},\ f_{ut},\ f_{vt} \ge 0} & & &
\end{array}$$

where the constraint for t could be omitted. The dual problem is as follows.

$$\begin{array}{llrrrrcl}
\max & w = & \pi_s & & & -\pi_t & & \\
\text{s.t.} & & \pi_s & -\pi_u & & & \le & 1 \\
& & \pi_s & & -\pi_v & & \le & 2 \\
& & & \pi_u & -\pi_v & & \le & 2 \\
& & & \pi_u & & -\pi_t & \le & 3 \\
& & & & \pi_v & -\pi_t & \le & 1
\end{array}$$

Omitting the row for t from the primal problem corresponds to omitting the column for t in the dual problem.

We will now show how complementary slackness can be used to find an optimal solution to the dual problem when an optimal solution to the primal problem is given.

It is not difficult to find an optimal solution to the primal problem. Clearly, the unique shortest path from s to t is the path (s, v, t) of length 3.

As the column of t is redundant, we can assume without loss of generality that $\pi_t = 0$.

Since the path uses arc (s, v), we have $f_{sv} = 1 > 0$ and hence, by Theorem 6.1, $\pi_s - \pi_v = 2$.

Similarly, since $f_{vt} > 0$, we have $\pi_v - \pi_t = 1$. As $\pi_t = 0$, it follows that $\pi_v = 1$.

By substituting $\pi_v = 1$ into the equality $\pi_s - \pi_v = 2$, we conclude that $\pi_s = 3$.

Complementary slackness does not give us any information about the value of π_u. However, this value does of course need to satisfy the dual constraints.

From the constraint $\pi_s - \pi_u \leq 1$ we learn that $\pi_u \geq 2$. From the constraint $\pi_u - \pi_v \leq 3$, and also from the constraint $\pi_u - \pi_t \leq 3$, we learn that $\pi_u \leq 3$. Hence, any choice of π_u with $2 \leq \pi_u \leq 3$ will satisfy the dual constraints.

We can conclude that the optimal solutions to the dual problem, assuming $\pi_t = 0$, are given by $\pi_s = 3$, $2 \leq \pi_u \leq 3$ and $\pi_v = 1$. □

Conversely, we can also use a given optimal solution to the dual problem to find an optimal solution to the primal problem. This is left as an exercise.

6.5 Exercises

Exercise 6.1. Consider the following LP problem.

$$\begin{array}{llrcrcrcrcrcr}
\max & z = & & - & 4x_2 & + & 3x_3 & + & 2x_4 & - & 8x_5 & & \\
\text{s.t.} & & 3x_1 & + & x_2 & + & 2x_3 & + & x_4 & & & = & 3 \\
& & x_1 & - & x_2 & & & + & x_4 & - & x_5 & \geq & 2 \\
& & \multicolumn{11}{l}{x_1, \ldots, x_5 \geq 0}
\end{array}$$

a) Give the corresponding dual problem.

b) Solve the dual problem graphically.

c) Use Complementary Slackness to find an optimal solution to the primal problem.

Exercise 6.2. Consider the following LP problem.

$$\begin{array}{lrcrcrcrcr}
\min z = & -4x_1 & - & 7x_2 & + & 5x_3 & - & 14x_4 & & \\
\text{s.t.} & 2x_1 & - & 4x_2 & + & x_3 & - & 8x_4 & \leq & 22 \\
& x_1 & + & x_2 & + & 3x_3 & + & x_4 & \leq & 8 \\
& \multicolumn{9}{l}{x_1, \ldots, x_4 \geq 0}
\end{array}$$

a) Is $(\pi_1, \pi_2) = (2, 1)$ a feasible solution of the dual (D) of the above LP?

b) Is $(\pi_1, \pi_2) = (2, 1)$ a basic solution of (D)?

c) Find an optimal solution to (D).

d) Use Complementary Slackness to find an optimal solution to the primal LP problem.

Exercise 6.3. Consider the following LP problem.

$$\begin{array}{lrrrl}
\min & 2\,x_1 & +3\,x_2 & +4\,x_3 & \\
\text{s.t.} & 2\,x_1 + & x_2 & & = 3 \\
 & & x_2 & +4\,x_3 & \leq 2 \\
 & \multicolumn{4}{c}{x_1, x_2, x_3 \geq 0}
\end{array}$$

a) Give the corresponding dual problem.

b) Solve the dual problem using the Simplex method.

c) Use Complementary Slackness to find an optimal solution to the primal problem.

Exercise 6.4. Consider an LP problem in standard form. Assume that it has an optimal solution and that no optimal solution is degenerate. Prove that the dual problem has a unique optimal solution.

Exercise 6.5. Prove Observation 6.1, i.e., show that at least one optimal solution to the LP formulation (6.3) of the Shortest Path problem is binary.

Chapter 7

Farkas' Lemma

Although Farkas' lemma is called a lemma, it is actually seen as an important theorem at the basis of linear programming duality. It was proven by the Hungarian mathematician Gyula Farkas in 1902. Where complementary slackness can be seen as a characterization of optimality, Farkas' lemma can be seen as a characterization of feasibility. It basically states that a given system of (in)equalities has a solution if and only if some other system of (in)equalities does *not* have a solution. Hence, presenting a feasible solution to one of these systems can be used as a certificate for showing that the other system is infeasible. There are many different forms of the lemma, with, for example, different types of (in)equalities. We will prove the currently most widely used form in the next section. Other variants of Farkas' lemma can then be derived by applying this theorem or by adapting its proof.

7.1 Farkas' lemma and its proof

Theorem 7.1 (Farkas' Lemma)**.** For every $m \times n$ matrix A and every vector $\boldsymbol{b} \in \mathbb{R}^m$, exactly one of the following two statements is true:

(i) $\exists \boldsymbol{x} \in \mathbb{R}^n$ such that $A\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{x} \geq \boldsymbol{0}$; or

(ii) $\exists \boldsymbol{y} \in \mathbb{R}^m$ such that $\boldsymbol{y}^\mathsf{T} A \geq \boldsymbol{0}^\mathsf{T}$ and $\boldsymbol{y}^\mathsf{T}\boldsymbol{b} < 0$.

Proof. We will prove the lemma by applying the duality theorems to the following primal-dual pair.

The primal problem (P):

$$\begin{array}{ll} \max & z = \boldsymbol{0}^\mathsf{T}\boldsymbol{x} \\ \text{s.t.} & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{array}$$

The dual problem (D):

$$\begin{array}{ll} \min & w = \boldsymbol{b}^\mathsf{T}\boldsymbol{y} \\ \text{s.t.} & A^\mathsf{T}\boldsymbol{y} \geq \boldsymbol{0} \\ & \boldsymbol{y} \in \mathbb{R}^m \end{array}$$

Note that $A^\mathsf{T}\boldsymbol{y} \geq \boldsymbol{0}$ is equivalent to $\boldsymbol{y}^\mathsf{T} A \geq \boldsymbol{0}^\mathsf{T}$ (in statement (ii) of the lemma).

The proof consists of two parts. First we show that if statement (i) holds, statement (ii) cannot hold. Hence, assume that statement (i) holds, i.e. there exists an $\boldsymbol{x} \in \mathbb{R}^n$ such that $A\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{x} \geq \boldsymbol{0}$. This means that $\boldsymbol{x}$ is a feasible solution to problem (P). Since the objective function of (P) is always 0, the optimal value is $z^* = 0$. It then follows by Theorem 5.3 that problem (D) also has an optimal solution with optimal value $w^* = z^* = 0$. If the optimal value of the minimization problem (D) is 0, then every feasible solution will have an objective value that is at least 0. In other words, for every $\boldsymbol{y} \in \mathbb{R}^m$ with $A^\mathsf{T}\boldsymbol{y} \geq \boldsymbol{0}$, we have that $\boldsymbol{b}^\mathsf{T}\boldsymbol{y} \geq 0$. This means that there exists no $\boldsymbol{y} \in \mathbb{R}^m$ with $A^\mathsf{T}\boldsymbol{y} \geq \boldsymbol{0}$ and $\boldsymbol{b}^\mathsf{T}\boldsymbol{y} < 0$, i.e., statement (ii) does not hold.

At this point, we have shown that statements (i) and (ii) cannot both hold simultaneously. However, we haven't shown yet that at least one of the two statements needs to hold. We will now prove this, by showing that if statement (i) does not hold, statement (ii) needs to hold.

Assume that statement (i) does not hold, i.e., there exists no $\boldsymbol{x} \in \mathbb{R}^n$ with $A\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{x} \geq \boldsymbol{0}$. This means that problem (P) has no feasible solution. Then, by Theorem 5.3, there are two possibilities for problem (D): either it is also infeasible, or it is unbounded. However, note that it is not possible that (D) is infeasible because $\boldsymbol{y} = \boldsymbol{0}$ is always a feasible solution. Hence, we conclude that (D) is unbounded. This means that there are feasible solutions with arbitrarily low objective function values. In particular, there must be feasible solutions with negative objective function values. Hence, there exists an $\boldsymbol{y} \in \mathbb{R}^m$ with $A^\mathsf{T}\boldsymbol{y} \geq \boldsymbol{0}$ and $\boldsymbol{b}^\mathsf{T}\boldsymbol{y} < 0$, i.e. statement (ii) does hold. □

While we have proven Farkas' lemma here using duality theory, it is important to note that Farkas' lemma already dates from 1902 and is hence much older than linear programming duality. In fact, Farkas' lemma was used in the original proof of the duality theorems.

7.2 Geometric view

There are different forms of Farkas' lemma. We now derive a geometric variant in terms of cones and angles.

The *cone* spanned by n vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ is defined as the set of all nonnegative linear combinations of these vectors:

$$\text{cone}(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n) = \{\lambda_1 \boldsymbol{u}_1 + \ldots + \lambda_n \boldsymbol{u}_n \mid \lambda_1, \ldots, \lambda_n \geq 0\}.$$

We use the notation $\angle(\boldsymbol{x}, \boldsymbol{y})$ to denote the angle between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$.

Theorem 7.2. If $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n, \boldsymbol{b}$ are vectors in $\mathbb{R}^m$, then exactly one of the following two statements is true:

(a) $\boldsymbol{b} \in \text{cone}(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n)$; or

(b) there exists a vector $\boldsymbol{y} \in \mathbb{R}^m$ such that $\angle(\boldsymbol{y}, \boldsymbol{b}) > \frac{1}{2}\pi$ and $\angle(\boldsymbol{y}, \boldsymbol{u}_i) \leq \frac{1}{2}\pi$ for all $1 \leq i \leq n$.

Proof. Let A be the matrix with columns $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ and apply Theorem 7.1.

We first show that statement (a) is equivalent to statement (i) of Theorem 7.1:

$$\begin{aligned} \boldsymbol{b} \in \text{cone}(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n) &\Leftrightarrow \exists \lambda_1, \ldots, \lambda_n \geq 0 : \boldsymbol{b} = \lambda_1 \boldsymbol{u}_1 + \ldots + \lambda_n \boldsymbol{u}_n \\ &\Leftrightarrow \exists x_1, \ldots, x_n \geq 0 : \boldsymbol{b} = x_1 \boldsymbol{u}_1 + \ldots + x_n \boldsymbol{u}_n \\ &\Leftrightarrow \exists \boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{b} = A\boldsymbol{x}, \boldsymbol{x} \geq \boldsymbol{0} \end{aligned}$$

Next we show that statement (b) is equivalent to statement (ii) of Theorem 7.1. Indeed, $\angle(\boldsymbol{y}, \boldsymbol{b}) > \frac{1}{2}\pi$ is equivalent to $\boldsymbol{y}^\mathsf{T}\boldsymbol{b} < 0$. In addition, $\angle(\boldsymbol{y}, \boldsymbol{u}_i) \leq \frac{1}{2}\pi$ is equialent to $\boldsymbol{y}^\mathsf{T}\boldsymbol{u}_i \geq 0$. Hence, $\angle(\boldsymbol{y}, \boldsymbol{u}_i) \leq \frac{1}{2}\pi$ for all $1 \leq i \leq n$ is equivalent to $\boldsymbol{y}^\mathsf{T} A \geq \boldsymbol{0}^\mathsf{T}$. □

This form of Farkas' lemma is easy to interpret geometrically, see Figure 7.1. The cone spanning vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_5$ is indicated by the grey area (including the black boundaries). If, as in the figure, vector $\boldsymbol{b}$ is not in this cone, then it can be separated from the cone by a linear hyperplane (the dash-dotted line). Then the vector $\boldsymbol{y}$ through $\boldsymbol{0}$ perpendicular to this hyperplane satisfies the conditions in statement (b) of Theorem 7.2. On the other hand, when $\boldsymbol{b}$ is in the cone, then it is not difficult to imagine (at an intuitive level) that there is no such separating linear hyperplane and therefore no vector $\boldsymbol{y}$ satisfying the conditions.

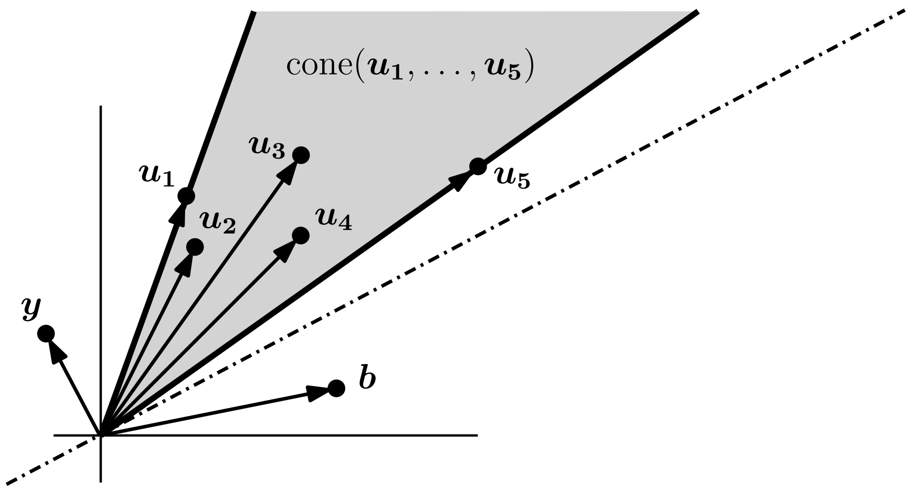


Figure 7.1: Geometric view of Farkas' lemma.

7.3 Exercises

Exercise 7.1. Prove the following variant of Farkas' lemma using the duality theorems.

For each $m \times n$ matrix A and vector $\boldsymbol{b} \in \mathbb{R}^m$, the following two statements are equivalent:

(i) the system of inequalities $A\boldsymbol{x} \leq \boldsymbol{b}$ is infeasible;

(ii) there exists a vector $\boldsymbol{y} \in \mathbb{R}^m$ satisfying $\boldsymbol{y}^\mathsf{T} A = \mathbf{0}^\mathsf{T}$, $\boldsymbol{y}^\mathsf{T}\boldsymbol{b} < 0$ and $\boldsymbol{y} \geq \mathbf{0}$.

Exercise 7.2. Fredholm alternative can be formulated as follows.

For every $m \times n$ matrix A and vector $\boldsymbol{b} \in \mathbb{R}^m$, exactly one of the following two statements is true:

(1) $\exists \boldsymbol{x} \in \mathbb{R}^n$ such that $A\boldsymbol{x} = \boldsymbol{b}$;

(2) $\exists \boldsymbol{y} \in \mathbb{R}^m$ such that $\boldsymbol{y}^\mathsf{T} A = \mathbf{0}^\mathsf{T}$ and $\boldsymbol{y}^\mathsf{T}\boldsymbol{b} = 1$.

Derive Fredholm alternative from Theorem 7.1.

Hint: substitute $\boldsymbol{x} = \boldsymbol{x}^+ - \boldsymbol{x}^-$ in order to get nonnegative variables $\boldsymbol{x}^+, \boldsymbol{x}^- \geq \mathbf{0}$ and then apply Theorem 7.1.

Exercise 7.3. Show that, for every $m \times n$ matrix A, every $m \times k$ matrix B and vector $\boldsymbol{b} \in \mathbb{R}^m$, exactly one of the following two statements is true:

(1) $\exists \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^k$ such that $A\boldsymbol{x} + B\boldsymbol{y} = \boldsymbol{b}$ and $\boldsymbol{x} \geq \mathbf{0}$;

(2) $\exists \boldsymbol{\pi} \in \mathbb{R}^m$ such that $\boldsymbol{\pi}^\mathsf{T} A \geq \mathbf{0}^\mathsf{T}$, $\boldsymbol{\pi}^\mathsf{T} B = \mathbf{0}^\mathsf{T}$ and $\boldsymbol{\pi}^\mathsf{T}\boldsymbol{b} < 0$.

Chapter 8

The Dual Simplex Method

As we have seen in Chapter 5, there is a clear relation between the primal optimality conditions and dual feasibility. Suppose we have a primal minimization problem:

$$\min z = \boldsymbol{c}^\top \boldsymbol{x} \tag{8.1}$$
$$\text{s.t. } A\boldsymbol{x} \geq \boldsymbol{b} \tag{8.2}$$
$$\boldsymbol{x} \geq \boldsymbol{0}\,. \tag{8.3}$$

In an optimal solution, all reduced costs are nonnegative. The expression for reduced costs, $\bar{\boldsymbol{c}}$, is:

$$\bar{\boldsymbol{c}}^\top = \boldsymbol{c}^\top - \boldsymbol{c}_B^\top B^{-1} A\,,$$

where B is the submatrix of A corresponding to the basic variables in optimum, and $\boldsymbol{c}_B$ is the objective vector for the basic variables, see also Sections 5.3 and 5.5.

The optimality conditions $\bar{\boldsymbol{c}}^\top \geq \boldsymbol{0}$ implies

$$\boldsymbol{c}^\top - \boldsymbol{c}_B^\top B^{-1} A \geq \boldsymbol{0}\,, \text{ or equivalently } \boldsymbol{c}_B^\top B^{-1} A \leq \boldsymbol{c}^\top\,. \tag{8.4}$$

Next, we consider the dual problem corresponding to (8.1)–(8.3).

$$\max w = \boldsymbol{b}^\top \boldsymbol{\pi} \tag{8.5}$$
$$\text{s.t. } A^\top \boldsymbol{\pi} \leq \boldsymbol{c} \tag{8.6}$$
$$\boldsymbol{\pi} \geq \boldsymbol{0} \tag{8.7}$$

Compare the expressions (8.4) and (8.6) and observe that $\boldsymbol{\pi}^\top = \boldsymbol{c}_B^\top B^{-1}$. We can then conclude that primal optimality is equivalent to dual feasibility. Given the primal-dual relationship, it of course also holds that dual optimality equals primal feasibility.

In the dual Simplex algorithm we start with a primal Simplex tableau in which the signs of all reduced costs are in accordance with primal optimality, but where one or more primal basic variables take negative values. The way we can view this is from the dual side; namely the dual problem is feasible, but does not satisfy the optimality conditions. So, even if we work in the primal Simplex tableau, we actually solve the dual problem.

Dual Simplex can for some problems be better to apply than the standard primal Simplex in order to solve an LP problem. The way we will use dual Simplex is in order to re-optimize a problem once a constraint has been added to the primal problem and this constraint cuts off the current LP-optimum. Examples when this occurs is for instance in Branch and Bound (Chapter 13) and in Cutting Plane algorithms (Chapter 14), which are the two most important approaches for solving integer linear problems.

We will illustrate the dual Simplex algorithm in an example.

Example 8.1. We consider the LP from Example 4.1:

$$\begin{array}{llrcrcrl}
\max & z = & 50x_1 & + & 45x_2 & & & \\
\text{s.t.} & & 6x_1 & + & 5x_2 & \leq & 60 & (1) \\
& & x_1 & + & 2x_2 & \leq & 15 & (2) \\
& & x_1 & & & \leq & 8 & (3) \\
& & \multicolumn{5}{l}{x_1,\ x_2 \geq 0} &
\end{array}$$

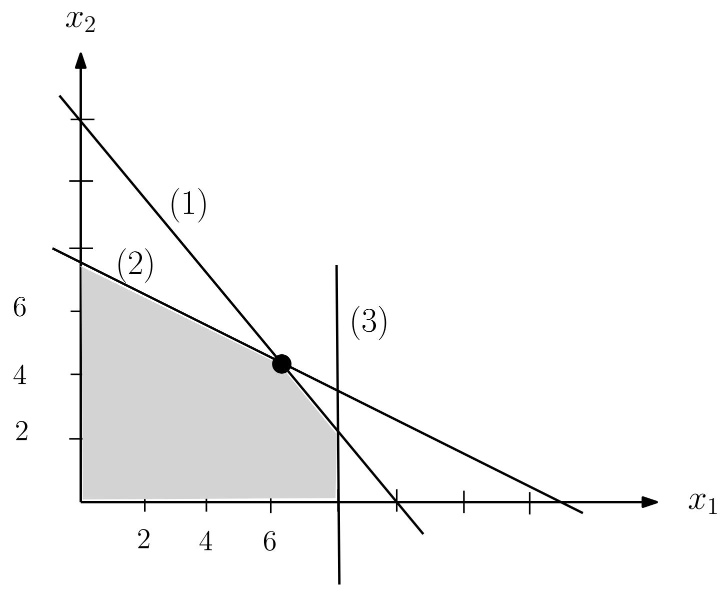


Figure 8.1: The polytope associated with the example, and the optimal solution.

The optimal solution in the Simplex tableau form is:

basis	$\bar{b}$	x_1	x_2	s_1	s_2	s_3
x_2	$4\frac{2}{7}$		1	$-\frac{1}{7}$	$\frac{6}{7}$	
s_3	$\frac{11}{7}$			$-\frac{2}{7}$	$\frac{5}{7}$	1
x_1	$6\frac{3}{7}$	1		$\frac{2}{7}$	$-\frac{5}{7}$	
$-z$	$-514\frac{2}{7}$			$-7\frac{6}{7}$	$-\frac{20}{7}$	

Suppose that we want to add the inequality $x_1 \leq 6$ and we want to re-optimize the problem, as the new constraint is violated by the current optimum. We start by putting the new constraint is equality form:

$$x_1 + s_4 = 6 \quad \text{with } s_4 \geq 0\,. \tag{8.8}$$

Since x_1 is basic, we need to express our new constraint in non-basic variables. From the third row in the tableau, which is the row in which x_1 is basic, we read:

$$x_1 + \frac{2}{7}s_1 - \frac{5}{7}s_2 = 6\frac{3}{7}, \text{ or, } x_1 = 6\frac{3}{7} - \frac{2}{7}s_1 + \frac{5}{7}s_2\,.$$

We use this expression to substitute for x_1 in (8.8) to obtain:

$$-\frac{2}{7}s_1 + \frac{5}{7}s_2 + s_4 = -\frac{3}{7}\,.$$

This row can now be added to the Simplex tableau, which gives:

basis	$\bar{b}$	x_1	x_2	s_1	s_2	s_3	s_4
x_2	$4\frac{2}{7}$		1	$-\frac{1}{7}$	$\frac{6}{7}$		
s_3	$\frac{11}{7}$			$-\frac{2}{7}$	$\frac{5}{7}$	1	
x_1	$6\frac{3}{7}$	1		$\frac{2}{7}$	$-\frac{5}{7}$		
s_4	$-\frac{3}{7}$			$-\frac{2}{7}$	$\frac{5}{7}$		1
$-z$	$-514\frac{2}{7}$			$-7\frac{6}{7}$	$-\frac{20}{7}$		

Since s_4 is required to be nonnegative, the current basic solution is primal infeasible. All reduced costs do however have the correct sign for primal optimality, which is equivalent to dual feasibility. We will maintain dual feasibility, and iterate to obtain primal feasibility.

Since all reduced costs have the right sign, it is not so obvious how to choose the entering basic variable. We therefore start by choosing the leaving basic variable as the one that has negative value, s_4. The pivot row is now known, row 4. To maintain dual feasibility we only pivot on a negative pivot element, and we then have the following version of the "min-ratio rule" that we used in primal Simplex:

$$\min\{\frac{\bar{c}_j}{\bar{a}_{4j}} \mid \bar{a}_{4j} < 0\} = \min\{\frac{-7\frac{6}{7}}{-\frac{2}{7}}\} = \frac{55}{2},$$

which means that s_1 is the entering variable. The pivot element is $\bar{a}_{s_4,s_1} = -\frac{2}{7}$. After pivoting, the Simplex tableau looks as follows:

basis	$\bar{b}$	x_1	x_2	s_1	s_2	s_3	s_4
x_2	$4\frac{1}{2}$		1		$\frac{1}{2}$		$-\frac{1}{2}$
s_3	2					1	-1
x_1	6	1					1
s_1	$\frac{3}{2}$			1	$-\frac{5}{2}$		$-\frac{7}{2}$
$-z$	$-502\frac{1}{2}$				$-22\frac{1}{2}$		$-27\frac{1}{2}$

The new optimal solution is

$$\begin{pmatrix} x_1^* \\ x_2^* \end{pmatrix} = \begin{pmatrix} 6 \\ 4\frac{1}{2} \end{pmatrix}, \quad \text{with } z^* = 502\frac{1}{2}.$$

□

We make three remarks before we formalize the dual Simplex method.

1. In the example we mention that we only pivot on a negative pivot element. This is in contrast to the primal Simplex method where we only pivot on a positive element. The reason why is as follows. If all elements in the row of the leaving variable are positive, the problem has to be infeasible. The pivot row can be written as: $x_l + \sum_{j=n-m}^{n} \bar{a}_{lj}x_j = \bar{b}_l$. Since $x_j \geq 0$ in all feasible solutions, it is impossible to get a $\bar{b}_l < 0$ if all $\bar{a}_{lj} \geq 0$.

2. In our example we considered a maximization problem, so all reduced costs are nonpositive. The pivot element is negative, so the ratio in the ratio test will be nonnegative. If we had

considered a minimization problem, the reduced costs would have been nonnegative which gives a nonpositive ratio in the ratio test and we would have had the following ratio test:

$$k = \text{argmax}_j\{\frac{\bar{c}_j}{\bar{a}_{lj}} \mid \bar{a}_{lj} < 0\}\,.$$

To simplify the description to avoid different rules for min- and max-problems, we simply take the absolute value in the ratio test, see Step 2 below.

Initialize: Given is a Simplex tableau in which all reduced costs are nonpositive (max problem) or nonnegative (min problem).

Step 1: Choose *leaving basic variable* x_l with $\bar{b}_l < 0$. If $\bar{b}_i \geq 0$ for all i, the current basic solution is optimal, stop.

Step 2: Choose *entering basic variable* x_k. Perform the min ratio test:

$$k = \text{argmin}_j\{|\frac{\bar{c}_j}{\bar{a}_{lj}}| \mid \bar{a}_{lj} < 0\}$$

If $\bar{a}_{lj} \geq 0$ for all j, then the problem is infeasible, stop. Otherwise, pivot on the element $\bar{a}_{lk}$, i.e., apply row operations to the current system such that element $\{l, k\}$ takes value $+1$, and such that all other elements in column k take value zero. Go to Step 1.

Remark 8.1. In Step 1, we have not specified how to choose leaving variable if more than one basic variable has negative value. As in primal Simplex, one specific choice is not provably better in general than all other choices, but a simple rule is to choose the variable with the smallest value.

8.1 Exercises

Exercise 8.1. Consider the following LP.

$$\begin{array}{llrcrcr}
\min & z = & 3x_1 & - & 6x_2 & & \\
\text{s.t.} & & 5x_1 & + & 7x_2 & \leq & 35 \\
 & & -x_1 & + & 2x_2 & \leq & 2 \\
 & & \multicolumn{5}{c}{x_1 \geq 0,\; x_2 \geq 0\,.}
\end{array}$$

In Exercise 4.4, we found that $x_1 = 0$, $x_2 = 1$ is an optimal solution to this LP. Find the corresponding Simplex tableau by introducing slack variables s_1, s_2 and applying row operations in order to get the right basic variables.

Now suppose that the constraint $2x_2 \leq 1$ is added to the problem. The current solution is not feasible any more. Find an optimal solution of the LP by adding the constraint (with a new slack variable) to the current tableau and applying the dual simplex method.

Exercise 8.2. Consider the following LP.

$$\begin{array}{llrcrcr}
\min & z = & 2x_1 & + & x_2 & & \\
\text{s.t.} & & x_1 & + & x_2 & \geq & 1 \\
 & & 3x_1 & + & 2x_2 & \leq & 6 \\
 & & \multicolumn{5}{c}{x_1 \geq 0,\; x_2 \geq 0\,.}
\end{array}$$

In Exercise 4.6, we found that $x_1 = 0$, $x_2 = 1$ is an optimal solution to this LP. Find the corresponding Simplex tableau.

Now suppose that the constraint $x_1 - x_2 \geq 1$ is added to the problem. Find an optimal solution of the new LP by applying the dual simplex method.

Exercise 8.3. Consider the following LP.

$$\begin{array}{lccrcrcl} \max & z & = & 240x_1 & + & 60x_2 & & \\ \text{s.t.} & & & 2x_1 & + & 2x_2 & \leq & 100 \\ & & & 6x_1 & + & x_2 & \leq & 100 \\ & & & 10x_1 & & & \leq & 100 \\ & & & & & \multicolumn{3}{r}{x_1 \geq 0,\ x_2 \geq 0\,.} \end{array}$$

In Exercise 4.8, we found that $x_1 = 10$, $x_2 = 40$ is an optimal solution to this LP. Find a corresponding Simplex tableau.

Now suppose that the constraint $-x_1 + x_2 \leq 20$ is added to the problem. Find an optimal solution of the new LP by applying the dual simplex method.

Part III

Polynomial-Time Algorithms

Chapter 9

Maximum Flow

Suppose that the Dutch railways have ordered 50 new trains in Italy and need to move them all to Amsterdam for maintenance. The European rail network is used intensively, so tracks have limited available capacity. How many trains can we move simultaneously from Rome to Amsterdam, choosing routes for each train in such a way that we do not exceed the capacity of any track?

This problem, which is called the Maximum Flow Problem, or Max Flow, has been studied extensively. We have quite efficient ways of solving it, for example using the Ford-Fulkerson algorithm, and we know a way to check whether a solution is optimal.

This way to check for optimality uses duality, like for LP problems. In fact, the dual of Max Flow has a nice interpretation, too. This dual is the Minimum Cut problem (Min Cut), where the goal is to find a set of edges of minimal weight such that removing these edges separates the source and the target.

9.1 A mathematical description of Max Flow

A natural first question when we talk about maximum flow problems is the question what a flow is exactly. In this section we answer this question, and then we give a mathematical description of the max flow problem.

9.1.1 Flows through networks

An easy way to think of a flow through a network is the following. Envision each edge of the network as a pipe through which a substance like water can flow. Of course each pipe has a certain size, or capacity, which limits the flow of water through the pipe. Let's say we want to get water from a source s to a target t in the network. A flow from s to t is then just the amount of water that flows through each pipe.

To make this into a mathematical definition, we need to make sure that we add properties we assume implicitly for a 'real' flow. For example, water does not just disappear, and a pipe cannot sustain more flow than its capacity. Now let us use this to write down a definition of a flow.

Definition 9.1. Let $D = (V, A)$ be a directed graph with *capacity function* $b : A \to \mathbb{R}_{\geq 0}$, and let s and t be two nodes of D called the source and the target. A *flow* from s to t is a function $f : A \to \mathbb{R}_{\geq 0}$ such that there is *conservation of flow* in each vertex except for s and t:

$$\sum_{(u,v)\in A} f_{uv} = \sum_{(v,w)\in A} f_{vw} \quad \forall v \in V \setminus \{s,t\}.$$

A flow is called *feasible* if additionally the capacity is not exceeded:

$$f_{uv} \leq b_{uv} \qquad \forall (u,v) \in A.$$

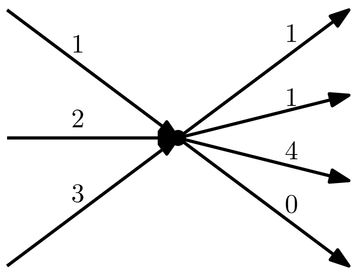


Figure 9.1: An example of conservation of flow at one node. The incoming flow $1 + 2 + 3 = 6$ is equal to the out-going flow $1 + 1 + 4 = 6$.

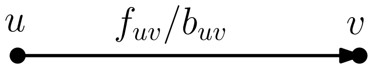


Figure 9.2: In flow diagrams where both the flow and the capacity are shown, we denote the flow f and capacity b as in this figure.

In this definition, and elsewhere, f_{uv} is shorthand for $f((u, v))$ and, similarly, b_{uv} is shorthand for $b((u, v))$. An example of flow conservation is given in Figure 9.1. When we illustrate a flow in a diagram, we will use the notation in Figure 9.2.

Note that Definition 9.1 only defines flows for directed networks. Sometimes it might make more sense to model a problem with an undirected network, such as for our example with the pipes and water: Water may flow in any direction in a pipe. In Exercise 9.5 we will see that this poses no problem, and the directed problem is actually flexible enough to also encode the undirected problem.

9.1.2 Max Flow problem formulation

In this section we define Max Flow formally. We already know we want to find an optimal flow, but what does optimal mean in this situation? For the example of water flowing from a source to a target, that is clear: we want to get as much water as possible from the source to the target. We can translate this to a more mathematical statement as follows.

Definition 9.2. Let f be a feasible flow in a directed graph D with source s, target t and capacity function b. The *value* of the flow f is

$$\sum_{(s,v)\in A} f_{sv} - \sum_{(u,s)\in A} f_{us}.$$

Close inspection of this definition shows that the value of the flow f is equal to the net out-flow at s. This is the correct definition for the amount of flow from s to t because there is conservation of flow by definition. This means that the net out-flow at s is equal to the net in-flow at t, and no amount of flow is lost anywhere else. The mathematically precise version of this statement is

$$\sum_{(s,v)\in A} f_{sv} - \sum_{(u,s)\in A} f_{us} = \sum_{(u,t)\in A} f_{ut} - \sum_{(t,v)\in A} f_{tv},$$

and the proof is an exercise (Exercise 9.1).

Knowing the value we want to optimize, it is easy to formulate the problem formally. We are searching for the feasible flow that has maximum value.

Max Flow
Given: directed graph $D = (V, A)$, vertices $s, t \in V$ and a capacity function $b : A \to \mathbb{R}_{\geq 0}$.

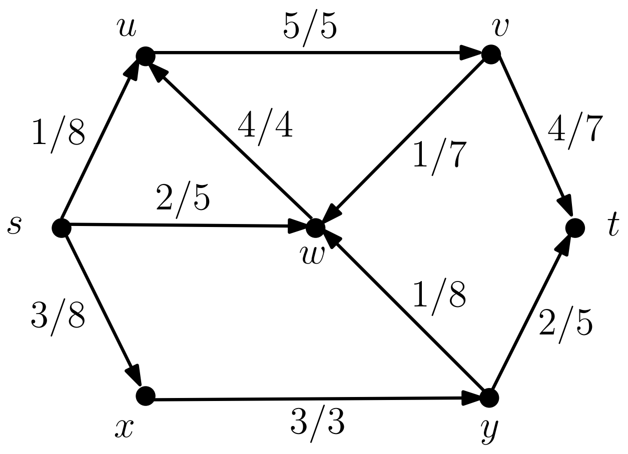


Figure 9.3: A network with flow and capacity.

Find: a feasible flow $f^* : A \to \mathbb{R}_{\geq 0}$ such that the value of f^* is maximized, i.e.:

$$f^* = \underset{f \text{ feasible flow}}{\operatorname{argmax}} \left(\sum_{(s,v)\in A} f_{sv} - \sum_{(u,s)\in A} f_{us} \right)$$

Solving this problem is not straightforward. How would we for example find out whether a feasible solution is optimal, or if it can be improved. Take a look at Figure 9.3, for example. At the moment we have no easy way to see if this flow is optimal. In the next sections we look at two ways of solving MaxFlow. The first one is by rewriting the problem as an LP, the second exploits other properties of the problem. We will also see that duality is useful for proving optimality.

9.2 An LP formulation

Because a lot is known about LPs, it might be very useful to formulate Max Flow as an LP. This often is a good idea even if it is only because there exist rather efficient solvers for LPs. This is not the only benefit. An LP formulation also allows us to study the dual problem, which is very insightful.

9.2.1 Node-arc incidence matrix

It is quite easy to rewrite Max Flow as an ILP using the so-called *node-arc incidence matrix.* This is a matrix encoding the structure of the network. The rows correspond to the nodes and the columns to the edges.

Definition 9.3. Let $D = (V, E)$ be a directed graph with $V = \{1, \ldots, n\}$ and $E = \{e_1, \ldots, e_m\}$, then the *node-arc incidence matrix* of D is the matrix with the following entries:

$$A_{ij} = \begin{cases} 1 & \text{if arc } e_j \text{ leaves } i \\ -1 & \text{if arc } e_j \text{ enters } i \\ 0 & \text{otherwise.} \end{cases}$$

We now formulate Max Flow as an LP problem using this matrix. Suppose we have an instance $(D = (V, E), s, t, b : E \to \mathbb{R}_{\geq 0})$ of Max Flow. Let A be the node-arc incidence matrix of D where the first row corresponds to s and the last row corresponds with t. Then the corresponding LP is the following.

$$\begin{aligned} \max\ & w \\ s.t.\ & A\boldsymbol{f} + \begin{bmatrix} -1 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} w = 0 \\ & \boldsymbol{f} \leq \boldsymbol{b} \\ & \boldsymbol{f} \geq \boldsymbol{0} \\ & w \geq 0 \end{aligned}$$

The decision variables are the f_{uv} (the flow on arc (u, v)) and w (the value of the flow from s to t).

Each restriction of $A\boldsymbol{f} + w(-1, 0, \ldots, 0, 1)^{\mathsf{T}} = \boldsymbol{0}$ corresponds to conservation of flow. The first and the last row do this in a special way: for the first row we ask the net out-flow from s to be equal to the value of the flow, and similarly for the last row and the in-flow at t. All these restrictions together therefore make each point into a flow on D from s to t. The restriction $\boldsymbol{f} \leq \boldsymbol{b}$ ensures that each feasible point of the LP corresponds to a feasible flow.

9.3 Minimum Cut: the dual problem

We know that the dual of an LP in principle contains the same information as the primal problem. So it is not surprising that a dual can provide us with new ways of looking at the same problem. However, not every LP has a dual with a nice interpretation. For Max Flow we are lucky: the dual has a nice interpretation and gives us an insightful way of proving optimality. This dual problem will turn out to be the following optimization problem.

Min Cut
Given: a directed graph $D = (V, A)$, vertices $s, t \in V$ and a capacity function $b : A \to \mathbb{R}_{\geq 0}$.
Find: a cut $W \subset V$ such that:

1. The cut separates s and t, i.e. $s \in W$ and $t \in V - W$.
2. the capacity of the cut is minimal

$$\sum_{(u,v) \in A, u \in W, v \in V - W} b_{uv} \qquad \text{is minimal.}$$

Note that the data for an instance of max flow are the same as for a min cut instance. This is one indication that it may make sense to see these problems as each other's duals.

It is important to realise that a cut is defined as a subset of the vertices, and not of the edges. It is tempting to think of a cut as a subset of the edges, but this makes it a lot harder to check whether the cut actually separates s and t. For subsets of vertices this easy, because we only have to check whether s is in W and t not in W.

For this problem we write down an ILP formulation with decision variables: π_v for each $v \in V$, and γ_{uv} for each $(u, v) \in A$.

$$\begin{aligned} \min\ & \sum_{(u,v) \in A} b_{uv} \gamma_{uv} \\ s.t.\ & \pi_u - \pi_v + \gamma_{uv} \geq 0 && \forall (u, v) \in A \\ & -\pi_s + \pi_t \geq 1 \\ & \pi_v \in \{0, 1\} && \forall v \in V \\ & \gamma_{uv} \in \{0, 1\} && \forall (u, v) \in A \end{aligned}$$

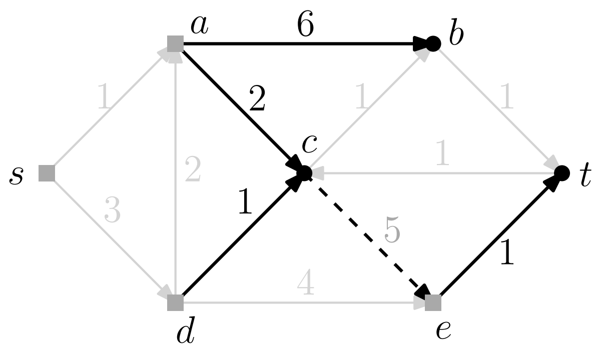


Figure 9.4: A network with a cut $W = \{s, a, d, e\}$ (square nodes) of capacity $6 + 2 + 1 + 1 = 10$. The light arrows are the arrows between the nodes in W and the arrows within $V - W$, the black solid arrows are from W to $V - W$ (these count for the capacity of the cut), the black dotted arrows are in the wrong direction to count for the capacity of the cut.

To check that this is indeed an ILP formulation of Min Cut, note that we can obtain a cut by taking $W := \{u \in V : \pi_u = 0\}$ and $V - W := \{v \in V : \pi_v = 1\}$. This way, the arrows with $\gamma_{uv} = 1$ are the arrows from W to $V - W$.

9.3.1 Min-cut max-flow

In this subsection we focus on the dual relation between Max Flow and Min Cut. To do this, we have to study the dual of the LP formulation of Max Flow instance. Let $(D = (V, A), s, t, b : E \to \mathbb{R}_{\geq 0})$ be an instance of Max Flow. The decision variables of the dual problem are: π_v for each $v \in V$ and γ_{uv} for each $(u, v) \in A$.

$$\begin{array}{lll} \min & \sum\limits_{(u,v)\in A} b_{uv}\gamma_{uv} & \\ s.t. & \pi_u - \pi_v + \gamma_{uv} \geq 0 & \forall (u,v) \in A \\ & -\pi_s + \pi_t \geq 1 & \\ & \pi_v \in \mathbb{R} & \forall v \in V \\ & \gamma_{uv} \geq 0 & \forall (u,v) \in A \end{array}$$

Note that this is very similar to the ILP formulation of Min Cut, with the important difference that Min Cut has integral ($\{0, 1\}$ valued!) decision variables. It turns out that, although one of these problems is an ILP and the other an LP, optimal solutions to these two problems have the same value.

At this point we cannot argue this completely, therefore we just argue that changing the restrictions $\pi_v \in \mathbb{R}$ and $\gamma_{uv} \geq 0$ to $\pi_v, \gamma_{uv} \in [0, 1]$ does not change the optimal value.

Lemma 9.1. Let $I = (D = (V, A), s, t, b : A \to \mathbb{R}_{\geq 0})$ be an instance of Min Cut, then there is an optimal solution to the LP formulation LP_I of I such that $\pi_v, \gamma_{uv} \in [0, 1]$.

Proof. Suppose we have a feasible solution π_v and γ_{uv} and let $c \in \mathbb{R}$ be a constant, then $\pi_v + c$ together with γ_{uv} is also a feasible solution with the same value. Therefore we can assume that $\pi_s = 0$ and $\pi_t \geq 1$.

Now we prove that we can also assume that $\pi_t = 1$. Suppose we have a feasible solution with $\pi_s = 0$ and $\pi_t > 0$, and consider the solution π_u/π_t together with γ_{uv}/π_t. Because $\pi_u - \pi_v + \gamma_{uv} \geq 0$, we also have

$$\frac{\pi_u - \pi_v + \gamma_{uv}}{\pi_t} = \frac{\pi_u}{\pi_t} - \frac{\pi_v}{\pi_t} + \frac{\gamma_{uv}}{\pi_t} \geq 0.$$

Similarly, we can see that $-\frac{\pi_s}{\pi_t} + \frac{\pi_t}{\pi_t} = 0 + 1 \geq 1$, hence the new solution is feasible. Because $\pi_t > 1$, the objective function decreases by a factor $1/\pi_t$. Therefore we can assume that an optimal solution has $\pi_s = 0$ and $\pi_t = 1$.

Lastly we show that for all other π_u we can also assume that their values lie between 0 and 1. Suppose there is a feasible solution with $\pi_s = 0$, $\pi_t = 1$. Then define a new solution π' with

$$\pi'_u = \begin{cases} \pi_u & \text{if } \pi_u \in [0,1] \\ 0 & \text{if } \pi_u < 0 \\ 1 & \text{if } \pi_u > 1 \end{cases},$$

and γ'_{uv} minimal such that the conditions are met. We claim that $\gamma_{uv} \geq \gamma'_{uv}$. This is easily proved if we rewrite the condition for γ_{uv} to $\gamma_{uv} \geq \pi_v - \pi_u$. Suppose for contradiction that $\gamma'_{uv} > \gamma_{uv}$. For that to happen we need $\gamma'_{uv} := \max(0, \pi'_v - \pi'_u) > \max(0, \pi_v - \pi_u) =: \gamma_{uv}$. This can only happen if $\pi'_v - \pi'_u > 0$, or $\pi'_v > \pi'_u$. This implies $\pi_v > \pi_u$, which tells us that we have the following two implications:

$$\pi_u > \pi'_u \implies \pi'_u = \pi'_v = 1 \implies \gamma'_{uv} = 0,$$

and

$$\pi_v < \pi'_v \implies \pi'_u = \pi'_v = 0 \implies \gamma'_{uv} = 0.$$

Indeed, if $\pi_u > \pi'_u$ then $\pi'_u = 1$, and because $\pi_v > \pi_u$, we also have $\pi'_v = 1$. Hence to get $\gamma'_{uv} > \gamma_{uv}$ we must have $\pi_u \leq \pi'_u$ and $\pi_v \geq \pi'_v$, but then $\pi'_v - \pi'_u \leq \pi_v - \pi_u$, so $\gamma'_{uv} \leq \gamma_{uv}$, a contradiction. We conclude that π', γ' is an optimal solution with all variables in $[0,1]$. □

In Chapter 12 we introduce the necessary tools for proving that we can also assume that the optimal solution is integral. This will complete the proof of the following theorem, stating that Max Flow and Min Cut are LP duals. This allows us to use all tools that come with LP duality.

Theorem 9.1 (Max-flow min-cut). Let $(D = (V, A), s, t, b : A \to \mathbb{R}_{\geq 0})$ be an instance of Max Flow and of Min Cut. Then the optimal value x^* of Max Flow for this instance is equal to the optimal value y^* of Min Cut for this instance.

This theorem solves one of the problems we mentioned earlier: how do you know whether a feasible solution of a Max Flow instance is optimal? We find a solution to Min Cut with the same value. Such a dual solution is called a *certificate*, because it certifies that the primal solution is optimal.

Here LP duality can help us even further, because Complementary Slackness gives us information about the optimal solution of the dual problem. For the pairs of instances we consider here, we can loosely say that each arrow that is 'in the cut' (i.e. from W to $V - W$) must be used to its capacity, and each arrow that is from $V - W$ to W cannot carry any flow. This helps us find a certificate.

Theorem 9.2 (Complementary slackness for Max Flow). Let $(D = (V, A), s, t, b : A \to \mathbb{R}_{\geq 0})$ be an instance of Max Flow and of Min Cut, then a flow f and a cut W are optimal if and only if

- $f_{uv} = b_{uv}$ for all $(u, v) \in A$ with $u \in W$ and $v \in V - W$.
- $f_{uv} = 0$ for all $(u, v) \in A$ with $u \in V - W$ and $v \in W$.

9.4 The Ford-Fulkerson algorithm

We now know some tricks to show that a feasible solution for a Max Flow instance is optimal, but we do not have an easy way of producing candidate solutions. Here we take a look at the algorithm of Ford and Fulkerson. This algorithm finds an optimal flow through a network by starting from no flow, and increasing it bit by bit.

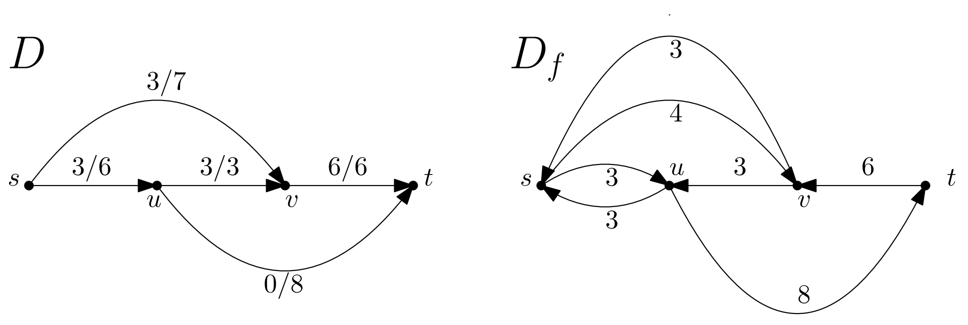


Figure 9.5: Left a network D with flow f and capacity b, right the auxiliary graph D_f.

9.4.1 The auxiliary graph

To increase the flow when we already have a feasible flow, we need to find a path over which we can improve the flow. Such a path is called a *flow-augmenting path*, and it can be found using the auxiliary graph.

Definition 9.4. Let f be a feasible s-t flow in a network $D = (V, A)$ with capacity b. The *auxiliary graph* $D_f = (V_f, A_f)$ is defined as follows. The set of nodes is the same, i.e. $V_f = V$. The arrows are

- For every $(u, v) \in A$ with $f_{uv} < b_{uv}$ an arrow $(u, v) \in A_f$ with capacity $l_{uv} = b_{uv} - f_{uv}$.

- For every $(u, v) \in A$ with $f_{uv} > 0$ an arrow $(v, u) \in A_f$ with capacity $l_{vu} = f_{uv}$.

This auxiliary graph is built exactly so that if there is a path from s to t in D_f, then we can augment the flow over this path. This is because each of the forward arrows is only there if there is some capacity left to use. Similarly the reverse arrows are there exactly if there is already some flow on that arc, hence we can decrease the flow there (which can also be seen as letting some flow go back on that arc).

9.4.2 The algorithm in pseudo code

Using the auxiliary graph we defined in the previous subsection, the Ford-Fulkerson algorithm can be written down quite compactly.

```
  Data: An instance of MAX FLOW $(D = (V, A), s, t, b : A \to \mathbb{R}_{\geq 0})$.
  Result: a maximum $s$-$t$ flow $f$
1 Set $f_{uv} = 0$ for all $(u, v) \in A$;
2 Construct auxiliary graph $D_f$ with capacity $l$;
3 while there is a directed $s$-$t$ path in auxiliary graph $D_f$ do
4     Let $P$ be a directed $s$-$t$ path in $D_f$;
5     Let $a$ be the maximum amount of flow $P$ can carry, that is $a := \min\{l_{uv} : (u, v) \in P\}$;
6     Augment the flow $f$ as follows:
          $f_{uv} := \begin{cases} f_{uv} + a & \text{if } (u,v) \in P \\ f_{uv} - a & \text{if } (v,u) \in P \\ f_{uv} & \text{otherwise} \end{cases}$ ;
       producing a new flow $f$ with bigger value;
7     Construct new auxiliary graph $D_f$ with capacity $l$;
8 end
9 return $f$
```

Algorithm 1: The Ford-Fulkerson for MAX FLOW

With just the pseudo code, it might not directly be clear why the output of the algorithm is optimal. We can prove this using the tools from previous sections about duality: we can give a certificate for the optimality of the output.

Theorem 9.3 (Correctness of Ford-Fulkerson)**.** The output f of Ford-Fulkerson is an optimal flow.

Proof. As the algorithm output f, there may not be an s-t path in D_f, otherwise the algorithm would have used this path to augment f. Hence we can define a cut as follows. Let W be the set of nodes that can be reached from s in D_f. Because there is no s-t path in D_f, this is an s-t cut.

Now we determine the value of the cut W. Let $(u, v) \in A$ be an arrow from W to $V - W$. By definition (u, v) is not in D_f, as otherwise v should be a node in W. This means that *not* $f_{uv} < b_{uv}$, or equivalently $f_{uv} = b_{uv}$. Now let $(x, y) \in A$ be an arrow from $V - W$ to W, then (y, x) is not an arrow of D_f as otherwise x would have been in W. This means that *not* $f_{xy} > 0$, as this would create the edge (y, x) in D_f with capacity f_{xy}. Hence $f_{xy} = 0$. By complementary slackness (Theorem 9.2) the cut is optimal and has same value as f. This means that f is also optimal. □

9.4.3 Running time

Another valid question about such an algorithm is how quickly it finds a solution, and also whether it finds a solution in finite time at all. The second question can easily be answered for integral instances: the flow strictly increases in each step of the algorithm by at least 1, so as long as there is a (bounded) optimal solution, the algorithm can only run for a finite number of iterations. The question about the running time is slightly more nuanced, and asks for a more detailed analysis. In Chapter 15 we look at ways of analysing running times of algorithms. There we will also analyse the running time of MAX FLOW.

9.4.4 Example

As an example, we will use Ford Fulkerson on the network of Figure 9.6. The first step in the algorithm is to set the flow on each arc to zero, this is our initial flow. Then we have to compute the auxiliary graph for this flow. Because there is no flow, this auxiliary graph is equal to the network for Figure 9.6.

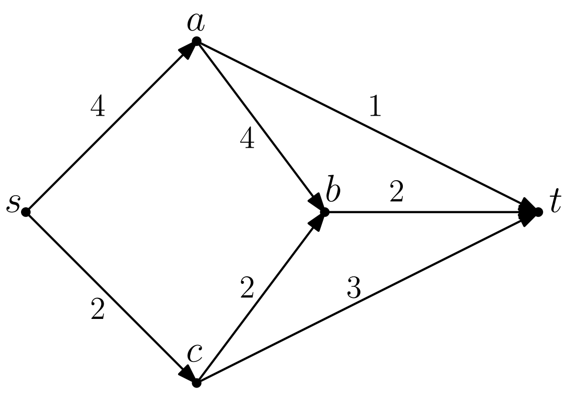


Figure 9.6: The network of which we compute the maximal flow using Ford-Fulkerson.

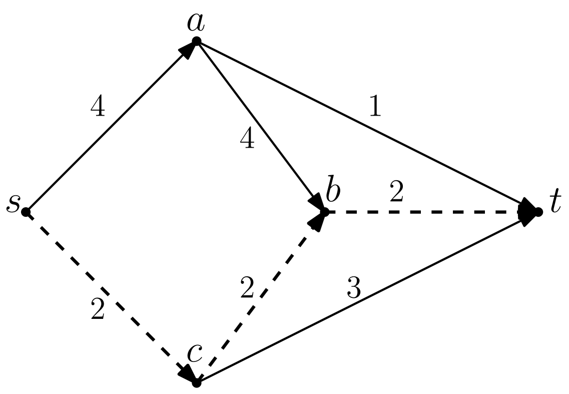


Figure 9.7: The first auxiliary graph, the augmenting path is shown with dashed arrows.

The auxiliary graph is shown in Figure 9.7. The next step of the algorithm asks for us to find an augmenting path. This is just a path from s to t in the auxiliary graph. An example of such a path is shown with the dashed arrows in Figure 9.7. The minimal capacity of the arcs on this path is 2, so we augment the flow by 2 on this path. This gives a new flow, which is shown in Figure 9.8

Now we are at the end of the contents of the while loop, so we need to decide whether we stay in the loop. For this, we need to know if there is an s-t path in the new auxiliary graph. Hence we again build the auxiliary graph, but now for the new flow. This graph is shown in Figure 9.9, where the black arrows indicate the remaining capacity $(c_e - f_e)$, and the grey arrows in reverse direction show the flow we can cancel. Because there is an s-t path in this graph (dashed arrows), we need to continue in the loop.

The next step is to use the path in the auxiliary graph to augment the flow again. Note that the minimal capacity of the arrows in the path is 2, so we augment on this path by 2. For each black arrow in the path we increment the flow on the arrow in the network by two, and if the path uses a grey arrow, we decrease the flow on the arc in the network by 2. This gives us a better flow, which is shown in Figure 9.10.

Again we have to decide whether we are done, or we have to continue the loop. For this we again produce an auxiliary graph (Figure 9.11). Again, there is an s-t path. This path has capacity 1, so we augment the flow along this path by 1. This gives the flow shown in Figure 9.12.

Now, as it will turn out, we produce the auxiliary graph for the last time (Figure 9.13). There is no s-t path in this graph (t can only be reached from c, and c can only be reached from t). This means that we do not have to continue the loop, and the flow of value 5 that we found is optimal.

Of course we should check that we did not make any mistakes, so we will try to find a certificate

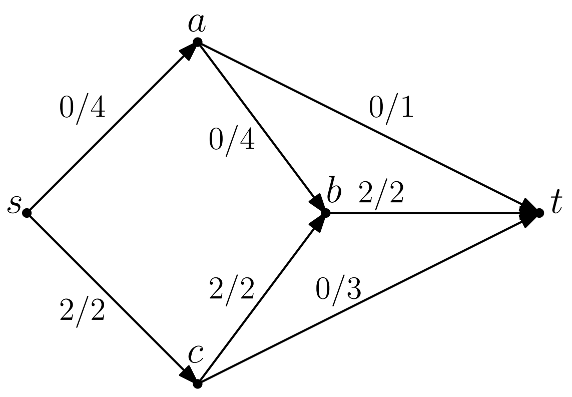


Figure 9.8: The network with capacity and flow after the first iteration of Ford Fulkerson.

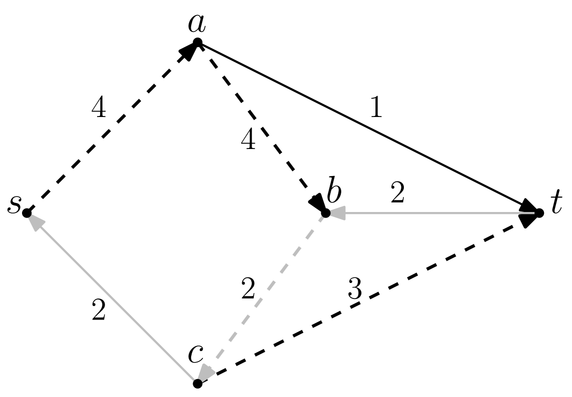


Figure 9.9: The second auxiliary graph. Forward arrows are black, and back arrows are grey. The augmenting path is shown with dashed arrows.

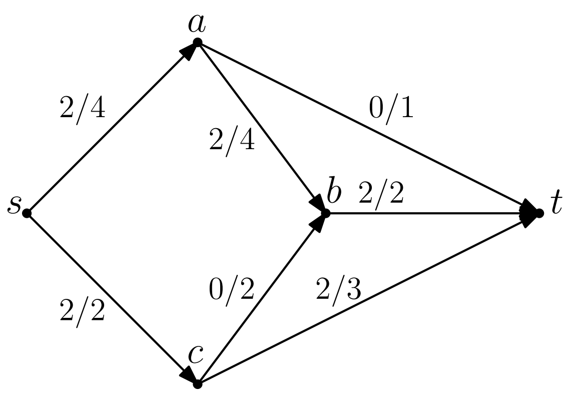


Figure 9.10: The network with capacity and flow after the second iteration of Ford Fulkerson.

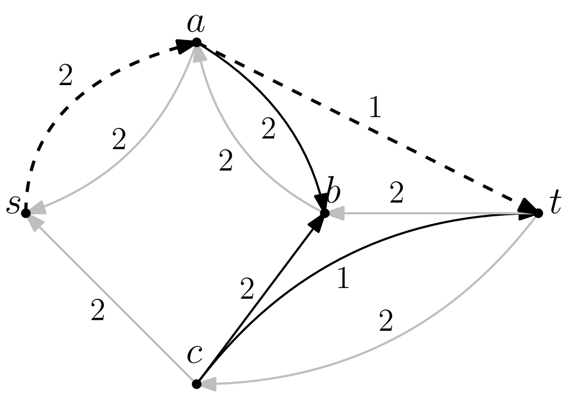


Figure 9.11: The third auxiliary graph. Forward arrows are black, and back arrows are grey. The augmenting path is shown with dashed arrows.

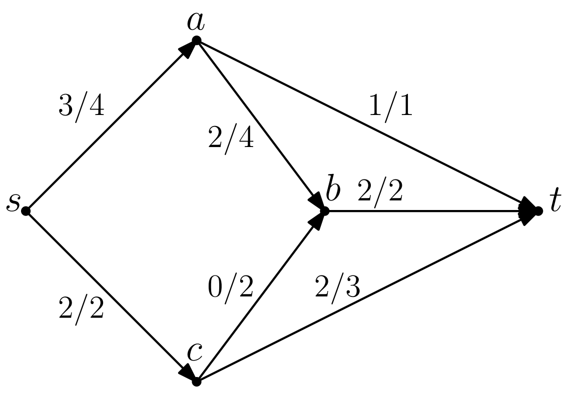


Figure 9.12: The network with capacity and flow after the third iteration of Ford Fulkerson.

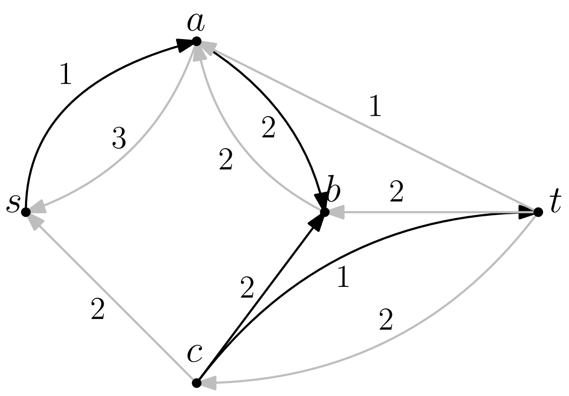


Figure 9.13: The fourth auxiliary graph. Forward arrows are black, and back arrows are grey. The augmenting path is shown with dashed arrows. Note that there is no path from s to t in this graph, so the algorithm terminates.

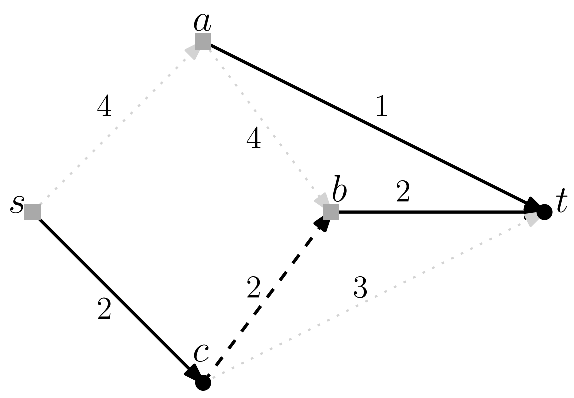


Figure 9.14: The minimal cut $W = \{s, a, b\}$ (square nodes) with $V \setminus W = \{c, t\}$ (circle nodes). The black arrows are arrows between W and $V \setminus W$. Note that the dashed black arrow is not counted for the capacity of the cut, because it goes from $V \setminus W$ to W. The capacity of the cut is 5: it is the capacity of the thick black arrows going from W to $V \setminus W$.

for optimality. We do this by finding a cut of value 5, the same value as our flow. Because there is a strong connection between maximal flows and minimal cuts, we can try to use our optimal flow to find a cut. This can be done using complementary slackness, Theorem 9.2. This theorem tells us that for any cut $W \subset V$, any arc (u, v) that is not fully used by the flow (i.e. $f_{uv} < b_{uv}$), can not have $u \in W$ and $v \in V \setminus W$. Similarly any arc (u, v) that has positive flow can not have $u \in V \setminus W$ and $v \in W$.

Hence, we may start with $W = \{s\}$ and add a vertex v to W when there is either an arc (u, v) with $u \in W$, $v \notin W$ and $f_{uv} < b_{uv}$ or an arc (v, u) with $u \in W$, $v \notin W$ and $f_{uv} > 0$. We continue like this until no vertices can be added to W in this way any more, implying that W and f satisfy the complementary slackness conditions. Note that this procedure corresponds exactly to adding all vertices v to W for which there exists a directed path from s to v in D_f.

We do this for the flow found by the algorithm. We start with $W = \{s\}$, then add a to W because $f_{sa} = 3 < 4 = b_{sa}$. Then we add b to W because $f_{ab} = 2 < 4 = b_{ab}$. Then we have obtained a cut $W = \{s, a, b\}$ satisfying the complementary slackness conditions, see Figure 9.14. The cut has capacity

$$\sum_{(u,v), u \in W, v \notin W} b_{uv} = b_{at} + b_{bt} + b_{sc} = 1 + 2 + 2 = 5.$$

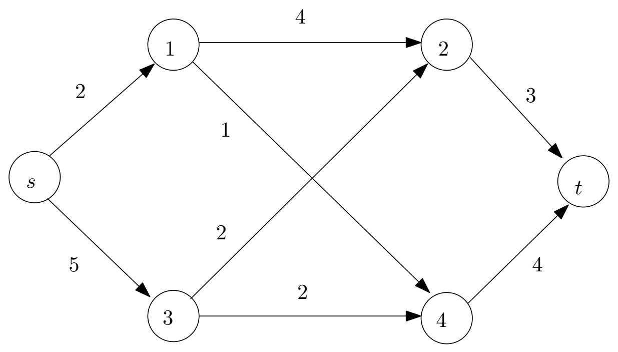
Since the capacity of the cut is equal to the value of the flow, we know that both are optimal.

9.5 Exercises

Exercise 9.1. Let f be a flow from s to t, prove that

$$\sum_{(s,v)\in A} f_{sv} - \sum_{(u,s)\in A} f_{us} = \sum_{(u,t)\in A} f_{ut} - \sum_{(t,v)\in A} f_{tv}.$$

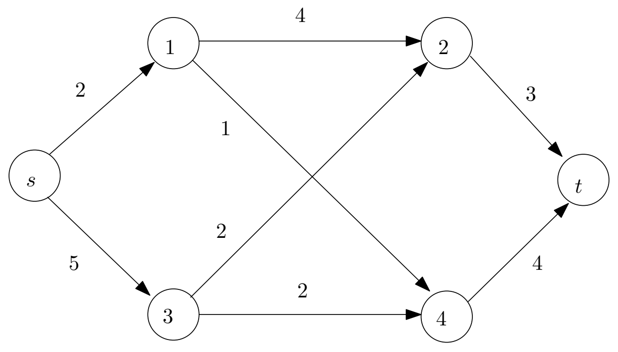
Exercise 9.2. **a)** Write down the node-arc incidence matrix of the following graph.



b) Draw the graph corresponding to the following node-arc incidence matrix.

$$\begin{array}{ccccccc}
-1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & -1 & 0 & 0 & 0 & -1 & 1 \\
0 & 0 & 0 & -1 & 0 & 1 & 0 \\
0 & 0 & -1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & -1
\end{array}$$

Exercise 9.3. **a)** Use an s-t cut to determine an upper bound for the flow from s to t in the network below. The labels at the arrows indicate the capacity.



b) Determine a maximal flow from s to t in the network using the Ford-Fulkerson algorithm. Write down the details of each step of the algorithm. A complete answer also includes the output of the algorithm: the value of the maximal flow and the realised flow on each arrow.

Exercise 9.4. Consider a modified Max Flow problem in which each node also has a capacity. Show that each instance of this problem can be reformulated to a normal Max Flow instance. Note that this allows us to solve this modified Max Flow problem using Ford-Fulkerson.

Exercise 9.5. Consider a modified Max Flow problem in which the underlying network is undirected, so flow in each edge can go either way. Show that each instance of this problem can be reformulated to a normal Max Flow instance. Note that this allows us to solve this modified Max Flow problem using Ford-Fulkerson.

Exercise 9.6. A group of students consisting of 62 men and 34 women want to form pairs for making homework assignments. Each student has a list of possible partners with whom they want to work together. It so happens that male students only have female students on their lists and vice versa. All lists are handed over to one student, who has the task to make a maximal number of pairs that want to work together, that is

- The number of pairs is maximized.
- Each member of a pair has the other member on his or her list.

Show that this problem can be rewritten to Max Flow, which makes it possible to solve this problem using Ford-Fulkerson.

Chapter 10

Shortest Paths

An optimization problem that one encounters regularly in daily life is to find a shortest route from A to B. Nowadays, most people usually find those routes with their smartphone. But how does a smartphone find a shortest route? In this chapter we will describe an efficient algorithm for this shortest path problem, which also has numerous less obvious applications. As in the previous chapter, the algorithm is based on linear programming duality.

10.1 Mathematical description

We want to define the Shortest Path problem rigorously, as this makes it possible to actually analyse it using mathematical tools. Although it seems quite straightforward – we just want to find the shortest path – we should be careful: what exactly is a path and what is its length? In this section we answer these questions, and we give an example that shows why we must be careful.

Let us start with the definitions of an s-t path and an s-t walk.

Definition 10.1. An *s-t walk* in a directed graph $D = (V, A)$ is an ordered list of vertices $(v_0, v_1, v_2, \dots, v_k)$ such that $v_0 = s$, $v_k = t$ and $(v_{i-1}, v_i) \in A$, for all $1 \leq i \leq k$. An *s-t path* in D is a directed walk in D that does not use any vertex more than once.

The problem we want to solve asks for a shortest path. To fully define the problem, we still need a description of this objective value: the length of a path. Because each path is a walk, we will give a more general definition: the length of a walk.

Definition 10.2. Let $D = (V, A)$ be a directed graph and $c : A \to \mathbb{R}_{\geq 0}$ a length function on the arcs of D. Define the length $c(W)$ of a walk $W = (v_0, v_1, v_2, \dots, v_k)$ as

$$c(W) := \sum_{i=1}^{k} c_{v_{i-1}v_i},$$

where, as usual, c_{uv} is shorthand for $c((u, v))$.

Shortest Path
Given: a directed graph $D = (V, A)$, vertices $s, t \in V$ and a length function $c : A \to \mathbb{R}_{\geq 0}$.
Find: an s-t path P^* that minimizes $c(P^*)$.

Note in particular that we search for a shortest s-t path and not for a shortest s-t walk. In addition, we require the arc-lengths to be nonnegative. It is natural to wonder how the problem changes when we replace path by walk. If we keep the arc-lengths nonnegative, then this does not change the problem

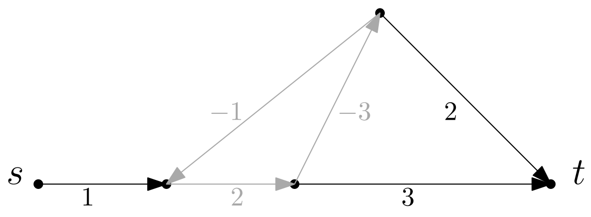


Figure 10.1: A directed graph with a negative length cycle (grey), in which a shortest s-t walk does not exist, while a shortest s-t path does exist.

at all. However, if we allow negative arc-lengths, the problem does change significantly, which is illustrated by the following example.

Suppose we ask for the shortest s-t path in the directed graph of Figure 10.1. Because there is only a finite number of s-t paths, there must be a shortest one. The number of walks, however, is unbounded in any graph with a cycle. This means the same argument cannot show that there is a shortest s-t walk. Even worse, this graph has a cycle with negative length, so we can keep walking along this cycle, decreasing the length of the walk every time we complete the walk over the cycle. This means even though this graph has a shortest s-t path, it does not have a shortest s-t walk. In Exercise 10.2 we analyse the relation between Shortest Path and Shortest Walk in more detail.

10.2 ILP formulation

In the last chapter we saw that Max Flow had a nice LP dual. In this section we try the same trick on Shortest Path. Like for Max Flow, we use the node-arc incidence matrix for a nice formulation.

Let (D, s, t, c) be an instance of Shortest Path, with $A(D)$ the arc-set of D and $n = |A(D)|$. Let A be the node-arc incidence matrix of D where the first row corresponds to s and the last row corresponds to t. Then the corresponding ILP is the following.

$$\begin{aligned} \min \sum_{(u,v)\in A(D)} & c_{uv} f_{uv} \\ s.t. \quad A\boldsymbol{f} &= \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ -1 \end{bmatrix} \\ \boldsymbol{f} &\in \{0,1\}^n \end{aligned}$$

The decision variables are the $f_{uv} = f_{(u,v)}$, which take value 1 if the arc (u, v) is in the shortest path, and 0 otherwise. All these decision variable together form a vector $\boldsymbol{f}$.

10.2.1 LP formulation

Like in previous chapter, we want to study the dual problem of the ILP we just formulated. To go to the dual, we should first try to get an LP formulation. The easiest way to get one, is to relax the integrality condition on f_e. In fact, we will go one step further, and replace $f_{uv} \in \{0, 1\}$ with $f_{uv} \geq 0$.

There are a couple of things we should check now, because it is not directly clear that this encodes the same problem. We essentially just want to be sure that we can not get a “better” solution after relaxing the condition on f_{uv}.

A quick inspection shows that relaxing the $f_{uv} \in \{0,1\}$ to $f_{uv} \in \mathbb{Z}_{\geq 0}$ is no problem: because all lengths are non-negative, no arc will get value $f_{uv} > 1$. Relaxing the integrality also poses no problem, but the proof is slightly more involved.

Lemma 10.1. Let (D, s, t, c) be an instance of Shortest Path. Then the LP relaxation where $f_{uv} \in \{0,1\}$ is replaced with $f_{uv} \geq 0$ has an optimal solution with $f_{uv} \in \{0,1\}$.

Proof. Let $\mathcal{P}$ be the set of all s-t paths in D, each given as an ordered list of arcs. Any feasible solution can be written as a linear combination

$$\sum_{P \in \mathcal{P}} \lambda_P P$$

of the paths with total coefficient $\sum_P \lambda_P$ greater or equal to 1. The total weight of a solution is $\sum_{P \in \mathcal{P}} \lambda_P c(P)$. It does not make sense to pick a solution with total coefficient greater than 1, as for any such solution, we can scale back the total coefficient to 1 and have lower total cost. Now note that any feasible solution with total coefficient 1 is a convex combination of paths, and the total cost is the convex combination of their costs. Furthermore, choosing one path with coefficient 1 is also feasible. So we observe that the optimal solution is a convex combination of shortest paths, and we can simply choose one of these shortest paths. □

Note that this proof only works if the lengths of the arcs are non-negative. To see why, write down this LP relaxation for the shortest path problem for the directed graph in Figure 10.1.

10.2.2 The dual problem

Now we can study the dual of our LP formulation of Shortest Path.

Primal:

$$\begin{aligned} \min \quad & \sum_{(u,v) \in A(D)} c_{uv} f_{uv} \\ s.t. \quad & A\boldsymbol{f} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ -1 \end{bmatrix} \\ & \boldsymbol{f} \geq \boldsymbol{0} \end{aligned}$$

Dual:

$$\begin{aligned} \max \quad & \pi_s - \pi_t \\ s.t. \quad & \boldsymbol{\pi}^\mathsf{T} A \leq \boldsymbol{c} \\ & \boldsymbol{\pi} \in \mathbb{R}^m \end{aligned}$$

Or equivalently:

$$\begin{aligned} \max \quad & \pi_s - \pi_t \\ s.t. \quad & \pi_u - \pi_v \leq c_{uv} \quad \forall (u,v) \in A(D) \\ & \boldsymbol{\pi} \in \mathbb{R}^m \end{aligned}$$

The decision variables in the dual are the π_u for all $u \in V$. The restrictions in the dual make sure that each $\pi_u - \pi_s$ is at most the length of the shortest path from s to u. This means that any feasible dual solution essentially consists of an upper bound for the shortest path from s to u for each $u \in V$.

The duality between these two problems also gives us complementary slackness to play with. Like we saw in last chapter, this is a nice tool for proving optimality. For Max Flow we used it to prove correctness of the Ford-Fulkerson algorithm. In this chapter, we use complementary slackness to prove correctness of Dijkstra's algorithm for Shortest Path.

Theorem 10.1 (Complementary slackness for Shortest Path)**.** Let $(D = (V, A), s, t, c)$ be an instance of Shortest Path and let $\boldsymbol{f}$ and $\boldsymbol{\pi}$ be feasible solutions to, respectively, the primal and dual LP formulations above, then both are optimal if and only if:

$$\begin{aligned} f_{uv} > 0 \quad & \Rightarrow \quad \pi_u - \pi_v = c_{uv} \\ \text{Or equivalently: } \pi_u - \pi_v < c_{uv} \quad & \Rightarrow \quad f_{uv} = 0. \end{aligned}$$

10.3 Dijkstra's algorithm

In this section we study *Dijkstra's algorithm*. This is a polynomial-time algorithm for Shortest Path. The algorithm actually works with the dual problem, and the final state is a feasible dual solution.

The idea of this algorithm is to keep values $\rho(u)$ and a subset of vertices W which we increase from $\{s\}$ to V. In each step of the algorithm we keep $\rho(u)$ the length of the shortest path from s to u using only nodes in W (see Exercise 10.3).

```
  Data: An instance of Shortest Path ($(D = (V, A), s, t, c : A \to \mathbb{R}_{\geq 0})$).
  Result: a shortest $s$-$t$ path
1 Set $W = \{s\}$;
2 Set $\rho(s) = 0$ and $\rho(u) = c_{su}$ for all $u \in V \setminus \{s\}$,
3 where $c_{su} := \infty$ if $(s, u) \notin A$;
4 while $V \setminus W \neq \emptyset$ do
5     Find $u \in V \setminus W$ with $\rho(u)$ minimal;
6     Set $W := W \cup \{u\}$;
7     For each $v \in V \setminus W$ with $(u, v) \in A$, set $\rho(v) := \min\{\rho(v), \rho(u) + c_{uv}\}$;
8 end
9 return $\rho(t)$
```

Algorithm 2: Dijkstra's algorithm for Shortest Path

This algorithm technically uses the dual formulation of Shortest Path to find (the length of) a shortest path. In each step the algorithm gives better upper bounds on the shortest paths from s to any $u \in V$. In the previous section we saw that this corresponds to feasible dual solutions. The actual correspondence between this algorithm and the dual problem is $\boldsymbol{\pi} := -\boldsymbol{\rho}$.

10.3.1 Correctness

We have seen that the algorithm produces a feasible dual solution. However, we do not yet know whether there exists a corresponding feasible solution for the primal problem, i.e. if there is an s-t path with length $\rho(t)$. Here we give two ways to find such a path using the values $\rho(v)$ from the algorithm. It then follows from duality theory that this path is a shortest s-t path.

The first method uses complementary slackness (Theorem 10.1). This theorem tells us that an arc (u, v) can only be on the shortest path if $\pi_u - \pi_v = c_{uv}$, i.e. if $\rho(v) - \rho(u) = c_{uv}$. This gives us a set of arcs, which contains at least one s-t path(because throughout the algorithm each vertex v has at least one incoming arc (u, v) with $\pi_u - \pi_v = c_{uv}$).

The second method is quite similar, but works by *backtracking* from t to s. Starting from $v := t$, choose an arc with (u, v) such that $\rho(v) - \rho(u) = c_{uv}$, set $v := u$ and continue until you end up at s. This gives you an s-t path.

Both these methods are guaranteed to produce a path P from s to t which has length

$$\rho(t) = \rho(t) - \rho(s) = \sum_{(u,v)\in P} \rho(v) - \rho(u) = \sum_{(u,v)\in P} c_{uv} = c(P)$$

and is therefore a shortest s-t path by duality theory.

10.3.2 Running time

In the previous section we saw that the output of Dijkstra's algorithm is optimal. However, we have not shown that the algorithm ever stops and gives us the answer. Therefore we have to consider the

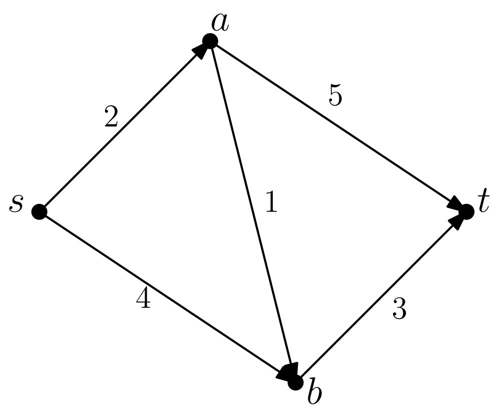


Figure 10.2: An instance of Shortest Path.

running time of the algorithm. We already promised Dijkstra's algorithm would be a polynomial-time algorithm. Here we give a rough idea of why this is the case.

The initialization of the algorithm (where we set initial values of variables) is definitely very fast: we only have to give each node a value, so this takes at most $|V|$ steps. Now we enter the While loop. To know how many steps we need to do in the loop in total, we need to know how much time each iteration of the loop costs, and how often we repeat everything inside the loop.

Let us first look at what happens within each iteration of the loop. First we find a node u in $V \setminus W$ with minimal $\rho(u)$. There are at most $|V|$ elements in $V \setminus W$, so we need to compare at most $|V|$ values. This takes about $|V|$ steps. Now we update W, and then we update the values of ρ. As there are $|V|$ values of ρ, this updating costs about $c|V|$ time ($c > 0$ a constant). The while loop ends here and we conclude that one iteration of the loop costs $C|V|$ time with $C > 0$ some constant.

Now consider how many times we have to repeat this. Note that we stay in the loop until $W = V$. Because W is always a subset of V, we could also say that we stay in the loop until $|W| = |V|$. Now note that in each iteration, $|W|$ increases by one. Hence we have at most $|V|$ iterations of the loop.

Putting everything together, the loop takes at most $C|V|^2$ time. This is because we have to do $C|V|$ steps at most $|V|$ times. Including the initialization, the algorithm runs in $C'|V|^2$ time, with C' a constant, which is usually denoted as $O(|V|^2)$, see Chapter 15.

10.3.3 Example

As an example, we give a detailed application of Dijkstra's algorithm on the network shown in Figure 10.2.

We initialize the algorithm by setting $W = \{s\}$ and labelling the vertices $\rho_s = 0$, $\rho_a = 2$, $\rho_b = 4$, and $\rho_t = \infty$ (Figure 10.3).

Now we enter the loop, and find a vertex $v \in V \setminus W$ with minimal ρ_v. There is only one such vertex, vertex a. We set $W = \{s, a\}$ and update the values of ρ to $\rho_b = 3$ and $\rho_t = 7$ (Figure 10.4).

Now $V \setminus W$ consists of b and t, so we continue the loop. Of these vertices, b has the lower label. Hence we set $W = \{s, a, b\}$ and update the values of ρ by setting $\rho_t = 6$ (Figure 10.5).

Now, for the last time we go through the loop with active node t. We set $W = V$ and update ρ (nothing happens). Now the loop ends because $W = V$, so the algorithm returns $\rho_t = 6$ as the length of the shortest s-t path.

We should now check that our solution is actually optimal, and we made no mistakes. Because the solution of the algorithm is actually a dual solution, this check consists in finding a primal solution with the same value, i.e. an s-t path of length 6. We have two ways of doing this: backtracking, and complementary slackness.

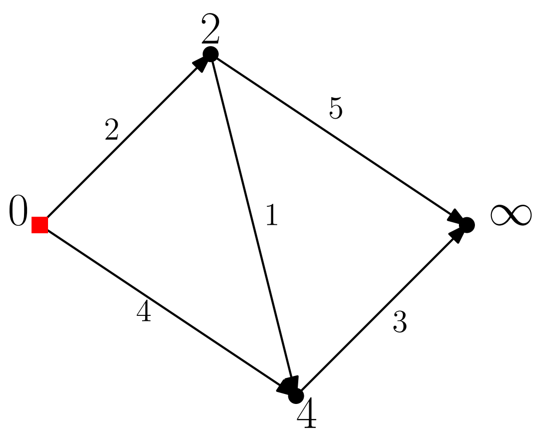


Figure 10.3: The instance of SHORTEST PATH from Figure 10.2 with the labels ρ_v after the initialization of Dijkstra's algorithm.

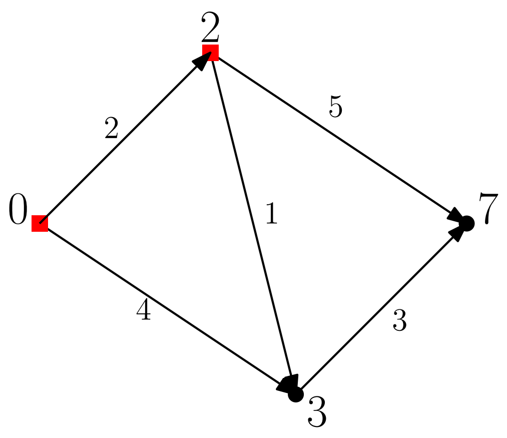


Figure 10.4: After one iteration of Dijkstra's algorithm.

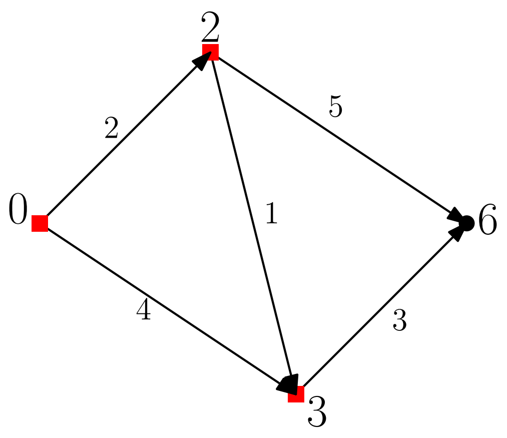


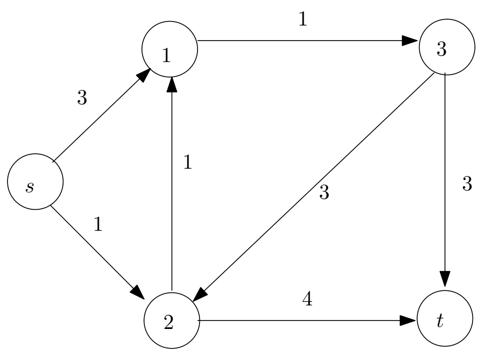
Figure 10.5: After two iterations of Dijkstra's algorithm.

If we use backtracking, we start at node t and find an incoming arc (u,t) where the potential difference $\rho(t)-\rho(u)$ is equal to the length c_{ut} of the arc. The only incoming arc with this property is (b,t). This means the shortest path uses (b,t) and we can backtrack further from b. We again find an incoming arc where the potential difference is equal to the length of the arc. Here again there is only one such arc: (a,b). Hence the shortest path uses (a,b) and we continue backtracking in node a. There is only one incoming arc in a, the arc (s,a), hence the shortest path uses this arc. We have reached s, so we have found the shortest path. We check that it indeed has length $c_{sa}+c_{ab}+c_{bt}=2+1+3=6$.

Using complementary slackness is similar to backtracking, with one difference. Instead of finding the arcs in the shortest path one by one from t to s, we identify all arcs where the potential difference is equal to the length at the same time. Any s-t path using only these arcs must be a shortest path. In this case the set of all these arcs is exactly $\{(s,a),(a,b),(b,t)\}$. There is only one s-t path using these arcs, and it must then be minimal, we check the length, which is still 6, which proves that our potential was optimal, and therefore that this is a shortest path.

10.4 Exercises

Exercise 10.1. Find a shortest path from s to t in the following digraph using Dijkstra's algorithm. The lengths of the arcs are given in the figure as labels next to the arcs. Write down each step, and give the shortest path as well as its length.



Exercise 10.2. Consider the LP formulation of SHORTEST PATH as defined in Section 10.2. Assume the lengths of the arcs can be negative as well. Suppose there is exist directed paths from s to v and from v to t for every point $v \in V$. Prove equivalence of the following statements:

(i) There exists a shortest s-t walk.

(ii) The dual LP problem has a feasible solution.

(iii) There is no directed cycle with negative total length.

NB: A *walk* can use nodes and arcs multiple times, a *path* uses each node and arc at most once.

Exercise 10.3. Prove that in each iteration of Dijkstra's algorithm and each $v \in V$, we have the following: $\rho(v)$ is the length of a shortest s-v path using only nodes from W (except for v).

Exercise 10.4. Let I be an instance of SHORTEST PATH with integer lengths. Prove that Dijkstra's algorithm can be changed in the following way, so that it still finds a shortest s-t path:

- If there are multiple options for choosing u in step 5, which includes t, choose t last.
- Replace the condition for the while loop by "while $t \notin W$".

Chapter 11

Minimum Spanning Trees

Suppose you want to connect a number of houses with cables and you want to minimize the total length of the cables used. Not each connection might be possible. Hence, we construct a graph with a vertex for each house and an edge between two houses if it is possible to connect them with a direct cable. The problem that we then need to solve in this graph is called the Minimum Spanning Tree problem. It is called this way because, assuming the lengths of the connections are positive, any optimal set of connections must form a tree (i.e. it is connected and has no cycles) and it is spanning in the sense that it connects all vertices.

In this chapter, we study three different algorithms for the Minimum Spanning Tree problem based on the same mathematical fact, a theorem about building spanning trees incrementally.

11.1 Spanning trees

Definition 11.1. A *spanning tree* of a graph G is a subgraph $T = (V, E')$ of G such that T is a tree.

Note that the vertex set of G is the same as the vertex set of T. Hence, an equivalent way of defining a spanning tree is as a subgraph without cycles that connects all the vertices.

A graph only has a spanning tree if it is connected (Exercise 11.1). Note that graphs often have more than one spanning tree. In fact, each connected graph that is not a tree has at least two spanning trees. An example can be found in Figure 11.1, where we see two spanning trees of the Petersen graph.

Note that by adding one edge to a spanning tree, we get a subgraph with exactly one cycle. If we remove one edge from this cycle, we again obtain a spanning tree (Figure 11.2). We now prove this more formally.

Lemma 11.1. Let $G = (V, E)$ be a connected graph, (V, F) a spanning tree of G, and $e \in E \setminus F$ an edge not in the spanning tree, then the following are true:

1. $(V, F \cup \{e\})$ has a unique cycle C;

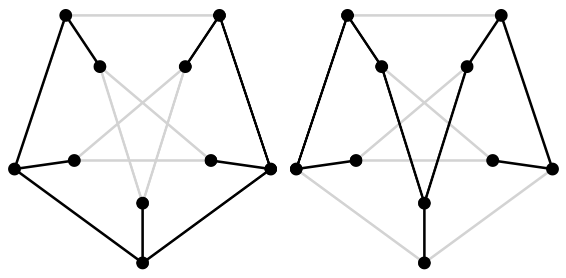


Figure 11.1: Two spanning trees of the Petersen graph.

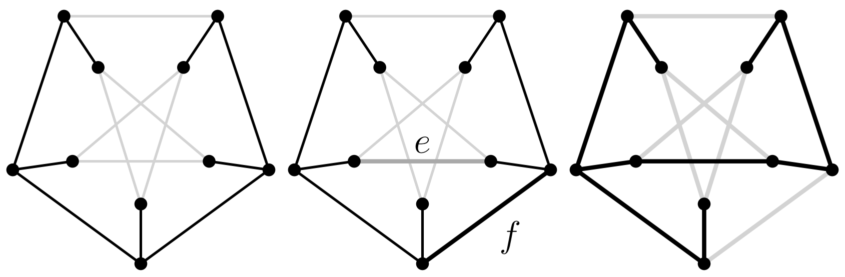


Figure 11.2: An example of how to get a new spanning tree using Lemma 11.1. Adding edge e and removing edge f gives a new spanning tree.

2. for each edge $f \in C$ of the cycle, the subgraph $(V, F \cup \{e\} \setminus \{f\})$ is a spanning tree of G.

Proof. Let us first look at the first statement. Note that (V, F) has no cycles, as it is a tree. Hence if $(V, F \cup \{e\})$ contains a cycle, this cycle must contain the new edge $e = (u, v)$. Indeed, if $(V, F \cup \{e\})$ contains a cycle C which does not involve e, then C is also a cycle of (V, F). Now note that there is a unique path P in (V, F) from u to v, so in $(V, F \cup \{e\})$ there is a unique cycle $P \cup \{e\}$ containing e. Furthermore, because each cycle in $(V, F \cup \{e\})$ must contain e, $P \cup \{e\}$ is the unique cycle of $(V, F \cup \{e\})$.

As for the second part, note that $(V, F \cup \{e\})$ is connected, and removing an edge from a cycle cannot disconnect the graph. Hence $(V, F \cup \{e\} \setminus \{f\})$ is connected and has $|F \cup \{e\} \setminus \{f\}| = |F| = |V| - 1$ edges. By Lemma A.1, $(V, F \cup \{e\} \setminus \{f\})$ is a tree with vertex set V, hence a spanning tree of G. □

11.1.1 The length of a spanning tree

It does not make sense to talk about minimum spanning trees for unweighted graphs (why not?). Hence we have to consider minimum spanning trees for graphs with edges that have a given length.

Definition 11.2. Let $G = (V, E)$ be a graph with lengths $l(e)$ for all $e \in E$. The *total length* of a subgraph $H = (V', E')$ of G is

$$l(H) := \sum_{e \in E'} l(e).$$

Now we have also defined the total length of a spanning tree of a weighted graph, as spanning trees are by definition subgraphs. This means we can now define the minimum spanning tree problem.

Minimum Spanning Tree (MST)
Given: a graph $G = (V, E)$ with length function $l : E \to \mathbb{R}$.
Find: a spanning tree of G with minimum total length.

11.2 Prim-Dijkstra algorithm for MST

In this section we take a look at the *Prim-Dijkstra* algorithm for the Minimum Spanning Tree problem. Note that, although the algorithm is usually named after Prim and/or Dijkstra, it was actually first discovered by Vojtěch Jarník already in 1930. This algorithm basically builds a minimum spanning tree by starting with a single vertex and repeatedly adding the shortest edge to a vertex that was not connected yet. The formal description of the algorithm uses the following definition.

Definition 11.3. Let $G = (V, E)$ be a graph and $U \subset V$ a subset of the vertices, then $\delta(U)$ is the set of all edges with exactly one endpoint in U:

$$\delta(U) := \{\{u, v\} \in E : u \in U \text{ and } v \notin U\}.$$

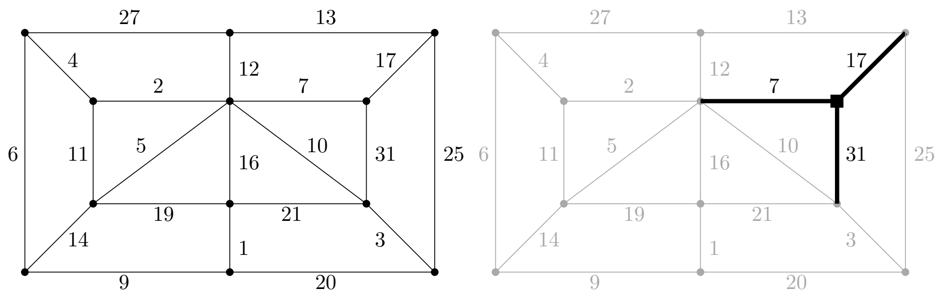


Figure 11.3: Left: an MST problem. Right: The situation at the beginning of the first iteration of Prim-Dijkstra for the MST problem at the left. Square nodes are nodes in U, dashed edges are edges in F, thick black edges are edges in $\delta(U)$.

Now the algorithm is quite simple.

```
Data: An instance of MINIMUM SPANNING TREE ($G = (V, E), l : E \to \mathbb{R}$).
Result: a minimum spanning tree $T$ of $G$.
1 Pick an arbitrary vertex $v_1 \in V$;
2 Set $U_1 := \{v_1\}$;
3 Set $F_1 := \emptyset$;
4 for $k = 1, 2, \dots, |V| - 1$ do
5     Pick an edge $e_k \in \delta(U_k)$ of minimal length;
6     Set $U_{k+1} := U_k \cup e_k$;
7     Set $F_{k+1} := F_k \cup \{e_k\}$;
8 end
9 return $T = (V, F_{|V|})$
```

Algorithm 3: The Prim-Dijkstra algorithm for MINIMUM SPANNING TREE

Note that $U_{k+1} := U_k \cup e_k$ means that we add both endpoints of the edge e_k to the set of vertices U_k, giving the set of vertices U_{k+1}. On the other hand, $F_{k+1} := F_k \cup \{e_k\}$ means that we add the edge e_k to the set of edges F_k, giving the set of edges F_{k+1}.

11.2.1 An example of Prim-Dijkstra

We now use the algorithm as an example. We start with the graph and lengths as shown at the top in Figure 11.3. To start the algorithm, we pick an arbitrary vertex v_1, the upper right vertex of the inner rectangle. The initial set U of vertices that we already connected consists of only this vertex v_1, and we look at $\delta(U)$, the edges with endpoint v_1 (Figure 11.3 bottom left). The shortest of these edges has length 7. We add its other endpoint to U, and add the edge to the forest F. This concludes the first iteration of the while loop.

For the second iteration, we again look for the edge leaving U of shortest length. We add this edge of length 2 to F and its endpoints to U. This is repeated in each next iteration. Note how the set $\delta(U)$ changes in each iteration: outgoing edges of the new node are added, and edges that now would induce a cycle if added to F are deleted.

At the end of the eleventh iteration of the loop, we have picked $11 = |V| - 1$ edges for F. Because we only add edges that do not create a cycle, these edges must form a spanning tree, and the algorithm terminates. The spanning tree we find (Figure 11.7) should be an MST, but proving this requires a theorem which we study in the next section.

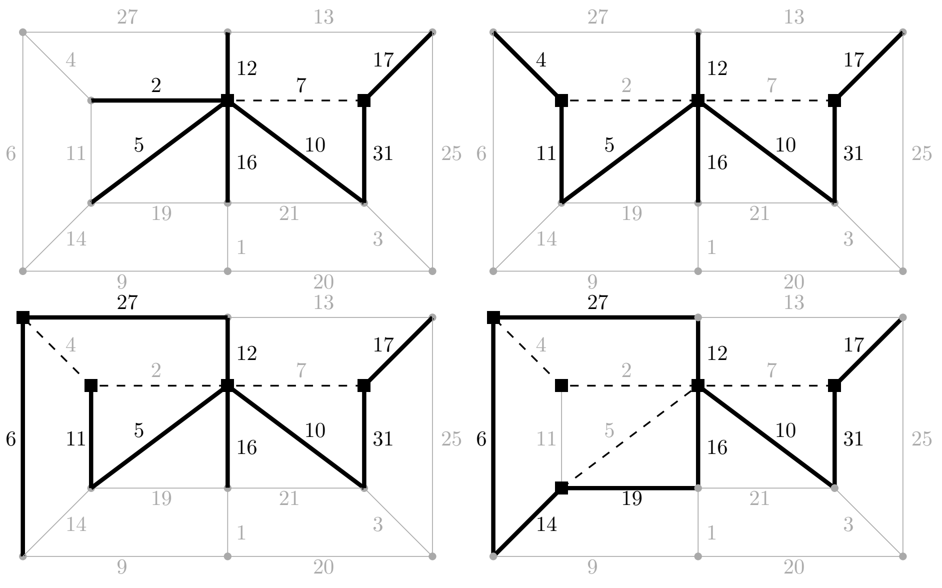


Figure 11.4: Iterations 2, 3, 4 and 5 of Prim-Dijkstra.

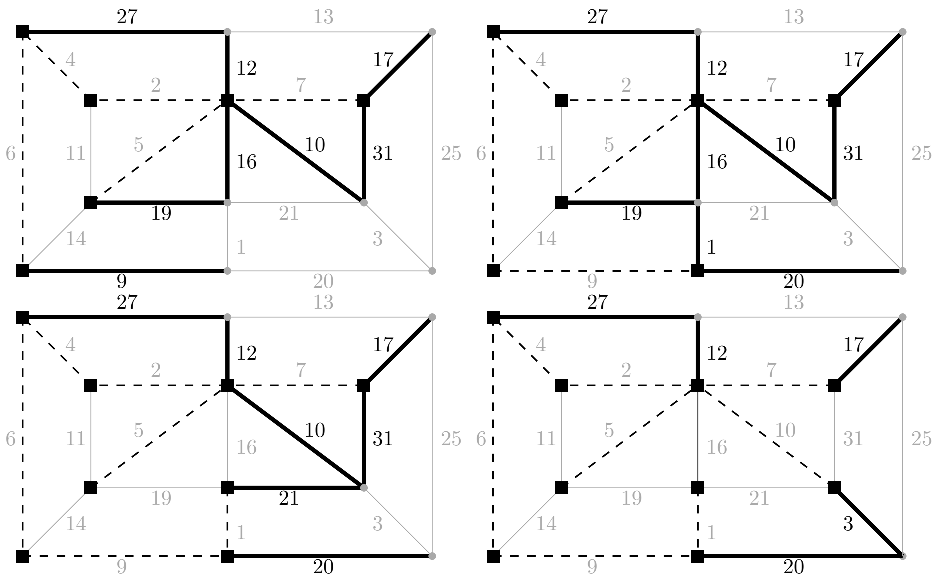


Figure 11.5: Iterations 6, 7, 8 and 9 of Prim-Dijkstra.

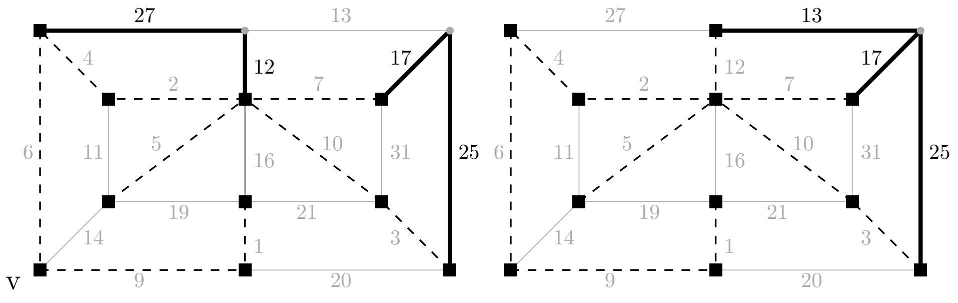


Figure 11.6: Iterations 10 and 11 of Prim-Dijkstra.

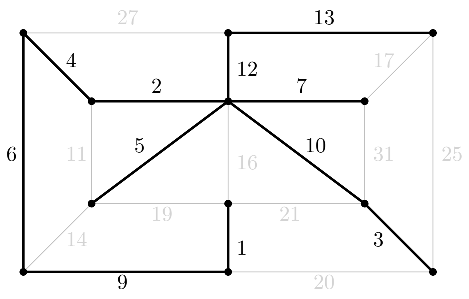


Figure 11.7: The spanning tree found by Prim-Dijkstra.

11.2.2 Correctness of Prim-Dijkstra

Correctness of the algorithm follows from the following definition and theorem.

Definition 11.4. A forest (V, F) is a *forest of* $G = (V, E)$ if $F \subseteq E$.

Theorem 11.1. Suppose (V, F) is a forest of $G = (V, E)$ and $U \subseteq V$ a connected component of (V, F). Let $e \in \delta(U)$ be an edge of minimal length in $\delta(U)$. Then there exists a spanning tree of G containing all edges in $F \cup \{e\}$ which has minimal length over all spanning trees of G that contain F.

Proof. Suppose for the sake of contradiction that there is a spanning tree (V, F') of G with $e \notin F'$ with total length less than the total length of each spanning tree of G that contains $F \cup \{e\}$. Note that $(V, F' \cup \{e\})$ contains a unique cycle which consists of e together with the path in (V, F') between the endpoints of e (Lemma 11.1 Part 1). Because exactly one of the endpoints of e is in U, the cycle contains an edge $f \in \delta(U)$ leaving U. By the second part of Lemma 11.1, removing f and adding e gives a spanning tree $(V, F' \cup \{e\} \setminus \{f\})$ of G. This new spanning tree contains $F \cup \{e\}$, and must have greater total length than (V, F') by our initial assumption. This means

$$\begin{aligned} l((V, F' \cup \{e\} \setminus \{f\})) &> l((V, F')) \\ \sum_{x \in F' \cup \{e\} \setminus \{f\}} l(x) &> \sum_{x \in F'} l(x) \\ l(e) + \sum_{x \in F' \setminus \{f\}} l(x) &> l(f) + \sum_{x \in F' \setminus \{f\}} l(x) \\ l(e) &> l(f). \end{aligned}$$

However, e was an edge of minimal length in $\delta(U)$, which contains both e and f, a contradiction. We conclude that there is no spanning tree (V, F') of G with $e \notin F'$ with total length less than the total length of each spanning tree of G that contains $F \cup \{e\}$. □

By this theorem, the Prim-Dijkstra algorithm is correct. Indeed, in each iteration we make sure we create a set of edges F_k which is contained in some minimum spanning tree. After the last iteration, we have obtained a set $F_{|V|}$ of $|V| - 1$ edges that is contained in some minimum spanning tree. Since there are $|V|$ vertices and the set $F_{|V|}$ contains $|V| - 1$ edges, it is not possible to add any edges to it and keep a tree (see Lemma A.1). Hence, the edges in $F_{|V|}$ are in fact a minimum spanning tree.

11.2.3 Running time of Prim-Dijkstra

Theorem 11.2. The running time of Prim-Dijkstra is $O(|V|^2)$.

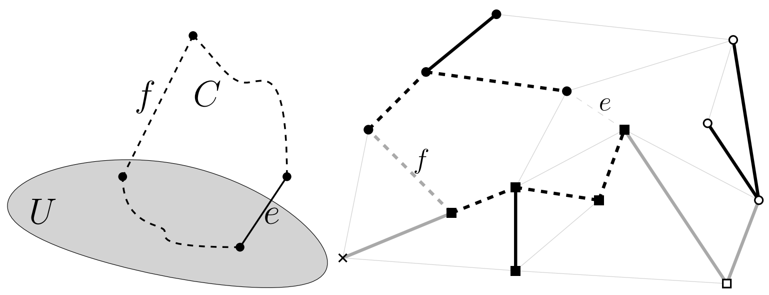


Figure 11.8: An illustration of the proof of Theorem 11.1. Left: an abstract representation of the situation. The cycle C consists of the dashed path P through F' and the edge e. The cycle must at some point leave U, this happens at the edge f, which is longer than e. Right: A concrete example in a given graph. The thick edges represent the current spanning tree, the black edges are the edges in the forest whose connected components each have a different vertex shape, the cycle consists of the dashed edges.

Proof. To find this running time, we should explain in more detail how the step of line 5 of Algorithm 3 works. This step says we find an edge in $\delta(U_k)$ of minimal length. To do this efficiently, for each node v in $V \setminus U_k$ we keep track of the edge (u, v) with $u \in U_k$ of minimal length (if it exists). Initialising this list entails going through all vertices $v \in V \setminus \{u_1\}$ once, and checking if the edge (v, u_1) exists. To update this list after adding u_{k+1}, we again check for each $v \in V \setminus U_{k+1}$ whether (v, u_{k+1}) is shorter than the previously shortest edge from v to U_k. This takes $O(|V|)$ steps. Now, as we have to update this list $|V| - 1$ times and the other steps take constant time in $|V|$, the total running time of the algorithm is $O(|V|^2)$. □

The implementation we assume in the previous theorem is not optimal. It is possible to make use of "Fibonacci Heaps" for an implementation with running time $O(|E| + |V| \log(|V|))$. We will not describe these formally here, but the main idea is as follows. As in the proof above, we store for each node v in $V \setminus U_k$ the length of a shortest edge (u, v) with $u \in U_k$. However, we store these values in a clever way such that we can find a smallest value quickly, and also update the data structure quickly. We omit the details.

11.3 Two more algorithms for MST

11.3.1 Borůvka's algorithm

The following algorithm was the first algorithm for MST, discovered in 1926.

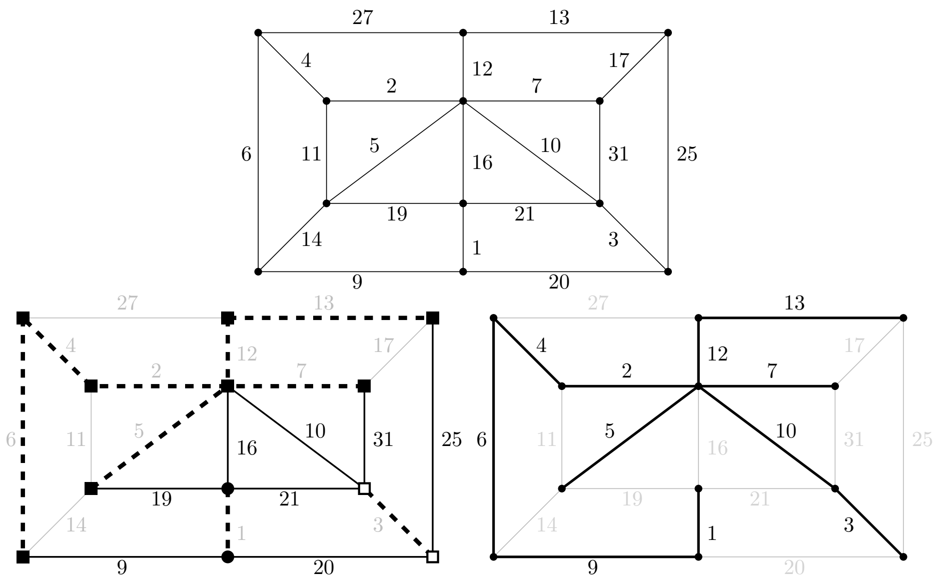


Figure 11.9: An example of Borůvka's algorithm, which solves the MST problem for the graph on the top in two iterations. The current forest after the first iteration (bottom left) is indicated with dashed edges, and the components of the forest each have their own vertex shape.

```
Data: An instance of MINIMUM SPANNING TREE ($G = (V, E), l : E \to \mathbb{R}$) where no two
      edges have the same length.
Result: a minimum spanning tree $T$.
1 Set $F := \emptyset$;
2 while $|F| < |V| - 1$ do
3     Let $U_1, \dots, U_k$ be the connected components of $(V, F)$;
4     for $i = 1, \dots, k$ do
5         Pick an edge $e_k \in \delta(U_k)$ of minimal length;
6     end
7     Set $F := F \cup \{e_1, \dots, e_k\}$;
8 end
9 return $T = (V, F)$
```

Algorithm 4: Borůvka's algorithm for MINIMUM SPANNING TREE

Theorem 11.3. Borůvka's algorithm (Algorithm 4) is correct and has running time $O(|E| \log(|V|))$.

Borůvka's algorithm can be done in quite few iterations, as exemplified by Figure 11.9. The proof of correctness is left as an exercise (Exercise 11.6). For the running time, it is easy to see that each iteration takes $O(E)$ time. Since the number of connected components is halved (in the worst case) in each iteration, at most $\log(|V|)$ iterations are needed.

An example of the use of Borůvka's algorithm can be found in Figure 11.9. It needs only two iterations to find the MST. At the start of the algorithm F is empty, so each vertex is a component. Hence in the first iteration, we add to F the shortest edge leaving each node. Note that the number of components decreases in each iteration. After the first iteration, the forest has only three components left. With the second iteration, each of these components is connected, too. This gives the MST.

11.3.2 Kruskal's algorithm

Possibly the simplest algorithm for MST was discovered in 1956 by Joseph Kruskal. It is an example of a "greedy" algorithm.

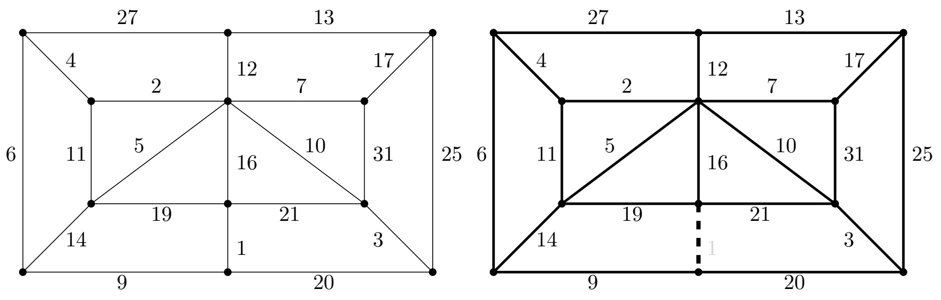


Figure 11.10: Left: an MST problem. Right: The situation at the beginning of the first iteration of Kruskal for the MST problem at the left. Dashed edges are edges in F, grey edges are edges whose addition to F creates a cycle.

```
  Data: An instance of MINIMUM SPANNING TREE ($G = (V, E, l), l : E \to \mathbb{R}$).
  Result: a minimum spanning tree $T$ of $G$.
1 Set $F := \emptyset$;
2 for $k = 1, 2, \ldots, |V| - 1$ do
3 |   Pick an edge $e_k \in E \setminus F$ of minimal length for which $(V, F \cup \{e\})$ is a forest;
4 |   Set $F := F \cup \{e_k\}$;
5 end
6 return $T = (V, F)$
```

Algorithm 5: Kruskal's algorithm for MINIMUM SPANNING TREE

Theorem 11.4. Kruskal's algorithm (Algorithm 5) is correct and has running time $O(|E| \log(|V|))$.

Correctness is proven in Exercise 11.7. The running time bound is again easy to see when noting that sorting the edges on their length takes $O(|E| \log(|E|))$ time, which is $O(|E| \log(|V|))$.

Again, we look at an example of using the algorithm; we use the same graph as before. The initialization of Kruskal's algorithm consists of setting the current forest to $F = \emptyset$. Then in each iteration, we add the shortest edge to F that does not induce a cycle. In the first iteration, this means we just add the shortest edge in the graph. This is the edge of length 1 (Figure 11.10).

In each of the following iterations, we add a shortest edge that does not induce a cycle to F (Figures 11.11, 11.12, and 11.13). After $|V| - 1 = 11$ iterations, we added $|V| - 1$ edges to F without creating cycles, hence F is a spanning tree. It is minimal by choice of the minimal length edge in each iteration, the reader is invited to prove this in Exercise 11.7.

11.4 Exercises

Exercise 11.1. Prove that a graph G has a spanning tree if and only if G is connected.

Exercise 11.2. Does there exist a graph with 100 vertices and exactly two spanning trees?

Exercise 11.3. Find a minimum spanning tree for the graph shown below using the following algorithms:

(a) Prim-Dijkstra

(b) Borůvka

(c) Kruskal

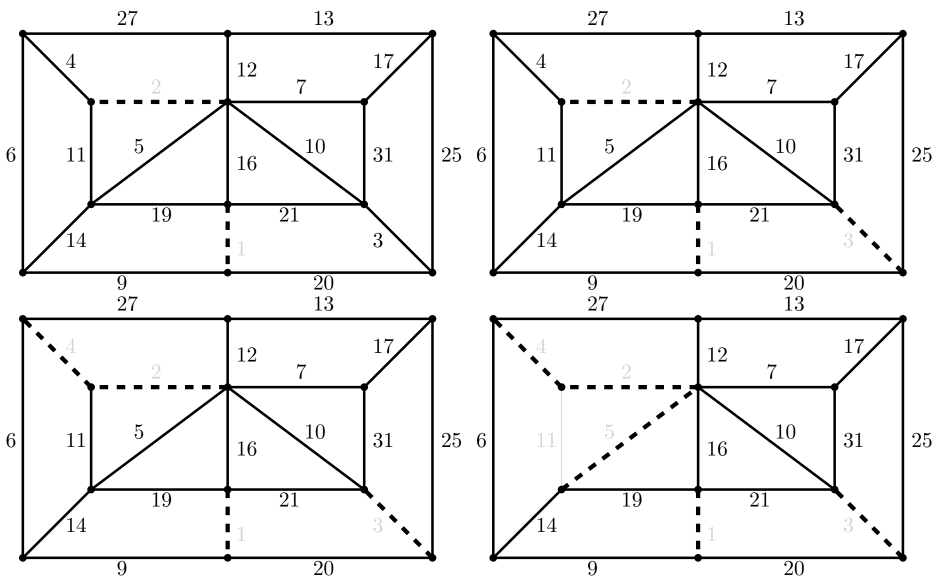


Figure 11.11: Iterations 2, 3, 4 and 5 of Kruskal's algorithm. Dashed edges are edges in F, grey edges are edges whose addition to F creates a cycle.

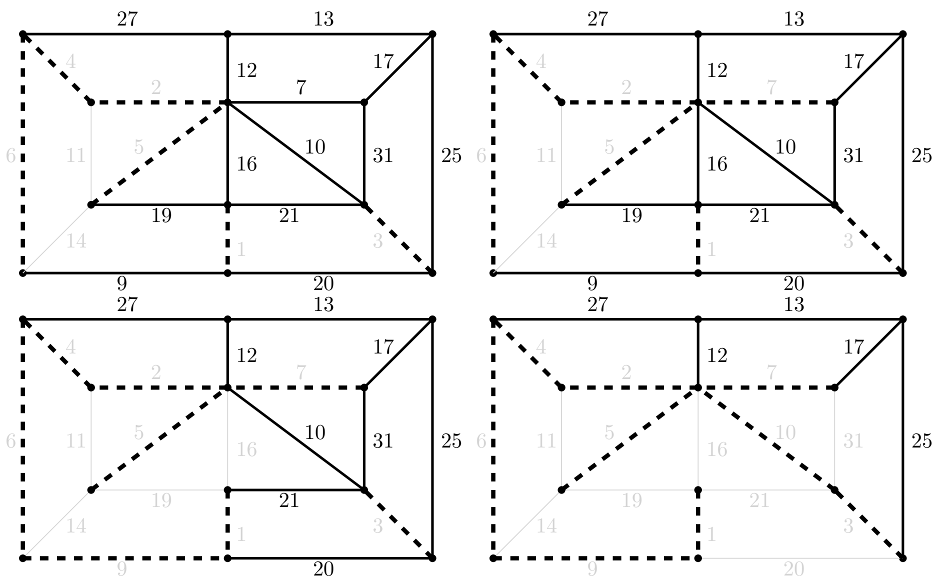


Figure 11.12: Iterations 6, 7, 8 and 9 of Kruskal's algorithm. Dashed edges are edges in F, grey edges are edges whose addition to F creates a cycle.

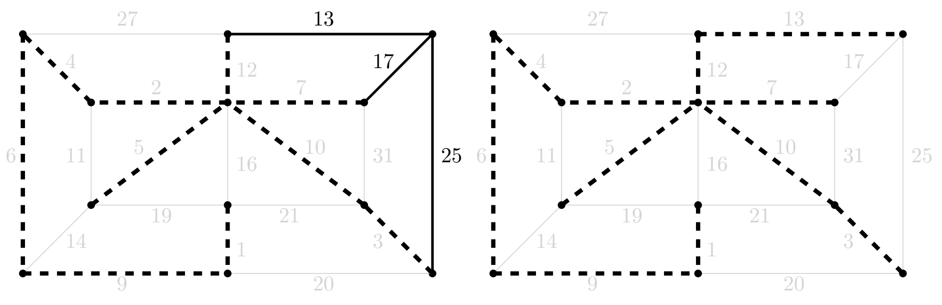
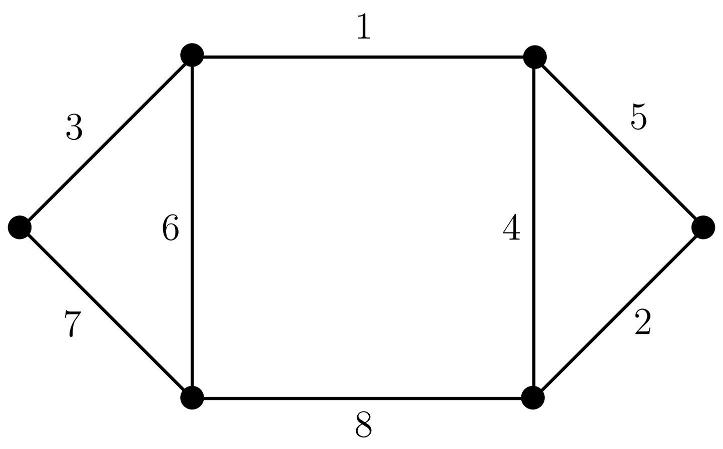


Figure 11.13: Iterations 10 and 11 of Kruskal's algorithm. Dashed edges are edges in F, grey edges are edges whose addition to F creates a cycle.



Exercise 11.4. Let $T_1 = (V, F_1)$ and $T_2 = (V, F_2)$ be two spanning trees of $G = (V, E)$ and $e \in F_1$ an edge of T_1. Prove that there exists an edge $f \in F_2$ of T_2 such that $(V, F_1 \setminus \{e\} \cup \{f\})$ is also a spanning tree of G.

Exercise 11.5. Let e be an edge in a cycle C of a graph G, and suppose e is longer than all other edges of C. Prove that no minimum spanning tree of G contains e.

Exercise 11.6. Prove that Borůvka's algorithm finds a minimum spanning tree.

Exercise 11.7. Prove that Kruskal's algorithm finds a minimum spanning tree.

Exercise 11.8. Let $G = (V, E)$ be a graph with length function $l : E \to \mathbb{R}$. Prove that if $U \subset V$ and $e \in \delta(U)$ such that

$$l(e) < l(f) \qquad \forall f \in \delta(U) \text{ with } f \neq e$$

then each minimum spanning tree of G contains the edge e.

Part IV

Integer Linear Programming

Chapter 12

LP-Relaxations

We have seen that many problems can be formulated as integer linear programming problems (ILPs). We have also seen some methods to solve linear programming problems (LPs). Because of the integrality constraints we cannot use the same LP-solving methods to solve ILPs directly. Still, studying LPs in the context of ILP can give us important information.

Here, we look at an LP problem related to our ILP, which we call the *LP-relaxation.* The solutions for the LP-relaxation are often better with respect to the objective value, but they are typically infeasible with respect to integrality. In this chapter we will study the *integrality gap*, a measure for how different the solutions to the ILP and the LP-relaxation are. We will also see that there is a class of ILPs where solving the LP-relaxation actually are guaranteed to give the optimal solutions for the ILP!

12.1 LP-relaxation example

We start by defining what we mean by a *relaxation.*

Definition 12.1. A problem

$$\begin{aligned} \max f(\boldsymbol{x}) & \\ \text{s.t. } \boldsymbol{x} &\in F \subset \mathbb{R}^n \end{aligned}$$

is a *relaxation* of the problem

$$\begin{aligned} \max g(\boldsymbol{x}) & \\ \text{s.t. } \boldsymbol{x} &\in X \subset \mathbb{R}^n \end{aligned}$$

if

(i) $X \subseteq F$

(ii) $f(x) \geq g(\boldsymbol{x})$

In particular, the problem

$$\begin{aligned} \max \boldsymbol{c}^{\mathsf{T}} \boldsymbol{x} & \\ \text{s.t. } A\boldsymbol{x} &\leq \boldsymbol{b} \\ \boldsymbol{x} &\geq \boldsymbol{0} \end{aligned}$$

is called the *LP-relaxation* of the IP

$$\begin{aligned} \max \boldsymbol{c}^{\mathsf{T}} \boldsymbol{x} & \\ \text{s.t. } A\boldsymbol{x} &\leq \boldsymbol{b} \\ \boldsymbol{x} &\geq \boldsymbol{0} \\ \boldsymbol{x} &\in \mathbb{Z}^n \end{aligned}$$

To give an example, suppose you have to work two jobs to live. One of the jobs pays more, but you can only work there for 3 days a week. The contract of the lower paying job states that this must be your main job, i.e., you must make at least as many hours on this job as on any other job. You highly value your free time, and hence want to minimize your total time spent on work. To live, you have to make \$500 per week. The lower paying job pays \$100 per day, and the higher paying job pays \$125 per day. Both jobs demand from you that you make full days. This JobCombination problem can be formulated as an ILP. The formulation and the feasible region are as follows.

$$\begin{array}{llrcrcrl} \min & z = & x_1 & + & x_2 & & & \\ \text{s.t.} & & 100x_1 & + & 125x_2 & \geq & 500 & (1) \\ & & x_1 & - & x_2 & \geq & 0 & (2) \\ & & & & x_2 & \leq & 3 & (3) \\ & & \multicolumn{5}{l}{x_1,\ x_2 \geq 0} & \\ & & \multicolumn{5}{l}{x_1,\ x_2 \in \mathbb{Z}} & \end{array}$$

You think it is preposterous that you can only work full days, and you demand from both your bosses they let you work flexible times. They both concede and you are allowed to work any part of the day. This corresponds to removing the integrality constraint $x_1,\ x_2 \in \mathbb{Z}$ from the problem formulation. The resulting linear program is called the LP-relaxation of the ILP.

Note that in any optimal solution of the original problem, you had to work $z^*_{ILP} = 5$ full days. In the relaxed problem you need to work about $z^*_{LP} \approx 4.5$ days, and you have half a day more to relax. This solution is obviously not feasible if you have to work full days.

In the next section we will see that the solution to the LP-relaxation always has an objective value z^*_{LP} at least as good as the optimal solution of the original ILP z^*_{ILP}. We will also look at the integrality gap, a measure for the difference between z^*_{ILP} and z^*_{LP}.

12.2 LP-relaxations and integrality gaps

In the previous section we saw an example of an LP-relaxation. Each ILP has an *LP-relaxation* obtained by removing the integrality constraint $\boldsymbol{x} \in \mathbb{Z}^n$. The following pair of optimization problems shows a minimization ILP in standard form and its LP-relaxation.

The ILP:

$$\begin{array}{lrl} \min & z_{ILP} = & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.} & & A\boldsymbol{x} = \boldsymbol{b} \\ & & \boldsymbol{x} \geq \boldsymbol{0} \\ & & \boldsymbol{x} \in \mathbb{Z}^n \end{array}$$

The LP-relaxation:

$$\begin{array}{lrl} \min & z_{LP} = & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.} & & A\boldsymbol{x} = \boldsymbol{b} \\ & & \boldsymbol{x} \geq \boldsymbol{0} \end{array}$$

Each feasible point for the ILP is a feasible point of the LP-relaxation. This is because the feasible region of the ILP is exactly $\mathbb{Z}^n \cap P$, where P is the feasible region of the LP-relaxation. Note that the objective functions of both problems are the same. Therefore, the optimal objective function value for the LP-relaxation is always at least as good as for the ILP problem. We can describe this with the following inequality

$$z^*_{ILP} \geq z^*_{LP}, \tag{12.1}$$

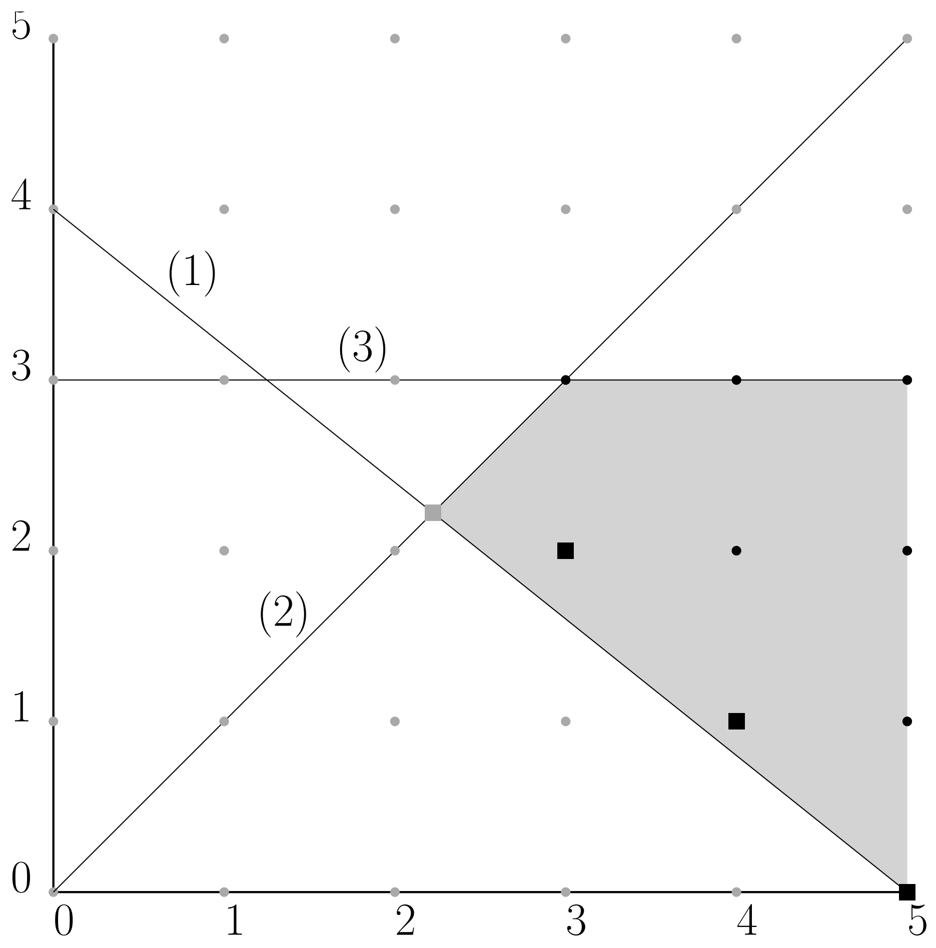


Figure 12.1: The feasible region for the JobCombination problem. The grey area is the feasible polytope for the LP-relaxation, the black dots are the feasible points for the ILP. Squares indicate optimal points for the ILP (black) and the LP (grey). The restrictions are numbered as in the ILP formulation.

where z^*_{ILP} denotes the optimal value of the ILP and z^*_{LP} the optimal value of the LP-relaxation. This tells us that solving the LP-relaxation gives us a lower bound on the optimal value for the ILP problem (and for maximization problems it gives an upper bound).

Sometimes we can sharpen this bound slightly by using integrality of solutions: if all coefficients of the objective function are integral, then all feasible values of the objective function are also integral. Hence we can round the bound to the nearest worst integer in such cases.

Lemma 12.1. Let $z_{ILP} = \boldsymbol{c}^T\boldsymbol{x}$ be the objective function of a minimization ILP problem with $\boldsymbol{c} \in \mathbb{Z}^n$ integral, and z_{LP} the objective function of the corresponding LP-relaxation. Then the optimal values for these problems are related in the following way

$$z^*_{ILP} \geq \lceil z^*_{LP} \rceil. \tag{12.2}$$

12.2.1 Integrality gap

In the last section, we noted that the solution to the LP-relaxation gives a lower bound on the optimal value of a (minimization) ILP. In this section, we will try to quantify how good these lower bounds are using the *integrality gap* which is defined as the worst-case value of z^*_{ILP}/z^*_{LP}. More formally, if $\mathcal{P}$ is a problem, i.e. a set of ILP instances, the integrality gap of $\mathcal{P}$ is defined as

$$\sup_{I \in \mathcal{P}} \frac{z^*_{ILP}(I)}{z^*_{LP}(I)},$$

where $z^*_{ILP}(I)$ denotes the optimal value for an instance $I \in \mathcal{P}$, and similarly $z^*_{LP}(I)$ for the optimal value of the LP-relaxation.

We look at the worst-case value, because the integrality gap is used to assess the quality of LP-relaxations for a certain ILP formulation of a class of optimization problem instances.

A good example is the Traveling Salesman Problem, which is actually a class of instances given by different digraphs and weights on their arcs. Here, we focus on the following variant of TSP.

Traveling Salesman Problem (TSP)
Given: a directed graph $D = (V, A)$ and a length $c_{ij} \in \mathbb{Z}$ for each arc $(i, j) \in A$.
Find: a directed Hamilton cycle in D with minimum length.

There are different ways of describing this problem as an ILP, including the following two ILP formulations, which assume that $V = \{1, \ldots, n\}$.

$$\begin{array}{llll}
\min z = & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & & \\
s.t. & \displaystyle\sum_{i=1}^{n} x_{ij} = 1 & j = 1, \ldots, n & (A) \\
& \displaystyle\sum_{j=1}^{n} x_{ij} = 1 & i = 1, \ldots, n & (B) \\
& \displaystyle\sum_{i \in S, j \in V \setminus S} x_{ij} \geq 1 & \forall S \subsetneq V, S \neq \emptyset & (C) \\
& x_{ij} = 0 & \forall i, j \mid (i, j) \notin A & (D) \\
& x_{ij} \in \{0, 1\} & \forall i, j \mid (i, j) \in A & (E)
\end{array}$$

and

$$
\begin{array}{llll}
\min z = & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & & \\
s.t. & \sum_{i=1}^{n} x_{ij} = 1 & j = 1,\ldots,n & (A) \\
 & \sum_{j=1}^{n} x_{ij} = 1 & i = 1,\ldots,n & (B) \\
 & u_i - u_j + nx_{ij} \le n-1 & 2 \le i,j \le n \,\text{and}\, i \ne j & (C') \\
 & x_{ij} = 0 & \forall i,j \mid (i,j) \notin A & (D) \\
 & x_{ij} \in \{0,1\} & \forall i,j \mid (i,j) \in A & (E) \\
 & u_i \in \mathbb{R} & i = 1,\ldots,n. & (F)
\end{array}
$$

At first sight, one may think that the second formulation is better than the first one because it only has a polynomial number of constraints while the first formulation has an exponential number of constraints (and the number of variables is the same). However, it turns out that in practice the first formulation is actually much more useful than the second one, which can be explained as follows.

It can be shown that the first formulation has a so-called *strong LP-relaxation*, i.e., the integrality gap is small. This means that for all instances of TSP, the fraction z^*_{ILP}/z^*_{LP} is small. Therefore, solving the LP-relaxation gives very good bounds, which is very useful when solving the ILP (see Chapter 13).

On the other hand, the second ILP formulation has a *weak LP-relaxation*, meaning that the integrality gap is big. Hence, solving the LP-relaxation does not give us good bounds, which makes it much harder to solve the ILP.

12.3 Rounding

In the previous sections we have seen that LP-relaxations give us lower bounds on the optimal value of the ILP (for minimization problems). We have not yet addressed the problem of finding some good feasible solution for the ILP. LP-relaxations can sometimes help us here, too. An obvious way of transforming an LP solution into an integer solution, is by rounding the values of the variables.

There may be a problem with this procedure. Even though the rounded solution will always be integer, it might not be feasible. An example of an ILP where this is the case can be found in Figure 12.2.

In certain cases rounding does give feasible solutions. We will encounter a few of those in the exercises.

12.4 A Totally and Utterly Manageable ILP

Suppose we want to find a shortest path through the graph below. We have seen there is an ILP formulation for the shortest path problem (see Section 6.4). Consider for example the graph below.

Then the ILP formulation for the shortest path problem becomes the following.

$$
\begin{array}{lllllllll}
\min & z = & 1x_{sr} & + & 1x_{rt} & + & 3x_{st} & & \\
\text{s.t.} & & x_{sr} & + & & & x_{st} & = 1 & (1) \\
 & & -x_{sr} & + & x_{rt} & & & = 0 & (2) \\
 & & & & -x_{rt} & + & -x_{st} & = -1 & (3) \\
 & & \multicolumn{5}{l}{x_{st},\ x_{sr},\ x_{rt} \ge 0} & & \\
 & & \multicolumn{5}{l}{x_{st},\ x_{sr},\ x_{rt} \in \mathbb{Z}} & &
\end{array}
$$

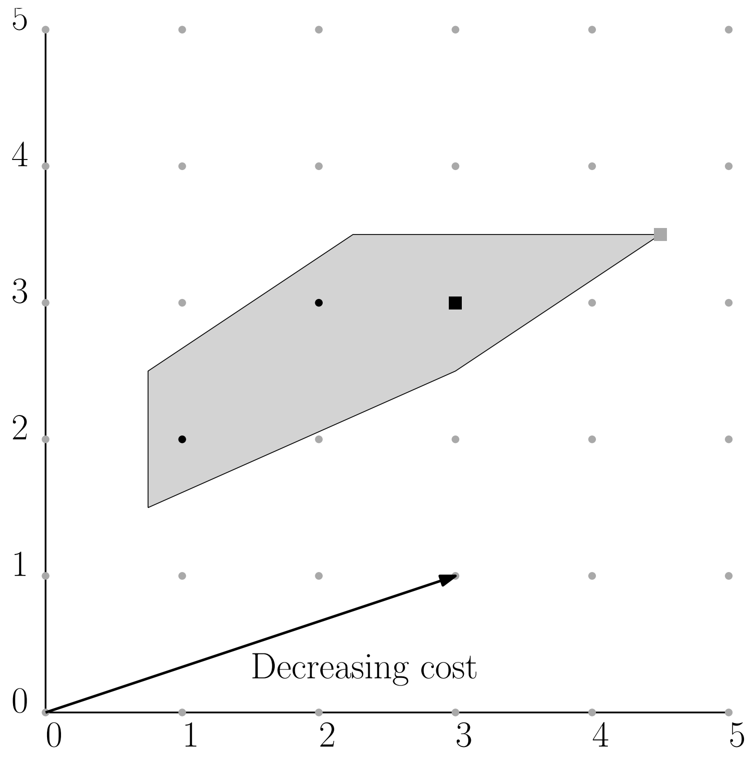
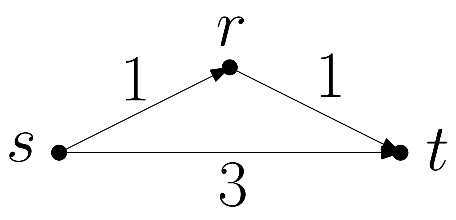


Figure 12.2: An example of an ILP in which rounding the optimal LP-relaxation solution (in any way) does not give a feasible ILP solution. The grey square is the optimal solution of the LP-relaxation while the black square is the optimal solution of the ILP.



A naive way of solving this problem is by using Simplex to find an optimal solution to the LP-relaxation. Let us see what would happen by investigating the polytope. The feasible region is exactly the line segment between $(1, 1, 0)$ and $(0, 0, 1)$ (corresponding to the path via r and the direct path respectively). Equivalently, it is the convex hull of the two integer points $(1, 1, 0)$ and $(0, 0, 1)$. This means that Simplex will find an integer solution to this problem, for any objective function. Although we want to solve an ILP, we can just use the algorithm for solving an LP!

The example above is quite simple. You might wonder whether the nice property we saw was a result of taking a simple instance. This is not the case, as it turns out that for any shortest path instance, the vertices of the feasible region are integer, using the following ILP formulation.

We use the node-arc incidence matrix

$$a_{ij} = \begin{cases} 1 & \text{if arc } e_j \text{ leaves } i \\ -1 & \text{if arc } e_j \text{ enters } i \\ 0 & \text{otherwise.} \end{cases}$$

to define the polytope. The ILP formualtion then becomes as follows, where we assume that the first row of the matrix corresponds to the source node, and the last corresponds to the target node:

$$\begin{array}{lrl} \min & z = & \sum_{e \in E} w(e) x_e \\ \text{s.t.} & A\boldsymbol{x} = & (1, 0, \ldots, 0, -1)^\mathsf{T} \\ & & \boldsymbol{x} \geq \mathbf{0} \\ & & \boldsymbol{x} \in \mathbb{Z}^n \end{array}$$

In this chapter, we will see why these ILP instances have a feasible region with only integer vertices. We will also see that the property of having only integer vertices is only dependent on the coefficient matrix A (and independent of the integral vector $\boldsymbol{b}$). We will also characterize a bigger class of matrices that have the same property, and we will find a few ways to show that a matrix is in that class. This is useful for identifying problems that we can solve by just using Simplex on the LP-relaxation.

12.5 Totally unimodular matrices

The class we mentioned earlier is the class of *totally unimodular matrices*, or *TUMs*. These matrices are quite technically defined by the determinants of their submatrices.

A square matrix is called *unimodular* if its determinant is either 1 or -1. A matrix is *totally unimodular (TUM)* if each nonsingular square submatrix is unimodular. Remember that a matrix is singular precisely when its determinant is 0. This means we can also define a totally unimodular matrix as a matrix whose square submatrices have determinant -1, 0 or 1. Note that a submatrix can be obtained by taking any selection of rows and columns (they do not need to be adjacent).

The following two theorems, which we state without proof, show that ILPs can be solved efficiently when the coefficient matrix A is totally unimodular and the right-hand side vector $\boldsymbol{b}$ integral.

Theorem 12.1. If an $m \times n$ matrix A is totally unimodular and $\boldsymbol{b} \in \mathbb{Z}^m$, then all vertices of the polyhedron

$$\{\boldsymbol{x} \in \mathbb{R}^m | A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq 0\}$$

are integer.

Theorem 12.2. If an $m \times n$ matrix A is totally unimodular and $\boldsymbol{b} \in \mathbb{Z}^m$, then all vertices of the polyhedron

$$\{\boldsymbol{x} \in \mathbb{R}^m | A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq 0\}$$

are integer.

The following theorem gives a sufficient condition for total unimodularity. It is important to remember that this is only a sufficient condition, so matrices not satisfying these conditions may still be totally unimodular!

Theorem 12.3. Let A be an integral matrix with $a_{ij} \in \{0, -1, 1\}$, then A is TUM if all the following hold:

- each column contains at most two non-zero entries;
- there is a partition of the rows of A into two sets I_1 and I_2 such that:
    1. if a column contains two elements of the same sign, then the corresponding rows belong to different sets of the partition and
    2. if a column contains two elements of different signs, then the corresponding rows belong to the same set of the partition.

Using this theorem, we can prove the claim of last section saying that any shortest path instance can be solved using Simplex.

Theorem 12.4. Let A be a TUM matrix and let I denote the identity matrix, then $[A|I]$ is a TUM matrix.

The proof is an exercise (Exercise 12.2).

12.6 Exercises

Exercise 12.1. We study the class of ILPs of the form

$$\begin{aligned} \min \quad & \boldsymbol{c}^\mathsf{T}\boldsymbol{x} \\ \text{s.t.} \quad & A\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \in \mathbb{Z}^n \end{aligned}$$

with A TUM and $\boldsymbol{b}$ integral. The cost vector $\boldsymbol{c}$ does not have to be integral. Indicate which of the following are true.

(a) The integrality gap for this class is 1.

(b) Each feasible point of the ILP is on the border of the feasible polytope of the LP-relaxation.

(c) Each optimal point of the LP-relaxation is an optimal point for the ILP.

(d) For each optimal point of the LP-relaxation, rounding each variable down to an integer will give a feasible point for the ILP.

Exercise 12.2. Give a proof of Theorem 12.4.

Exercise 12.3. (a) Indicate whether the following matrices are, or are not TUM (totally unimodular). Also give an argument for why they are or are not TUM.

$$X = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & 1 & 1 & 0 \\ -1 & -1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

(b) Let A be an $m \times n$ totally unimodular matrix. Show that A^T en

$$\begin{bmatrix} -I \\ A^\mathsf{T} \\ -A^\mathsf{T} \end{bmatrix}$$

are also totally unimodular.

(c) You may use the following known result: *if A is TUM and $\boldsymbol{b}$ is integer, then the polyhedron $P = \{\boldsymbol{x} \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$ only has integer vertices.*

Show that for each TUM A and integer vectors $\boldsymbol{c}$ and $\boldsymbol{b}$, each of the problems in the following primal dual pair have an integer solution.

$$\begin{array}{ll} \max & \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} \\ \text{s.t.} & A\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \in \mathbb{R}^n \end{array} \qquad \text{and} \qquad \begin{array}{ll} \min & \boldsymbol{b}^{\mathsf{T}}\boldsymbol{\pi} \\ \text{s.t.} & A^{\mathsf{T}}\boldsymbol{\pi} = \boldsymbol{c} \\ & \boldsymbol{\pi} \geq \boldsymbol{0} \end{array}$$

Exercise 12.4. Show that the following proposition (the converse of Theorem 12.1) is not true. If for all integer $\boldsymbol{b}$ the polyhedron

$$\{\boldsymbol{x} \in \mathbb{R}^m | A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$$

has only integer vertices, then A is an $n \times m$ TUM matrix.

Exercise 12.5. Consider the following ILP problem, where all elements of $A, \boldsymbol{b}$ and $\boldsymbol{c}$ are strictly positive.

$$\begin{array}{ll} \max & \boldsymbol{c}^T\boldsymbol{x} \\ s.t. & A\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \\ & \boldsymbol{x} \in \mathbb{Z}^n \end{array}$$

Let $\boldsymbol{x}^*$ be an optimal solution to the ILP, and let $\boldsymbol{x}^{LP}$ be an optimal solution to the LP-relaxation.

(a) Show that $\lfloor \boldsymbol{x}^{LP} \rfloor$ is a feasible solution of the ILP.

(b) Show that the difference in objective functions for $\lfloor \boldsymbol{x}^{LP} \rfloor$ and $\boldsymbol{x}^*$ is not greater than

$$\sum_{i=1}^{n} c_i.$$

Chapter 13

Branch and Bound

In last chapter we saw that some ILPs can be solved by simply solving the LP relaxation. However, for most ILPs this does not work and we have not yet seen any method to actually solve ILPs in general.

That is what we will do in this chapter, using the method Branch and Bound. This method consists of repeatedly branching into subproblems. The obtained subproblems are organized in a tree. We use bounds to "prune off" parts of the tree. For example, if for a certain subproblem we find out that it cannot have a feasibly solution better than the best feasible solution we already found, then we do not need to investigate this subproblem or any subproblems below it any further, so we "prune" this part of the tree.

13.1 An example of Branch and Bound

Let us solve an ILP using Branch and Bound but without first studying the formal description of this method. We can do this because each step should make sense on its own and each step will give us more information about the actual solution.

Suppose we want to solve the following ILP:

$$\begin{aligned} z_{IP} = \min\ & -4\,x_1 + x_2 & & \\ s.t.\ \ & \ 7\,x_1 - 2\,x_2 \le 14 & & (1) \\ & \qquad\qquad\ \ x_2 \le \ \ 3 & & (2) \\ & \ 2\,x_1 - 2\,x_2 \le \ \ 3 & & (3) \\ & x_1, x_2 \ge 0 & & \\ & x_1, x_2 \in \mathbb{Z} & & \end{aligned}$$

The first thing we do is solving the LP relaxation. As we saw in last chapter, this might give us the solution, and if it does not, it will at least give us a bound on the solution. The solution to the LP relaxation can be found using Simplex. We will not explicitly use Simplex in this section, but we just give the solution

$$\boldsymbol{x}_{\text{LP}} = (\frac{20}{7}, 3), \qquad z^*_{LP} = -\frac{59}{7}.$$

which is also shown in Figure 13.1.

Note that the solution to the LP relaxation is not integral. This means we are not done yet; we have not found a solution to the ILP yet. What we did find is a lower bound for the ILP. Indeed, the solution to the LP relaxation must be at least as good as the solution to the ILP. Note that in this instance, the coefficients of the objective function are all integral, so we can sharpen the lower bound to $\underline{z} = \lceil -\frac{59}{7} \rceil = -8$ (Lemma 12.1).

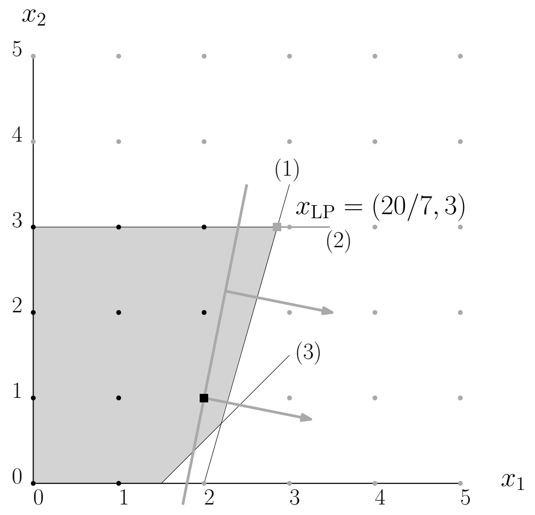


Figure 13.1: The feasible region for the LP relaxation. The grey area is the feasible polytope for the LP relaxation, the black dots and square are the feasible points for the ILP. The restrictions are numbered as in the ILP formulation. The grey line represents the objective function, and the square dots represent the optimal solution for the LP relaxation (grey) and the ILP (black).

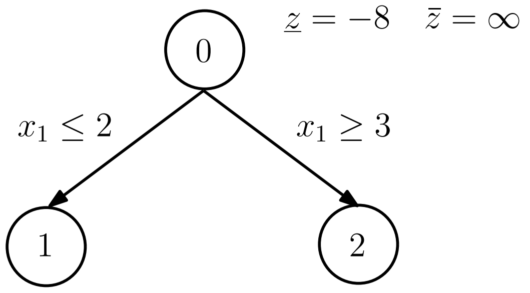


Figure 13.2: The tree representing the Branch and Bound algorithm just after the first branching step

13.1.1 Branching

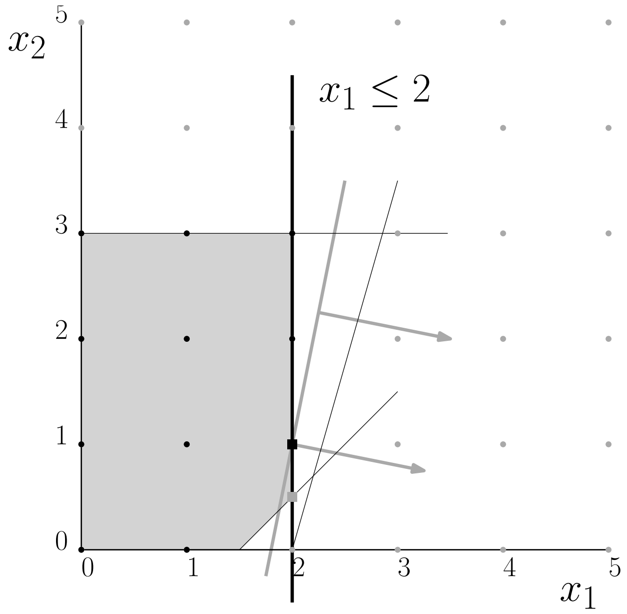
At this point, we cannot easily get any more information from the LP relaxation. Therefore, it is time to branch out into subproblems. To decide what kind of subproblems are useful, we again look at the solution to the LP relaxation.

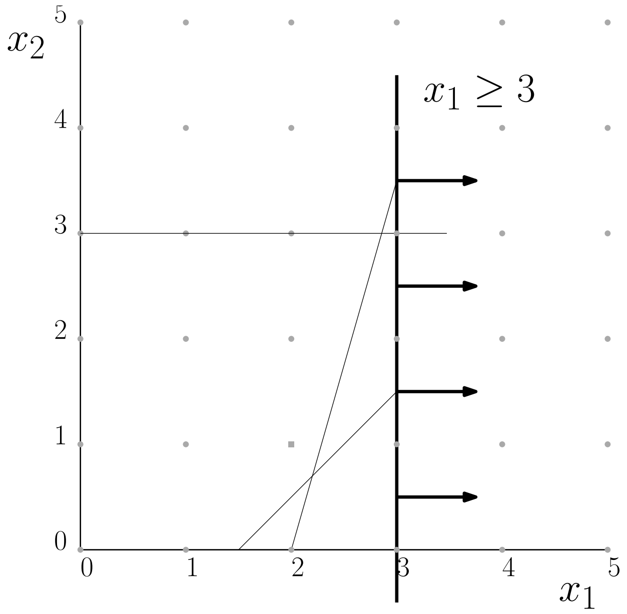
Note that in the solution of the LP relaxation, the first variable x_1 has value 20/7. Such a fractional value is not allowed in a solution to the ILP. Hence we try to exclude this value by choosing subproblems where this value is not feasible. One way to do this is to split the feasible region in two parts: one where $x_1 \leq 2$ and one where $x_1 \geq 3$. It is clear that these two regions together still contain all integral points, so no feasible solution for the ILP is lost when we split into subproblems.

Splitting up the region into two is equivalent to considering two subproblems. In the first subproblem we add the restriction $x_1 \leq 2$ to the original ILP, and in the second we add the restriction $x_1 \geq 3$ to the original ILP. Because each feasible integer point is in the feasible region of one of these problems, we can find the optimal solution to the original ILP by solving both subproblems, and taking the best optimal solution of the two.

In this case we get the following two subproblems:

$$
\begin{array}{rrrl}
z^1 = \min & -4\,x_1 + & x_2 & \\
s.t. & 7\,x_1 & -2\,x_2 & \leq 14 \\
 & & x_2 & \leq 3 \\
 & 2\,x_1 & -2\,x_2 & \leq 3 \\
 & x_1 & & \leq 2 \\
 & & x_1, x_2 & \geq 0 \text{ integral}
\end{array}
\qquad
\begin{array}{rrrl}
z^2 = \min & -4\,x_1 + & x_2 & \\
s.t. & 7\,x_1 & -2\,x_2 & \leq 14 \\
 & & x_2 & \leq 3 \\
 & 2\,x_1 & -2\,x_2 & \leq 3 \\
 & x_1 & & \geq 3 \\
 & & x_1, x_2 & \geq 0 \text{ integral}
\end{array}
\tag{13.1}
$$





We can see there are no feasible solutions to the second subproblem by looking at the graphical representation. This means that we cannot get any additional information from this subproblem, nor from any of its possible subproblems. We therefore do not have to branch on this problem anymore. In the tree representation of the algorithm this means that there are no branches coming from the node labeled 1. This is why we say "we prune (the tree at) this node".

13.1.2 Bounding and branching in subproblem 1

Now consider the first subproblem. We first solve the LP relaxation again giving the solution $\boldsymbol{x}^1_{LP} = (2, 0.5)$ and $z^1_{LP} = -7.5$.

For this subproblem, like for the original problem, the LP solution gives a lower bound for the optimal solution to the ILP subproblem. By using integrality of the parameters, we get a lower bound $\underline{z}^1 = -7$. Note that this bound only holds for the first subproblem; another branch might contain a better solution!

We do not get a better upper bound, because we have not found any integral solutions yet. This means that the previous upper bound still holds for this subproblem.

We have not found a reason to stop investigating this subproblem yet: we cannot prove there is no solution, nor have we found an optimal solution yet. Hence we need to branch on this subproblem again.

We again find a variable with fractional value in the optimal solution to the LP relaxation. In this case we find $x_2 = 0.5$, and we branch by creating two subproblems: one where we add restriction $x_2 \leq \lfloor 0.5 \rfloor = 0$ and one where we add $x_2 \geq \lceil 0.5 \rceil = 1$.

The resulting two problems (subproblems 3 and 4) are the following:

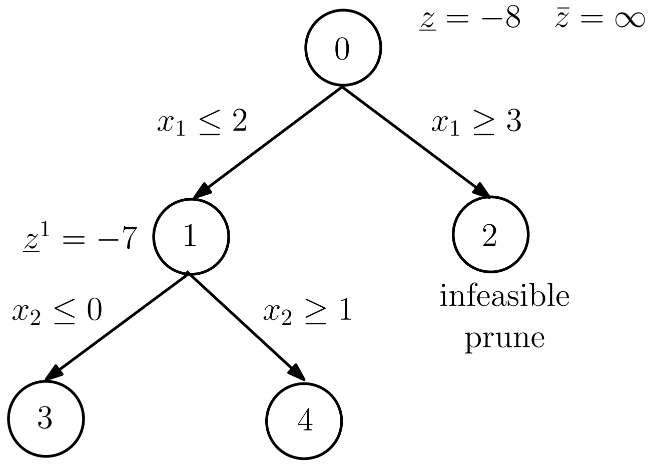
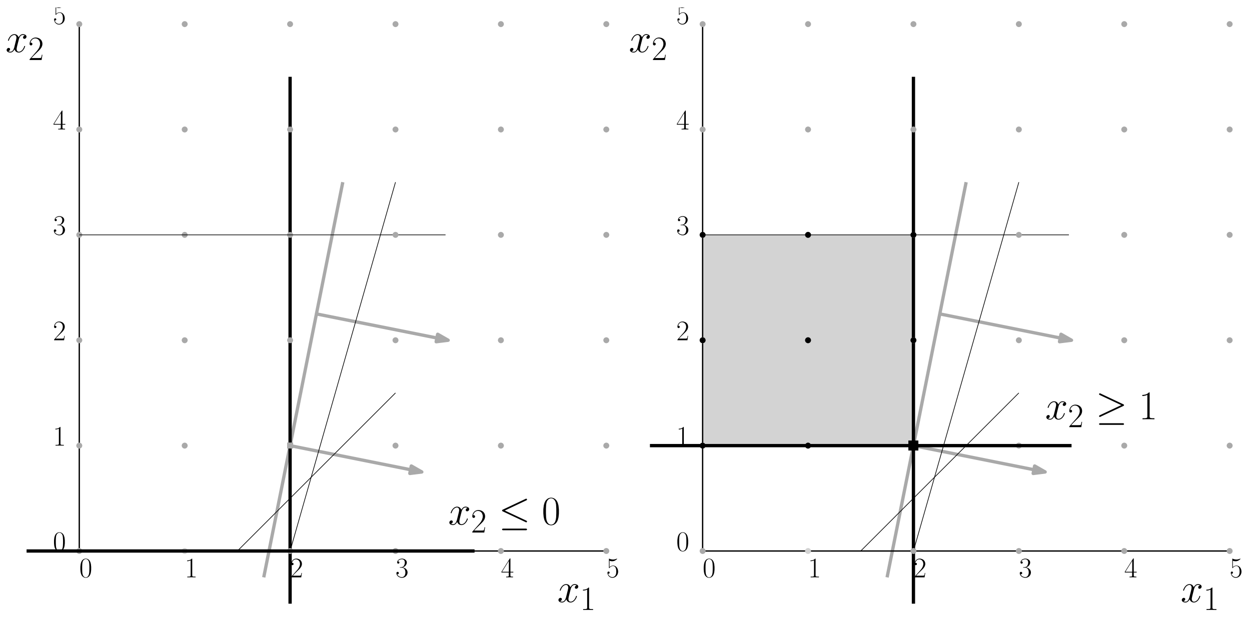


Figure 13.3: The tree representing the Branch and Bound algorithm directly after branching on node 1.

$$\begin{aligned} z^3 = \min\ & -4x_1 + x_2 \\ s.t.\ & 7x_1 - 2x_2 \leq 14 \\ & x_2 \leq 3 \\ & 2x_1 - 2x_2 \leq 3 \\ & x_1 \leq 2 \\ & x_2 \leq 0 \\ & x_1, x_2 \geq 0 \text{ integral} \end{aligned} \qquad \begin{aligned} z^4 = \min\ & -4x_1 + x_2 \\ s.t.\ & 7x_1 - 2x_2 \leq 14 \\ & x_2 \leq 3 \\ & 2x_1 - 2x_2 \leq 3 \\ & x_1 \geq 3 \\ & x_2 \geq 1 \\ & x_1, x_2 \geq 0 \text{ integral} \end{aligned} \tag{13.2}$$



13.1.3 Bounding and pruning in subproblem 4

We now decide to look at subproblem 4. We solve the LP relaxation giving solution $\boldsymbol{x}_{LP}^4 = (2, 1)$ and $z_{LP}^4 = -7$. By similar arguments to before, we get a lower bound of $\underline{z}^4 = -7$ for this node.

Now note that the solution to the LP relaxation is integral! This means that we have found a feasible solution. A feasible solution to a subproblem is automatically a feasible solution to the original problem. Therefore the optimal solution to the original problem needs to be at least as good as this solution. In other words, the integral solution we found gives an upper bound to the original problem. We update the upper bound for the original problem to $\overline{z} = -7$ and we update the graphical representation of the Branch and Bound procedure up to this point as seen in Figure 13.4.

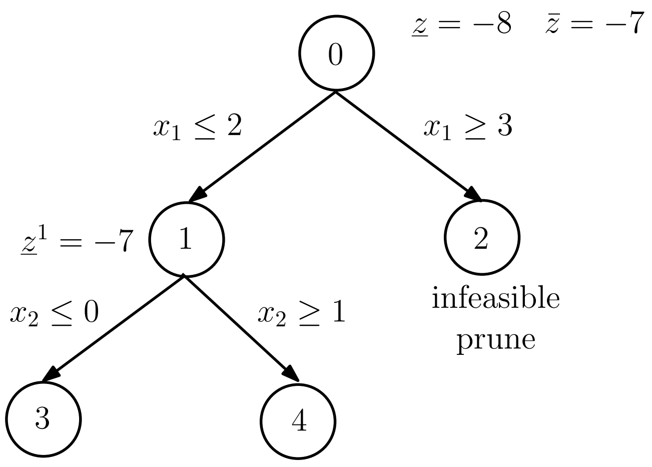


Figure 13.4: The tree representing the Branch and Bound algorithm directly after bounding for node 4.

The next step is to see at which nodes we can prune, and at which node we continue. Node 2 has already been pruned, so we do not have to continue in that node. For node 4 we just found an optimal integral solution. This means we cannot find a better integral solution by branching on this node, so we can prune node 4 by optimality.

The last leaf of the current tree is node 3. We can solve the LP relaxation for subproblem 3 and find a lower bound. However, we know that this lower bound cannot be lower than the lower bound for its parents problem, problem 1. This means we can directly conclude that the lower bound of node 4 is at least -7. Because we have already found a solution with value -7 we do not have to investigate this subproblem any further, and we can prune this node by bound. Another way to see this is by looking at the bounds: the worst possible solution to the original problem is at least as good as the best solution for subproblem 3, i.e. $\overline{z} \leq \underline{z}^3$.

Because each leaf of the tree has been pruned, we cannot branch any further and the Branch and Bound procedure ends. This means that one optimal solution is the integral solution we found for subproblem 4: $\boldsymbol{x}_{\text{IP}} = (2, 1)$ and $z^* = -7$.

13.2 Branch and Bound for ILP

The following pseudocode is a general description of solving ILPs using branch and bound. In the rest of this section we will look at each of the steps in detail. We will spend some extra time on lines 5, 8, 15, 17 and 19, where there are actually multiple options, or there is a reason to prune a branch.

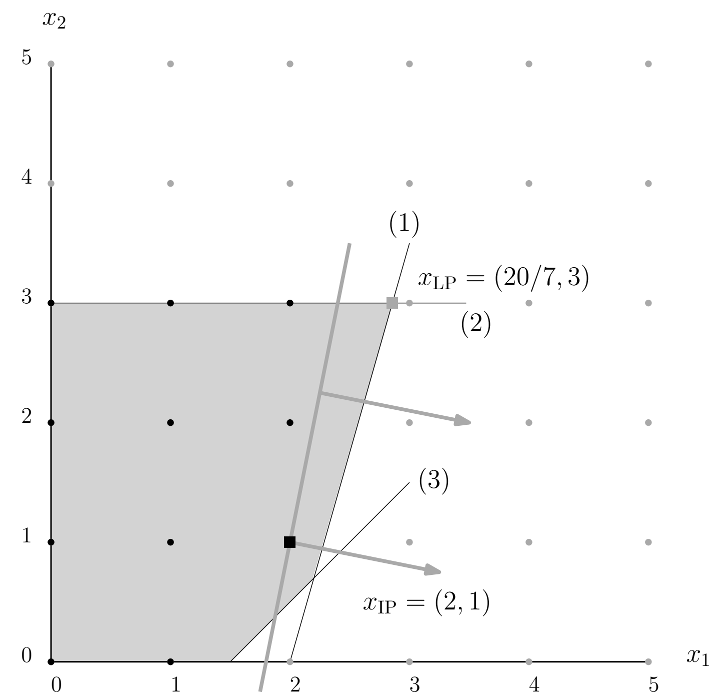


Figure 13.5: The optimal solution of the ILP problem.

```
Data: An ILP instance $\mathcal{I}_0$: $\min z = \boldsymbol{c}^\mathsf{T}\boldsymbol{x}$ s.t. $A\boldsymbol{x} \le \boldsymbol{b}$, $\boldsymbol{x} \in \mathbb{Z}^n$ with $\boldsymbol{c}$ integral
Result: an optimal solution $\boldsymbol{x}^*$ or NONE if there is no optimal solution
Set the set of current problems to be $C = \{\mathcal{I}_0\}$;
Set the current upper bound $\overline{z} = \infty$;
Set the current solution to $\boldsymbol{x}^* =$ NONE;
while $C \neq \emptyset$ do
    Choose current problem $\mathcal{I}_i \in C$;
    Solve the LP-relaxation $\mathcal{I}_i^{LP}$ of $\mathcal{I}_i$;
    if $\mathcal{I}_i^{LP}$ has no feasible solution then
        Prune by infeasibility;
    else
        $\mathcal{I}_i^{LP}$ has a feasible solution $\boldsymbol{x}_i^*$ with value $z_i^*$;
        Set the lower bound $\underline{z}_i := \lceil z_i^* \rceil$ for the subproblem;
        if the optimal solution $\boldsymbol{x}_i^*$ is integral and $z_i^* < \overline{z}$ then
            Set the upper bound $\overline{z} := z_i^*$;
            Set the current best solution $\boldsymbol{x}^* := \boldsymbol{x}_i^*$;
            Prune by optimality;
        else if $\underline{z}_i \ge \overline{z}$ then
            Prune by bound;
        else
            Branch the current problem adding two subproblems to $C$;
        end
    end
    Remove $\mathcal{I}_i$ from $C$;
end
return $\boldsymbol{x}^*$
```

Algorithm 6: Branch and Bound for ILP

Lines 1,…,3 are the initialization of the algorithm. In these steps we define the variables we use and we set them to their initial values.

Then in lines 4,…,22 the actual algorithm is described: The while loop says we have to keep solving subproblems until there are no subproblems left ($C = \emptyset$). The first step in the loop is to choose a subproblem to solve. We could pick an arbitrary problem from C, but there are smarter ways to choose a problem, which we discuss in Section 13.2.2.

After solving the ILP relaxation of the chosen problem, the next step depends on the solution. If there is no feasible solution, then we have solved the subproblem, because there can certainly be no integral solution. This means we can prune at this node because of infeasibility. If the solution is integral, then we have completely solved the subproblem, too. Because we have found an optimal solution, we say we prune by optimality. Another possibility is that we know the best solution to this subproblem cannot be better than another solution we already found. We can know this if $\underline{z}_i > \overline{z}$, i.e. the best possible solution to the subproblem ($\underline{z}_i$) is worse than the worst possible solution to the whole problem ($\overline{z}$). Because we recognize this situation by looking at the bounds, we say we can prune by bound. If $\underline{z}_i = \overline{z}$, we can also prune by bound because we are looking for just one optimal solution, and in this case the subproblem can at best have a solution that is just as good as the best solution we already have.

If none of the conditions above hold, we cannot prune the node and we have to branch. Branching means that we split the problem in subproblems. We add these new subproblems to the set C of problems we still have to investigate. There may be multiple ways to branch, which we see in Section 13.2.3

The last line of the algorithm is the result: if there was an integral solution, $\boldsymbol{x}^*$ contains an optimal solution to the ILP. If there was no integral solution, then all problems will in the end be pruned by infeasibility, and $\boldsymbol{x}^*$ still has value NONE.

Finally, we note that the current description of the algorithm is restricted to instances where the cost vector $\boldsymbol{c}$ is integral. However, the algorithm can easily be adapted to non-integral cost vectors. The only difference is that in line 11 we set $\underline{z}_i := z_i^*$ without rounding up, because when $\boldsymbol{c}$ is not integral also the optimal solution value z^* does not need to be integral.

13.2.1 The Branch and Bound tree

The structure of the Branch and Bound algorithm naturally fits in a tree. We start with one node v_0 representing the ILP $\mathcal{I}_0$. This node is called the root of the tree. We can then branch at this node, producing two subproblems that we can represent by two new nodes, connected to the root. Now every time we branch at a problem $\mathcal{I}_i$, we add two nodes v_k and v_{k+1} and connect them via an edge to the node v_i. Doing this, we produce a tree. For convenience, we say that problem $\mathcal{I}_i$ is the parent problem of problems k and $k+1$, or that v_i is the parent node of nodes v_k and of v_{k+1}. Similarly, we say that k and $k+1$ are i's children.

We have already seen an example of a branch and bound tree in the example of this chapter. Figure 13.4 shows the corresponding Branch and Bound tree. The original problem (problem $\mathcal{I}_0$ or just problem 0) is shown at the root (top) of the tree, together with the lower bound found by solving the LP relaxation LP_0. The children of 0 are 1 and 2, and node 1 has two children: nodes 3 and 4.

The figure contains some more information about the branch and bound procedure. The label of each arc (i, j) shows the extra restriction of problem j compared to problem i. Furthermore, the label at each node i gives the lower bound $\underline{z}_i$.

13.2.2 Choosing a new subproblem

Algorithm 6 still has two steps (lines 5 and 19) where we have not in detail decided what we must do. The first of these is choosing a new subproblem. Although the algorithm works for any choice of subproblem, there are a few standard ways of picking the new problem.

The first of these methods is *depth first search*. This means one chooses to go down the search tree as fast as possible to quickly find an integral solution. If a branch you are investigating is pruned, and there are still branches left, go up to the first non-pruned node, and go down from there again. The hope with this method is to quickly find a feasible solution of good quality.

A second method is *best node first*. For this method we continue with the node that has the best solution to its LP relaxation. This seems like a good way to quickly find a very good solution, but it might take quite some time to find an upper bound $\overline{z}$ unlike for the previous method. Note that for this method we need to know the solutions to the LP relaxations of each problem. However, in the algorithm we have not calculated all of these yet, so choosing the best subproblem also entails computing the LP relaxation solutions for new nodes.

The last method we discuss is a combination of the above: use depth first search until we have one integral solution, and hence also an upper bound. After this, use best node first search. This can be a relatively quick way of going through the tree, but is not guaranteed to be faster than any of the other two.

13.2.3 Branching

Branching means to split up a problem into subproblems. For ILPs this means we split up the feasible region into multiple parts. Each of these parts together with the original objective function defines a subproblem. The solution to the original problem is the best of all optimal solutions to the subproblems.

We need to have some kind of guarantee that the optimal solution to the original problem is still present in one of these subproblems. One way to do this is by making sure that each feasible point of the original problem is feasible in at least one of the subproblems. For ILPs this means we can define subproblems by taking subsets of the feasible region so that each integral point is in at least one of the feasible regions of the subproblems.

In the example, we used one particular way to do this. We pick one variable x_i and a value c and introduce two new restrictions (one for each of the two subproblems): $x_i \leq c$ and $x_i > c$. We then note that for a solution to one of these problems to be integral, we also need x_i to be integer. Hence we can just as well round c to integral values and use the restrictions $x_i \leq \lfloor c \rfloor$ and $x_i > \lceil c \rceil$. This cannot cut away any integral solutions.

The variable x_i we pick to branch on is one that has fractional value in the optimal solution to the LP relaxation. This is so we can be sure to cut away the current optimal but not integral solution. To do this, we also have to pick the value c smartly. To cut away the optimal solution, we can choose c to be the value of x_i in the optimal solution to the LP relaxation.

In the example there was always one fractional variable in the solution of the LP relaxation. In general this is not the case. When there are multiple such variables, we have to choose on which to branch. A general rule is to choose the variable with fractional value closest to 1/2.

13.3 Exercises

Exercise 13.1. Solve the following problem using Branch and Bound:

$$\begin{aligned} \max\ x_1 + 5x_2 & & \\ \text{s.t.} - 4x_1 + 3x_2 &\leq & 6 \\ 3x_1 + 2x_2 &\leq & 18 \\ x_1,\ x_2 &\geq & 0,\ \text{integral} \end{aligned}$$

Use the following search strategy:

- depth-first,
- $\leq$-branch first,
- fractional part closest to $\frac{1}{2}$.

Draw the feasible region and the added restrictions. A number of solutions to LP relaxations are given below.

Solution to the original LP-relaxation (LP_0):
$(x_1, x_2)^T = (2.47, 5.29)^T,\ z_{LP_0} = 28.9$
LP_0 with added restriction $x_1 \leq 2$:
$(x_1, x_2)^T = (2, 4.67)^T,\ z_{LP} = 25.3$
LP_0 with added restriction $x_1 \geq 3$:
$(x_1, x_2)^T = (3, 4.5)^T,\ z_{LP} = 25.5$
LP_0 with added restrictions $x_1 \leq 2$ and $x_2 \leq 4$:
$(x_1, x_2)^T = (2, 4)^T,\ z_{LP} = 22$
LP_0 with added restrictions $x_1 \geq 3$ and $x_2 \leq 4$:
$(x_1, x_2)^T = (3.33, 4)^T,\ z_{LP} = 23.33$
LP_0 with added restrictions $x_1 \geq 3$ and $x_2 \leq 4$ and $x_1 \leq 3$:
$(x_1, x_2)^T = (3, 4)^T,\ z_{LP} = 23$

Exercise 13.2. Solve the following problem using Branch and Bound:

$$\begin{aligned}
\min\ & 2x_1 - 3x_2 - 4x_3 \\
\text{s.t.} -x_1 + x_2 + 3x_3 &\leq 8 \qquad (13.3)\\
3x_1 + 2x_2 - x_3 &\leq 10 \qquad (13.4)\\
x_1,\ x_2,\ x_3 &\geq 0, \text{ integral}
\end{aligned}$$

The optimal tableau of the LP relaxation is as follows (s_1, s_2 are the slack variables corresponding to restrictions (13.3) and (13.4)):

basis	$\bar{b}$	x_1	x_2	x_3	s_1	s_2
$-z$	$\frac{138}{7}$	$\frac{18}{7}$	0	0	$\frac{11}{7}$	$\frac{5}{7}$
x_3	$\frac{6}{7}$	$-\frac{5}{7}$	0	1	$\frac{2}{7}$	$-\frac{1}{7}$
x_2	$\frac{38}{7}$	$\frac{8}{7}$	1	0	$\frac{1}{7}$	$\frac{3}{7}$

Use the following search strategy:

- depth-first,
- $\geq$-branch first,
- fractional part closest to $\frac{1}{2}$.

Use the dual simplex method to solve the subproblems. You may use that the original problem with added restriction $x_2 \geq 6$ is infeasible.

Exercise 13.3. Solve the following problem using Branch and Bound:

$$\begin{array}{lrcrcl}
\max z = & 13x_1 & + & 8x_2 & & \\
\text{s.t.} & x_1 & + & 2x_2 & \leq & 10 \\
 & 5x_1 & + & 2x_2 & \leq & 20 \\
 & x_1, & & x_2 & \geq & 0, \text{ integral}
\end{array}$$

The LP-relaxations may be solved graphically, or using an optimization package such as Qsopt.

Use the following search strategy:

- depth-first,
- $\leq$-branch first,
- fractional part closest to $\frac{1}{2}$.

Exercise 13.4. Consider the following problem:

$$\max\left\{ x_1 \;\middle|\; \begin{array}{c} 2x_1 + 2x_2 + \cdots + 2x_n = n \\ x_j \in \{0,1\},\ 1 \leq j \leq n \end{array} \right\}$$

with n odd, making the problem infeasible.

(a) Let J be a subset of $\{1, 2, \ldots, n\}$ with at most $\frac{n-1}{2}$ elements.

Assign a value 0 or 1 to each x_j, $j \in J$ and prove that the LP relaxation is feasible (after fixing the values of the variables in J). In other words, prove that the problem

$$\max\left\{ x_1 \;\middle|\; \begin{array}{l} \sum_{j \notin J} 2x_j = n - \sum_{j \in J} 2x_j \\ 0 \leq x_j \leq 1,\ j \notin J \end{array} \right\}$$

is always feasible.

(b) Determine a lower bound on the number of subproblems a branch and bound procedure has to go through to solve the original problem.

Hint: Using Branch and Bound on a $0-1$-optimization problem creates subproblems by fixing the variables to either 0 or 1. Complete enumeration then results in a tree of depth n with $2^{n-1}-1$ nodes.

Exercise 13.5. Algorithm 6 works for minimization problems only. Which changes do we need to make so it works for maximization problems?

In Section 13.1.3, instead of solving the LP-relaxation of subproblem 3, we used the upper bound of its parent problem, subproblem 3.

Modify Algorithm 6 so it also uses this 'shortcut' of using lower bounds of parent problems.

Exercise 13.6. Evaluate the proposed proof of the following result:

Result: Let P be a polytope such that $P \cap \mathbb{Z}^n = \emptyset$. Suppose we use Branch and Bound on an ILP with feasible region P, then the algorithm finishes after a finite number of steps.

Proof: Because the ILP is infeasible, each node of the branch and bound tree is pruned by infeasibility. Each of these pruning steps corresponds to cutting off a half plane from P, because it is equivalent to excluding the subproblem corresponding to the other side of the half plane. The part we cut off has non-zero volume and P has a finite volume, so after a finite number of pruning steps, we have cut off the whole of P. Hence the branch and bound tree has a finite number of leafs, and therefore the branch and bound algorithm only has a finite number of steps.

Chapter 14

Cutting Planes

The feasible region of an ILP consists of the integer points in a given polytope. To solve the ILP, we would ideally have a quick way to make sure all vertices of the polytope are integer. Unfortunately, no such method exists: it is hard to even decide whether a polytope contains an integer point.

This means that an ILP is generally defined in a 'stupid' way, with a complex polyhedron whose integer points might be described more easily. A way to cope with this, is by cutting off parts of the polytope that do not contain integer points. Methods that use this approach are called *cutting plane* methods, because they cut the polytope with a plane (affine subspace).

In this chapter we study ways of producing cutting planes for a given polytope, and we see how they can be used to solve ILP instances.

14.1 Improving the LP-relaxation

Suppose we are given an ILP with feasible region defined by the polytope as seen in Figure 14.1. In this case, we can find the integer points by inspection. These points can easily be described with three linear inequalities, defining the triangle with all points at the faces of the triangle.

Finding the integral points is impossible for a general case. Even in this example it is hard to see if the horizontal side of the triangle needs to be extended to the right one more.

Another problem with this polytope is that it is much bigger than the triangle, which could also describe the integral points of the ILP. Note that the polytope is 'bigger' in two ways: it has a lot

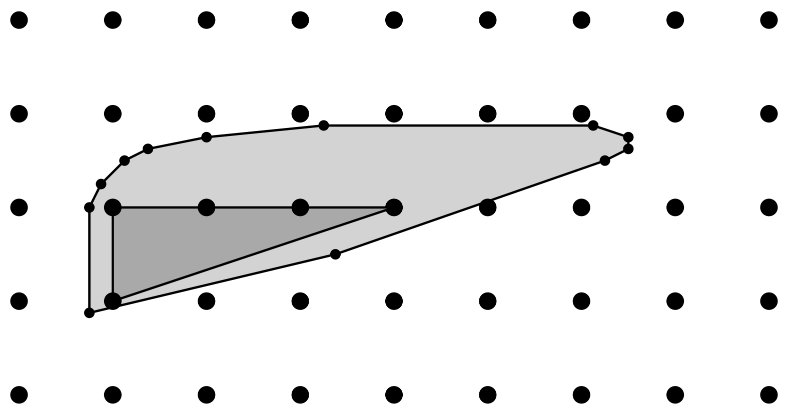


Figure 14.1: A polytope P in light grey together with the strongest possible formulation of $P \cap \mathbb{Z}^2$ within it (dark grey).

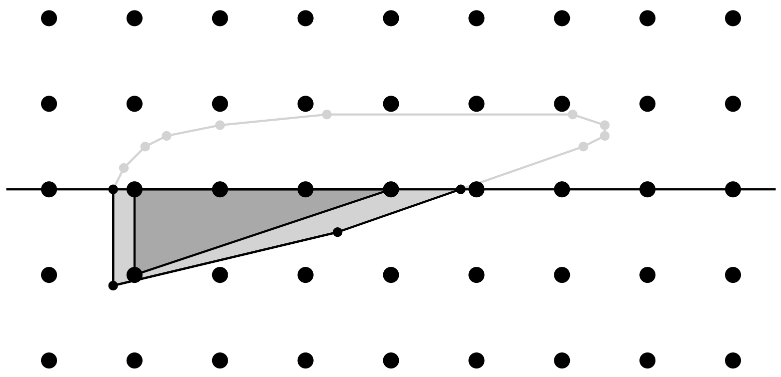


Figure 14.2: The polytope of Figure 14.1 with an extra cut.

more faces (12 vs. 3) and it covers a lot more area with several vertices being far from any vertex of the triangle.

These problems can be solved (at least partly) by cutting off parts of the polytope that do not contain integral points. An example of such a cut would be adding the inequality corresponding to the horizontal face of the triangle.

Now suppose we try to solve the ILP with this polytope and objective function max $z = x$ (with x the variable corresponding to the horizontal axis). The LP-relaxation finds one of the points on the right-hand side of the polytope, quite far from the actual optimum. One way to improve this approximate solution is to add an inequality to the formulation, for example the one we just mentioned, cutting off along the horizontal face of the triangle. Solving the LP-relaxation corresponding to the new ILP formulation gives a significantly better approximation to the optimal ILP solution. In the next sections, we define concepts so that we can more precisely talk about different formulations of an ILP, and we will see how to find cutting planes we can add to the formulation.

14.2 ILP formulations

In the example above, we saw that there are multiple formulations of an ILP: we saw three different polytopes defining the same set of integral points. We say that a polytope P is a *formulation* for a set $S \subset \mathbb{Z}^n$ if $P \cap \mathbb{Z}^n = S$. In words this says that the integral points in P must be exactly the set S.

As in the previous example, there can be multiple formulations for a given set of points. In fact, if there exists one formulation, there exist infinitely many (Exercise 14.6). A simple example can be found if we consider the set $S = \{\mathbf{0}\} \subset \mathbb{R}^n$: any square containing $\mathbf{0}$ contained in $(-1, 1)^n$ is a formulation for S. By varying the position and the size of the squares (both continuous parameters), we can find infinitely many such squares (Figure 14.3).

We can also formalize how good a formulation is. In the example, the initial polytope was unnecessarily big. A way to get a better formulation was to cut off parts of the polytope. This corresponds to the following definition: a formulation P is *stronger* than a formulation Q if $P \subset Q$.

When cutting off parts of the polytope, we need to make sure we do not cut off any integral points. This means that the new inequality must hold for any integral point in the polytope. We say an inequality $\boldsymbol{a}^\top \boldsymbol{x} \leq b$ is *valid* for P if for any point $\boldsymbol{s} \in S = P \cap \mathbb{Z}^n$ the inequality $\boldsymbol{a}^\top \boldsymbol{s} \leq b$ holds.

Now you might wonder whether there is a strongest formulation, and what it looks like. The answer is that such a strongest formulation exists in the form of the *convex hull* (see also Chapter 3, Definition 3.7) of the integral points of S. The convex hull of a finite set of points $S = \{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k\} \subset \mathbb{R}^n$ is

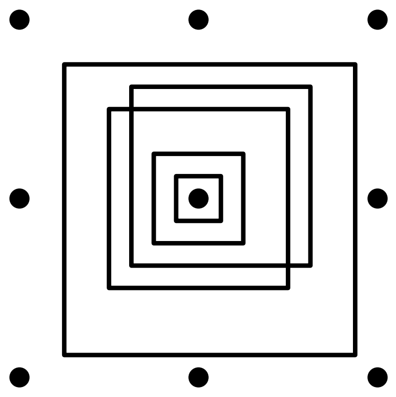


Figure 14.3: Multiple formulations (squares) of a single integer point.

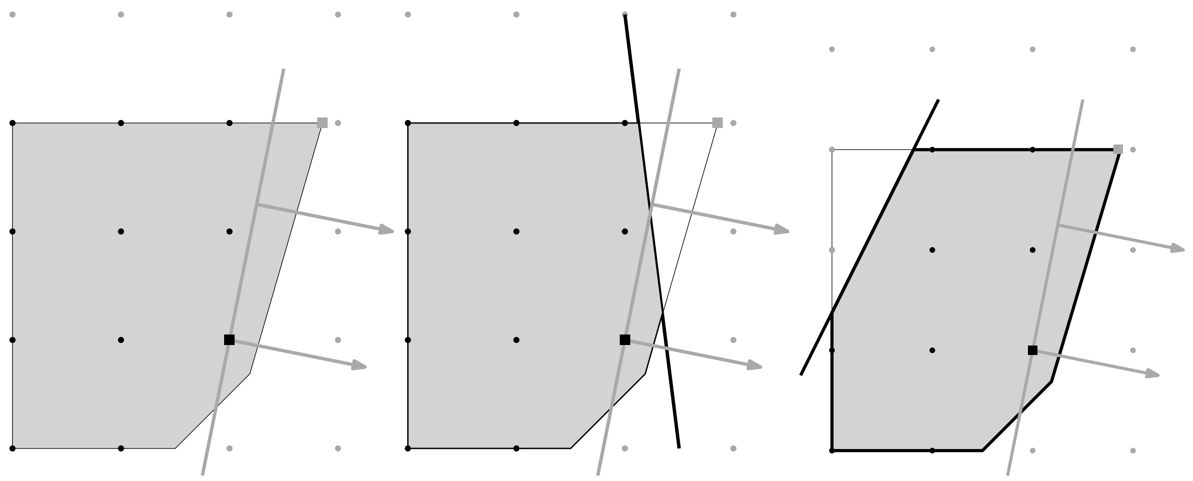


Figure 14.4: A formulation for a set of integer points and two different cuts. The left figure gives the situation for an ILP instance. The cut in the middle figure is valid. The cut in the figure on the right is not valid, even though it does not cut away the optimal point for the ILP.

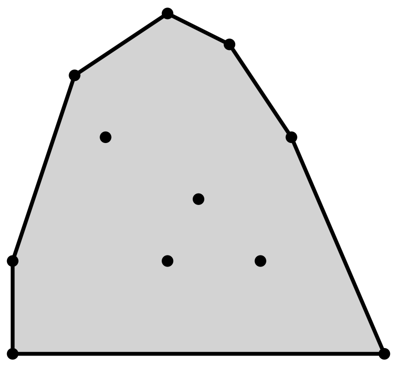


Figure 14.5: A set of points, with their convex hull.

defined as

$$\operatorname{conv}(S) := \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} = \sum_{i=1}^{k} \lambda_i \boldsymbol{s}_i, \sum_{i=1}^{k} \lambda_i = 1\}.$$

This definition might seem complex, but the idea is simple: the convex hull of a set S is the set of all points enclosed by S. The dark grey triangle in Figures 14.1 and 14.2 above is an example of a convex hull, and of a strongest formulation for some set S. Another example can be found in Figure 14.5

14.3 Gomory cutting planes

In this section we study a specific type of cutting planes: Gomory cutting planes (GCPs). These cutting planes can easily be generated from an optimal simplex tableau, which is why they are so useful.

14.3.1 Example

Let us first consider an example: take the ILP we studied in Section 13.1.

$$\begin{aligned} z_{IP} = \min\ & -4\,x_1 + \ x_2 \\ s.t.\quad & 7\,x_1 - 2\,x_2 \leq 14 \qquad (1) \\ & \qquad\qquad x_2 \leq 3 \qquad (2) \\ & 2\,x_1 - 2\,x_2 \leq 3 \qquad (3) \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z} \end{aligned}$$

In that section we found the optimal solution to this problem using Branch and Bound. Here we take a different approach: after calculating the optimal solution to the LP-relaxation, we add cutting planes to improve the bound given by the LP-relaxation. For this we use the optimal Simplex tableau:

basis	$\bar{b}$	x_1	x_2	s_1	s_2	s_3
x_1	20/7	1	0	1/7	2/7	0
x_2	3	0	1	0	1	0
s_3	23/7	0	0	-2/7	10/7	1
$-z$	59/7	0	0	4/7	1/7	0

Let us now inspect the first row of this Simplex tableau.

$$x_1 + \frac{1}{7}s_1 + \frac{2}{7}s_2 = 2\frac{6}{7}$$

Note that this equality has some fractional, and some integral parts. We can split these parts, and then put all integral parts on one side of the inequality, and the fractional parts on the other side.

$$\begin{aligned} 1x_1 + (0 + \frac{1}{7})s_1 + (0 + \frac{2}{7})s_2 &= 2 + \frac{6}{7} \\ x_1 - 2 &= \frac{6}{7} - \frac{1}{7}s_1 - \frac{2}{7}s_2 \end{aligned}$$

Because the left-hand side is integral, the right-hand side must also be integral. Furthermore, we know that s_1 and s_2 are nonnegative. This means the only integral values that $\frac{6}{7} - \frac{1}{7}s_1 - \frac{2}{7}s_2$ can take are $0, -1, -2, \ldots$. We can summarize this by saying this expression is smaller or equal to 0, giving the new inequality

$$\begin{aligned} \frac{6}{7} - \frac{1}{7}s_1 - \frac{2}{7}s_2 &\leq 0 \\ \frac{6}{7} &\leq \frac{1}{7}s_1 + \frac{2}{7}s_2 \\ 6 &\leq s_1 + 2s_2. \end{aligned}$$

This inequality is the Gomory cut corresponding to the first row of the Simplex tableau. It is more insightful to rewrite this inequality in the original variables x_1 and x_2 so we can draw the cut in the feasible region. Using the standardized version of the original inequalities

$$\begin{aligned} 7x_1 - 2x_2 + s_1 &= 14 \\ x_2 + s_2 &= 3, \end{aligned}$$

we can reduce the new inequality to

$$\begin{aligned} 6 &\leq s_1 + 2s_2 \\ 6 &\leq (14 - 7x_1 + 2x_2) + 2(3 - x_2) \\ -14 &\leq -7x_1 + 2x_2 - 2x_2 \\ -14 &\leq -7x_1 \\ x_1 &\leq 2. \end{aligned}$$

This gives us a new ILP with the same optimal solution as the original ILP (Figure 14.6). We could solve this ILP by starting over completely, but there is a smarter solution: we add the new inequality to the simplex tableau and solve the new LP-relaxation using dual simplex after converting it to standard form. After all, the old solution is not primal feasible anymore, but it is feasible in the dual (convince yourself by looking what happens in the simplex tableau).

We can keep repeating this procedure. Each cut will either improve the current formulation or it remains the same. At some point we might find an optimal solution to the LP that is integral, hence giving an optimal ILP solution. However, this is not always the case in general. Therefore, cutting planes are used in practice in combination with Branch and Bound to solve ILPs.

14.3.2 A general method for producing GCPs

The method we used in the previous section is quite general. In fact, for each of the rows of the optimal simplex tableau, there is a Gomory cut. Because the objective row also represents a valid equality, there is even a Gomory cut corresponding to the objective row!

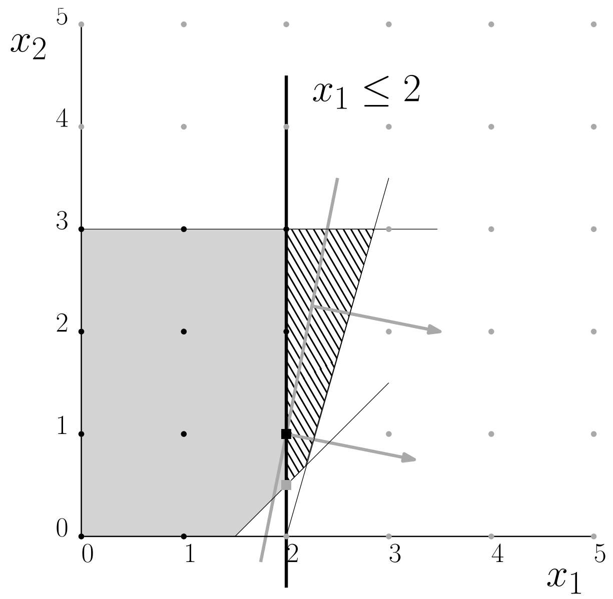


Figure 14.6: The new problem with the added Gomory cut $x_1 \leq 2$.

The procedure is as follows: take any row of the simplex tableau; this gives an equality of the form:

$$\lambda_1 a_1 + \cdots + \lambda_k a_k = b,$$

where the λ_i and b are constants, and the a_i are the variables of the problem (including slack and surplus variables).

The next step is to separate the fractional and the integral parts. The *integral part* of a number $x \in \mathbb{R}$ is defined as $\lfloor x \rfloor$ and the *fractional* part of $x \in \mathbb{R}$ is defined as $x - \lfloor x \rfloor$. Hence, the fractional parts are in the interval $[0, 1)$. This is also the case when x is negative. For example, the integral part of $-4/3$ is -2 and its fractional part is $2/3$.

So we split each λ_i into the integral part λ_i^{int}, and the fractional part λ_i^{frac}. Similarly, we split b into b^{int} and b^{frac}. Then we take all the integral parts to one side of the equality, and the fractional parts to the other side, giving

$$\lambda_1^{\text{int}} a_1 + \ldots + \lambda_k^{\text{int}} a_k - b^{\text{int}} = -\lambda_1^{\text{frac}} a_1 - \ldots - \lambda_k^{\text{frac}} a_k + b^{\text{frac}}.$$

Assuming the problem was in standard form, and the variables all non-negative, we see that the right-hand side is less than 1 for each feasible point. Indeed, each a_i is non-negative, each λ_i^{frac} is in $[0, 1)$, and also b^{frac} is in $[0, 1)$. Hence each summand $-\lambda_i^{\text{frac}} a_i$ is negative, and $b^{\text{frac}} < 1$. As the left-hand side is integral for each feasible point of the ILP, the right-hand side must also be integral. Since the right-hand-side is less than 1 and integral it must be smaller or equal to 0, giving the new inequality:

$$\lambda_1^{\text{frac}} a_1 + \cdots + \lambda_k^{\text{frac}} a_k \geq b^{\text{frac}}.$$

This inequality is a Gomory cutting plane for the chosen row of the simplex tableau. The cut can be added to the simplex tableau by adding a new slack variable for the new inequality to make it an equality, and then adding the produced equality, expressed in non-basis variables, as an extra row to the simplex tableau. This gives a tableau with a dual feasible solution, so we can continue solving the LP-relaxation with dual simplex.

14.4 Exercises

Exercise 14.1. Consider the following ILP instance:

$$\begin{aligned} \max\ & 4x_1 + 3x_2 \\ \text{s.t. } & 2x_1 + x_2 \le 11 \\ & -x_1 + 2x_2 \le 6 \\ & x_1,\ x_2 \ge 0,\ \text{integral} \end{aligned}$$

The optimal solution to the corresponding LP-relaxation is:

basis	$\bar{b}$	x_1	x_2	s_1	s_2
x_1	$\frac{16}{5}$	1	0	$\frac{2}{5}$	$-\frac{1}{5}$
x_2	$\frac{23}{5}$	0	1	$\frac{1}{5}$	$\frac{2}{5}$
$-z$	$-\frac{133}{5}$	0	0	$-\frac{11}{5}$	$-\frac{2}{5}$

a) Determine a Gomory cut for each of the rows of the simplex tableau, including the row of the objective function.

b) Write every Gomory cut in terms of x_1 and x_2, and draw the feasible region of the LP-relaxation together with the Gomory cuts.

c) Add the cut corresponding to the x_2 row to the Simplex tableau and use the dual Simplex algorithm. What is the new objective value?

Exercise 14.2. Consider the following ILP instance:

$$\begin{aligned} \max z_{IP} = \ & 2x_1 + x_2 + 4x_3 \\ \text{s.t.}\ & x_1 + x_2 + 2x_3 \le 10 \\ & -x_1 + 2x_2 - x_3 \ge 5 \\ & 4x_1 + 2x_2 + x_3 \le 10 \\ & x_1,\ x_2,\ x_3 \ge 0,\ \text{integral} \end{aligned}$$

The LP-relaxation of this problem was solved using the Simplex algorithm. The final tableau is the following:

basis	$\bar{b}$	x_1	x_2	x_3	s_1	e_2	s_3
s_1	$\frac{5}{4}$	$-\frac{19}{4}$	0	0	1	$-\frac{3}{4}$	$-\frac{5}{4}$
x_2	$\frac{15}{4}$	$\frac{3}{4}$	1	0	0	$-\frac{1}{4}$	$\frac{1}{4}$
x_3	$\frac{5}{2}$	$\frac{5}{2}$	0	1	0	$\frac{1}{2}$	$\frac{1}{2}$
$-z$	$-\frac{55}{4}$	$-\frac{35}{4}$	0	0	0	$-\frac{7}{4}$	$-\frac{9}{4}$

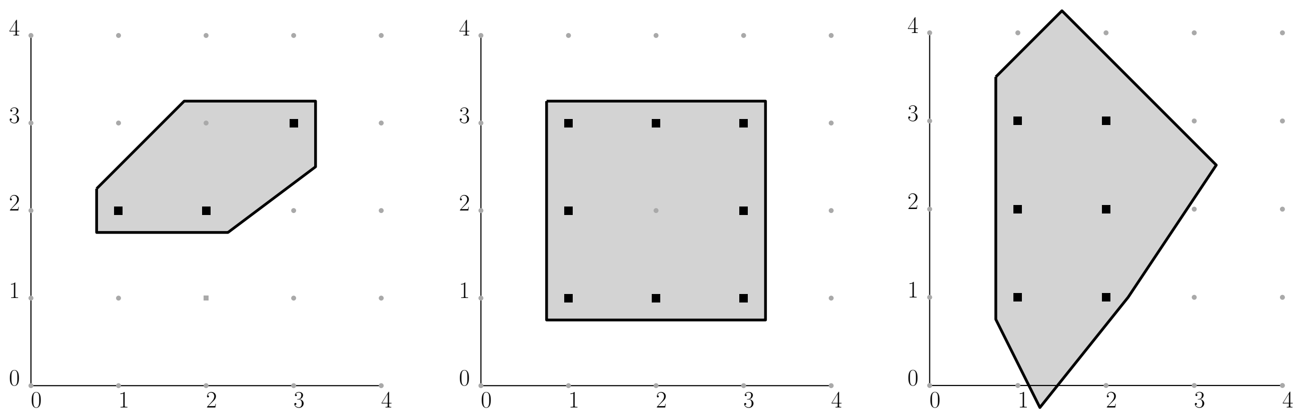
a) Determine a Gomory cut using the row corresponding to x_3 in the tableau.

b) Write this cut in terms of the original variables x_1, x_2 and x_3.

c) Add the Gomory cut to the optimal tableau and re-optimize using Dual Simplex. What is the optimal solution to the LP-relaxation after adding the cut?

d) Explain whether the solution you found in c) is also optimal for the original ILP.

Exercise 14.3. Given a the following pictures of P (grey polytope) and S (black square points), which P are formulations of the corresponding S?



Exercise 14.4. Prove or disprove: There exists a formulation for any finite set $S \subset \mathbb{Z}^n$.

Exercise 14.5. Prove that there are infinitely many formulations of

$$S = \{(0,0), (0,1), (1,0)\}$$

by giving an infinite set of polytopes P each of which is a formulation for S.

Exercise 14.6. Let $S \subset \mathbb{Z}^n$ be finite such that $ch(S) \cap \mathbb{Z}^n = S$, with $ch(S)$ the convex hull of S. Prove that there are infinitely many formulations of S.

Part V

Complexity Theory and Approximation Algorithms

Chapter 15

Analyzing Algorithms

In the preceding chapters, we have seen quite a lot of algorithms for a variety of problems. In theory, each of these algorithms produces an optimal solution, given a feasible input. However, we have not discussed how long it will take to find such an optimal solution. Of course, this depends on the instance and in particular on the size of the instance. For huge instances, the computation time is likely to be much longer than for small instances. However, how quickly does the computation time increase when the instance size increases? This greatly varies between algorithms. In turns out that for some algorithms the computation time scales only linearly in the instance size, while for other algorithms the computation time grows exponentially, e.g., when you allow one extra vertex/variable in the input, the computation time may double! In this chapter, we explain how you can estimate the computation time of an algorithm as a function of the instance size.

15.1 Problems, instances and algorithms

Before we can formalize what we mean by computation time, we first need to explain the important distinction between problems and instances, and formalize what we mean by an algorithm.

Definition 15.1. A *problem* consists of precise descriptions of allowed input and desired output as function of the input.

For example, the following is a problem.

Shortest Path
Given: a directed graph $D = (V, A)$, vertices $s, t \in V$ and a length function $c : A \to \mathbb{R}_{\geq 0}$.
Find: a path P from s to t that minimizes

$$c(P) = \sum_{a \in P} c(a).$$

Shortest Path is the name of the problem. After **Given:** the allowed input is described and after **Find:** the desired output.

An *instance* of a problem is obtained by fixing a specific input. For example, an instance of Shortest Path is, for example, to find a shortest path from s to t in the directed graph in Figure 15.1 with the indicated arc-lengths.

Definition 15.2. An *algorithm* for a problem is a precise list on instructions that can be executed in finite time and finds desired output for each allowed input.

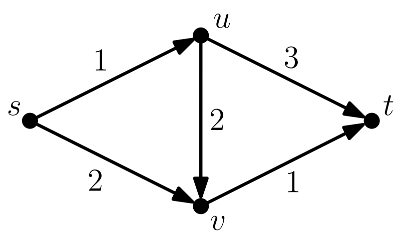


Figure 15.1: An instance of the SHORTEST PATH problem.

15.2 Big-Oh notation

The running time (computation time) of an algorithm does generally not only depend on the input size, but also on other factors such as how difficult the given instance is to solve. Therefore, it is generally impossible to give an exact description of the running time as a function of the input size n. Instead, we estimate the running time, using the following "big-Oh" notation.

Definition 15.3. Let f and g be functions from $\mathbb{N}$ to $\mathbb{R}^+$, then $f(n) = O(g(n))$ if there exist $c > 0$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

In words, this definition basically says that $f(n) = O(g(n))$ if, for big n, $f(n)$ grows at most as fast as $g(n)$, up to a constant factor.

For example, if $f(n) = 5n$ and $g(n) = n$ then we may write $f(n) = O(g(n))$. Hence, $5n = O(n)$. If $f(n) = O(n)$ then we say that f grows *linearly.*

If $f(n) = O(n^2)$ then we say that f grows *quadratically*, e.g. $5n^2 + 3n + 100$ grows quadratically.

If $f(n) = O(p(n))$ for some polynomial $p(n)$, then we say that f grows *polynomially.* For example, $10n^5 + 5n + 3$ grows polynomially, but also $5\sqrt{n}\log(n)$ grows polynomially.

If $f(n) = O(1)$ then we say that f is *constant*, e.g. $f(n) = 1000$ is constant.

While this big-Oh notation can be used to describe upper bounds, we can describe lower bounds using the following notation:

Definition 15.4. Let f and g be functions from $\mathbb{N}$ to $\mathbb{R}^+$, then $f(n) = \Omega(g(n))$ if there exist $c > 0$ and $n_0 \in \mathbb{N}$ such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$.

In words, $f(n) = \Omega(g(n))$ means that $f(n)$ grows at least as fast as $g(n)$ for big n, up to a constant factor.

For example, $3^n(8n^3 + 5) = \Omega(3^n)$. A function f is said to grow *exponentially* when $f(n) = \Omega(c^n)$ for some constant $c > 1$.

Finally, we write $f(n) = \Theta(g(n))$ when $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, i.e. when $f(n)$ and $g(n)$ grow just as fast for big n, up to a constant factor.

The following theorem can be useful when proving O, Ω and Θ relations. Note, however, that each of the three statements holds in only one direction; they are sufficient but not necessary conditions.

Theorem 15.1. The following three statements hold.

(i) If $\lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = 0$ then $f(n) = O(g(n))$.

(ii) If $\lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = c > 0$ then $f(n) = \Theta(g(n))$.

(iii) If $\lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = \infty$ then $f(n) = \Omega(g(n))$.

15.3 Input size

The running time of an algorithm is expressed as a function of the *input size*, which is the number of bits that is needed to store the input. However, how many bits you need depends on how you store the input. In this section, we will discuss what the input size is when the input consists of numbers, undirected graphs or directed graphs.

15.3.1 Representing numbers

Suppose the input contains a natural number n. How many bits does it take to store this number? This depends on the encoding. If we use a binary encoding, we need $\log_2(n)$ bits. If we use a decimal encoding, we need $\log_{10}(n)$ bits, which is slightly less. However, choosing decimal over binary encoding does not have a big impact on the input size. We have

$$\lim_{n\to\infty} \frac{\log_2(n)}{\log_{10}(n)} = \log_2(10),$$

which is a constant. Therefore, by Theorem 15.1, $\log_2(n) = O(\log_{10}(n))$.

Similarly, we find that all k-ary encodings are equivalent under the Big-Oh notation, for $k \geq 2$. So, we write $O(\log n)$ and ignore the base.

However, using unary encoding (where, e.g. 10 is expressed as 1111111111) does use exponentially more bits than binary encoding (or other k-ary encodings with $k \geq 2$).

15.3.2 Representing undirected graphs

Consider an undirected graph $G = (V, E)$. We describe three different ways to store G.

The *adjacency matrix* M of G is the $|V| \times |V|$ matrix with $M_{ij} = 1$ if the ith and jth vertex of G are adjacent and $M_{ij} = 0$ otherwise. If we store G using an adjacency matrix, the input size is $|V|^2$.

The *incidence matrix* N of G is the $|V| \times |E|$ matrix with $N_{ij} = 1$ if the ith vertex is incident to the jth edge and $N_{ij} = 0$ otherwise. If we store G using an incidence matrix, the input size is $|V| \cdot |E|$.

The third way of reprenting G is by having an *adjacency list* L_v for each vertex $v \in V$, where L_v contains all vertices that are adjacent to v in G. If we store G using adjacency lists, the input size is $|V| + 2|E|\log(|V|)$, because we need $|V|$ lists, and for each $e \in E$, we need to store two numbers that are at most $|V|$.

15.3.3 Representing directed graphs

Now consider a directed graph $D = (V, A)$. It can also be stored in (at least) three different ways, similar to the undirected case.

The *adjacency matrix* M of D is the $|V| \times |V|$ matrix with $M_{ij} = 1$ if there is an arc from the ith to the jth vertex of G and $M_{ij} = 0$ otherwise. If we store D using an adjacency matrix, the input size is $|V|^2$.

The *incidence matrix* N of D is the $|V| \times |A|$ matrix with $N_{ij} = 1$ if the ith vertex is the tail of to the jth arc, $N_{ij} = -1$ if the ith vertex is the head of the jth arc, and $N_{ij} = 0$ otherwise. If we store D using an incidence matrix, the input size is $O(|V| \cdot |A|)$.

The third way of representing D is by having an *in-list* L_v^{in} and an *out-list* L_v^{out} for each vertex $v \in V$, where L_v^{in} contains all vertices w for which there is an arc $(w, v) \in A$ and L_v^{out} contains all vertices w for which there is an arc $(v, w) \in A$. If we store G using in- and out-lists, the input size is $|V| + 2|A|\log(|V|)$, because we need $|V|$ lists, and for each $a \in A$, we need to store two numbers that are at most $|V|$.

15.4 Time complexity

We will now, more or less formally, define what we mean by the running time of an algorithm.

Definition 15.5. The *running time*, or *time complexity*, of an algorithm A, is the function $f : \mathbb{N} \to \mathbb{N}$ with $f(n)$ the number of elementary steps that A needs to solve a worst-case instance with input size n.

Here, *elementary steps*, includes all basic arithmetic operations such as addition, multiplication, comparing two numbers, etc. This model is not very precise, but it has the advantage that it is very easy to use. When a more precise model is needed, "Turing machines" can be used, but these will not be discussed here.

An algorithm is said to be *polynomial-time* if its running time is $O(p(n))$ with $p(n)$ a polynomial function of the input size n. An algorithm is said to be *exponential-time* if its running time is $\Omega(c^n)$ with $c > 1$.

Polynomial-time algorithms are greatly preferred over exponential-time algorithms because they scale much better. While an exponential-time algorithm may work well for small inputs, its running time will grow quickly for larger inputs, and for very large inputs the algorithm will never terminate (theoretically, it will terminate some time, but perhaps not within 1000 years).

Polynomial-time algorithms also profit much more from using faster computers. Suppose you get hold of a computer that is 10 times at fast, then a polynomial-time algorithm will be able to solve instances that are a certain factor larger (e.g. a factor 10 when the algorithm is linear-time), while an exponential-time algorithm will only be solve instances that are a few bits larger (e.g. $\log_2(10) \approx 3.3$ bits larger when the running time is 2^n). So, if an algorithm is polynomial-time, you get a multiplicative speedup, whereas the speedup is only additive if the algorithm is exponential-time.

15.5 Shortest path

The running time of Dijkstra's algorithm 10.3 for the SHORTEST PATH problem was analysed in Section 10.3.2. Since the running time is $O(|V|^2)$, this algorithm is a polynomial algorithm.

We note that the algorithm spends most time on finding a vertex $u \in V \setminus W$ minimizing $\rho(u)$, in each iteration. If we store the values $\rho(u)$ using a clever data structure, we can find a minimal value much faster and obtain a better running time. We will not discuss the details here, but only mention that Dijkstra's algorithm can be implemented in such a way that the running time is $O(|E| + |V| \log(|V|))$ (using a "Fibonacci heap"). For graphs with many edges (called *dense* graphs), this does not improve over $O(|V|^2)$. However, for many practical instances the number of edges is much less than $|V|^2$ (for example, $O(|V| \log(|V|))$) and then $O(|E| + |V| \log(|V|))$ is indeed much better than $O(|V|^2)$.

15.6 Max flow

In this section, we will analyse the running time of the Ford-Fulkerson algorithm for the MAXFLOW problem, described in Section 9.4.

At first sight, this may seem like a polynomial-time algorithm.

The number of iterations of the algorithm can be bounded by $b_{\max}|A|$ with $b_{\max} = \max_{(u,v)\in A} b_{uv}$. The reason for this is that in each iteration the value of the flow is increased by at least one, and the value of the flow can never get bigger than $b_{\max}|A|$. Constructing the auxiliary directed graph D_f takes $O(|A|)$ time. After this, we need to find, in each iteration, an s-t pad P in D_f, calculate α, modify the flow on P (which contains at most $|V|$ arcs) and update D_f. All of this takes at most $O(|V|^2)$ time, per iteration. Hence, the total running time can be bounded by $O(b_{\max}|A||V|^2)$.

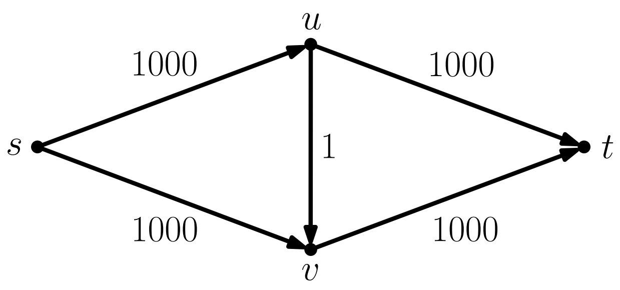


Figure 15.2: Example of a small instance of MaxFlow where the Ford-Fulkerson algorithm may need 2000 iterations. When the capacities that are 1000 are replaced by 2^N, the running time of the Ford-Fulkerson algorithm becomes exponential.

This may look like a polynomial function of the input size. However, it is not. The reason for this is that $b_{\max}$ can be much larger than the input size. Recall that the input size is the number of bits needed to store the input. To store a value $b_{\max}$, we need only $\log_2(b_{\max})$ bits. Moreover, $b_{\max} = 2^{\log_2(b_{\max})}$. Hence, $b_{\max}$ may be exponential in the input size.

So far, we have only argued that the current upper bound on the running time may be exponential in the input size. To see that the number of iterations may indeed be exponential, consider the example in Figure 15.2. Suppose the Ford-Fulkerson chooses to augment the flow over the path (s, u, v, t), then the value of the flow will increase by 1. The new auxiliary directed graph D_f will have a directed path (s, v, u, t), so the next iteration of the algorithm could choose to augment the flow over that path, again increasing the value of the flow by 1. The new auxiliary directed graph will again have a directed path (s, u, v, t) and so the algorithm may continue like this, increasing the value of the flow by 1 in each iteration. Since the value of the maximum flow is 2000, the algorithm will need 2000 iterations. This is a lot of iterations for such a small graph. Moreover, if we replace the capacities that are now 1000 by 2^N, then the algorithm will need 2^{N+1} iterations while the input size is $O(\log_n(2^N)) = O(N)$. Hence, the number of iterations may be exponential in the input size for this class of instances.

We may conclude that the Ford-Fulkerson algorithm (1956) is an exponential algorithm.

However, there is a simple improvement to the algorithm that makes it run in polynomial time. Instead of choosing an arbitrary s-t path in D_f, we choose a shortest s-t path in D_f (shortest w.r.t. the number of arcs). If we do this in each iteration, we need only a polynomial number of iterations. This was first shown by Dinitz (1970) and independently by Edmonds and Karp (1972). To prove this, we first introduce some notation and prove a lemma that will be the main ingredient of the proof.

Consider a directed graph $D = (V, A)$ with $s, t \in V$. Let $\mu(D)$ denote the length of a shortest s-t path in D and let $\alpha(D)$ denote the set of all arcs that lie on at least one shortest s-t path. Finally, let $\overleftarrow{\alpha(D)} := \{(v, u) : (u, v) \in \alpha(D)\}$.

The following lemma says the following. If, for each arc that lies on some shortest s-t path, we add the reverse arc to D, then this does not change the length of a shortest s-t path, nor the set of arcs that lie on a shortest s-t path.

Lemma 15.1. If $D = (V, A)$ is a directed graph with $s, t \in V$ and $D' = (V, A \cup \overleftarrow{\alpha(D)})$, then $\mu(D) = \mu(D')$ and $\alpha(D) = \alpha(D')$.

Proof. Consider some arc $a = (u, v) \in A$ that lies on a shortest s-t path P and let $\overleftarrow{a} = (v, u)$. We will show that adding $\overleftarrow{a}$ to D then $\mu(D)$ and $\alpha(D)$ remain the same. The lemma will follow because we can simply add each such arc one by one.

Suppose that adding $\overleftarrow{a}$ to D does change at least one of $\mu(D)$ and $\alpha(D)$. This is only possible when adding $\overleftarrow{a}$ creates a new shortest s-t path Q, which must contain $\overleftarrow{a}$.

Let P_1 denote the part of path P between s and u and let P_2 denote the part of P between v

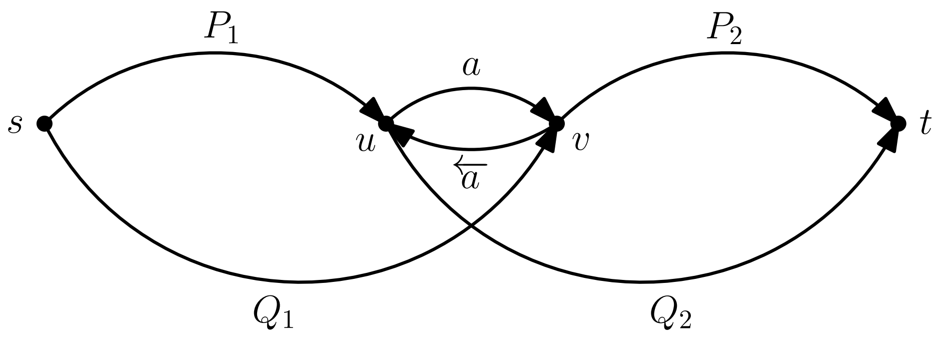


Figure 15.3: Illustration for the proof of Lemma 15.1.

and t. Let Q_1 denote the part of Q between s and v and let Q_2 denote the part of Q between u and t. See Figure 15.3.

We consider two cases. If P_2 is shorter than Q_2, then Q_1 combined with P_2 is shorter than Q, contradicting that Q is a shortest s-t path in $D \cup \{\overleftarrow{a}\}$.

The second case is that P_2 is at least as long as Q_2. If that is the case, then P_1 combined with Q_2 is shorter than P. Since P_1 and Q_2 do not use $\overleftarrow{a}$, this contradicts that P is a shortest s-t path in D.

Since we obtain a contradiction in both cases, we may conclude that our assumption that adding $\overleftarrow{a}$ to D changes at least one of $\mu(D)$ and $\alpha(D)$ was wrong. The lemma follows. $\square$

We are now ready to prove the theorem of Dinitz and Edmonds-Karp.

Theorem 15.2 (Dinitz, Edmonds-Karp)**.** The Ford-Fulkerson algorithm is polynomial-time if, in each iteration, an s-t path in D_f is chosen that has a minimum number of arcs.

Proof. Consider any iteration of the algorithm. Let f be the current flow and suppose that the flow is augmented to a new flow f', over a shortest s-t path P in the auxiliary directed graph D_f. Observe that the new auxiliary directed graph $D_{f'}$ is a subgraph of $D' = (V, A \cup \overleftarrow{\alpha(D)})$. Hence,

$$\mu(D_{f'}) \geq \mu(D') = \mu(D_f),$$

where the equality follows from Lemma 15.1.

In addition, if $\mu(D_{f'}) = \mu(D_f)$, then

$$\alpha(D_{f'}) \subseteq \alpha(D') = \alpha(D_f),$$

with again the equality following from Lemma 15.1.

Moreover, it is not possible that $\alpha(D_{f'}) = \alpha(D_f)$ because there is at least one arc $a^* = (u, v)$ on P with capacity $l_{uv} = \alpha$. This arc a^* is in $\alpha(D_f)$ but does not exist in $D_{f'}$ and can therefore not be in $\alpha(D_{f'})$. Therefore, we may conclude that, if $\mu(D_{f'}) = \mu(D_f)$, then $\alpha(D_{f'}) \subsetneq \alpha(D_f)$.

It follows that, in each iteration, either $|\alpha(D_f)|$ is reduced by at least one, or $\mu(D_f)$ is increased by at least one. Therefore, the number of iterations is at most $|A||V|$. Since we have argued before that at most $|V|^2$ time is needed for each iteration, the total running time is at most $O(|A||V|^3)$, which is polynomial.

$\square$

15.7 Linear programming

Although we will not show it here, there exist instances (called the "Klee-Minty cube") where the running time of the Simplex algorithm grows exponentially in the number of variables as it visits each vertex of the polytope. While each iteration can be executed very fast, the number of iterations can be exponential in the worst case. Hence, the Simplex algorithm is an exponential algorithm. It is still not known whether there exists some pivoting rule that does make the Simple algorithm polynomial. Nevertheless, for almost all practical instances, the Simplex method is very fast.

There do exist other algorithms, however, that solve linear programming in polynomial time. The first such algorithm was the Ellipsoid method (introduced in 1979), but it is not used in practice because for almost all practical instances it is much slower than the Simplex method. Some Interior Point Methods (introduced in 1984, many versions introduced later) also run in polynomial time and also perform very well in practice (comparable to the Simplex method). As the name suggests, such methods do not move from corner point to corner point of the feasible region, but move through the interior.

15.8 Exercises

Exercise 15.1. Let $f(n) = 10n^3 + 100n^2 + 200$ and $g(n) = n^3$. Show that $f(n) = \Theta(g(n))$.

Exercise 15.2. Let $f(n) = 2^n$ and $g(n) = n!$. Indicate for each of the following two statements whether it is true or false.

(a) $f(n) = O(g(n))$

(b) $g(n) = O(f(n))$

Exercise 15.3. Give an example of a running time that is neither polynomial nor exponential (you do not need to describe an algorithm, just a running time).

Exercise 15.4. For graphs $G = (V, E)$, we consider the number of edges $|E|$ as function of the number of vertices $n = |V|$.

(a) A graph is *connected* if there exists a path between each pair of vertices. Show that $|E| = \Omega(n)$ for the class of connected graphs.

(b) Show that $|E| = O(|V|^2)$ for all graphs.

(c) Show that $|E| = O(|V|)$ for trees.

Exercise 15.5. Explain that, for an algorithm with as input a connected graph, a running time of $O((|V| + |E|)^k)$ is equivalent to a running time of $O(|E|^k)$, for all $k \in \mathbb{N}$.

Exercise 15.6. Consider problems with as input a graph $G = (V, E)$ with a weight $w(e) > 0$ for each edge e. Adjust each of the three discussed graph representations (adjacency matrix, incidence matrix, adjacency lists) to represent such weighted graphs. Give, for each of the three representations, the input size as function of $|V|, |E|$ and $W = \max\{w(e) \mid e \in E\}$.

Exercise 15.7. Suppose you are in the middle of a labyrinth and you want to get out. The labyrinth consists of rooms connected by corridors. You come up with the following algorithm for searching for an exit:

- Every time you enter a corridor, you mark the entrance of that corridor with a $\times$.
- When you enter a certain room for the first time, you mark the entrance of the corridor through which you entered the room with a $\circ$.
- When you exit a room:
    1. you never enter a corridor of which the entrance is marked with a $\times$;
    2. you only enter a corridor of which the entrance is marked with a $\circ$ if there are no other options (i.e., when all other corridor entrances are marked with $\times$).

(a) Prove that you will find a way to an exit, assuming such a way exists.

(b) Estimate the running time of your algorithm as function of the number of rooms and number of corridors.

Remarks: The entrance of a corridor can get up to two marks ($\circ$ and $\times$). You do not erase any marks.

Exercise 15.8. Describe an algorithm with running time $O(|E|)$ for the following problem.

Given: a graph $G = (V, E)$ and vertices $s, t \in V$
Decide: does there exist a path from s to t in G?

Chapter 16

Complexity Theory

In the previous chapters, we have seen that many problems can be solved with polynomial-time algorithms: linear programming, max flow, minimum spanning tree and the shortest path problem. However, for integer linear programming, the described branch & bound method runs in exponential time. The reason for not describing a polynomial-time algorithm is that no such algorithm is known for this problem. Moreover, there are many other problems for which no polynomial algorithm is known even though these problems have been studied intensively. Why not? In this chapter, we will show how you can provide evidence against the existence of a polynomial-time algorithm for a given problem.

16.1 Optimization and decision problems

So far, we have discussed optimization problems, where the goal is to maximize or minimize a certain objective (e.g. maximize the value of the flow, or minimize the length of the path). However, in complexity theory, it is more convenient to work with decision problems, which are problems that ask for a yes or no as answer. When we want to study the complexity of an optimization problem, we consider the decision variant of that problem. Each optimization problem has a decision variant, which is obtained by setting an upper bound (for a maximization problem) or a lower bound (for a minimization problem) on the objective function. Consider, for example, the travelling salesman problem.

Travelling Salesman Problem (TSP)
Given: a complete graph $G = (V, E)$ and a length function $\ell : E \to \mathbb{Z}$.
Find: a Hamilton cycle in G with minimum length.

Recall that a Hamilton cycle of a graph is a cycle that visits each vertex exactly once. The length of a cycle C is defined as

$$\ell(C) := \sum_{e \in C} \ell(e).$$

This is an optimization problem, since we are asked to minimize the length of the cycle. The decision variant is as follows.

Travelling Salesman Problem Decision (TSPD)
Given: a complete graph $G = (V, E)$, a length function $\ell : E \to \mathbb{Z}$ and an integer $k \in \mathbb{Z}$.
Decide: does there exists a Hamilton cycle in G with length at most k?

This variant is clearly a decision problem because it asks for a yes or no answer.

Intuitively, the complexity of both problems must be the same. It is clear that you can solve TSPD if you can solve TSP. The other way round, if you can solve TSPD, then you can solve TSP by trying different values of k. Be careful, however, because the number of possible values of k can be exponential in the input size (when the edge lengths are very large). In such cases, you can use *binary search*, which works as follows.

It is clear that the length of a smallest tour must lie in $\{1, \dots, L\}$ with $L := \sum_{e \in E} \ell(e)$. Binary search first solves TSPD with $k = \lfloor L/2 \rfloor$. If the answer is yes, then the algorithm recursively solves TSP on the interval $\{1, \dots, \lfloor L/2 \rfloor\}$. If the answer is no, then the algorithm recursively solves TSP on the interval $\{\lfloor L/2 \rfloor + 1, \dots, L\}$. The number of times we need to solve TSPD is $O(\log_2(L))$, which is polynomial in the input size.

16.2 The classes P, NP and co-NP

Definition 16.1. The class P is the class of decision problems for which there exists a polynomial-time algorithm solving the problem.

Although we have defined the class P only for decision problems here, it can also be defined more generally for optimization problems. On the other hand, the next class that we define, the class NP, can only be defined for decision problems.

Definition 16.2. The class NP is the class of decision problems for which there exists a certificate for each yes-instance and a polynomial-time algorithm that checks whether a given certificate proves that a given instance is a yes-instance.

Example 16.1. Consider TSPD, the decision variant of TSP (see Section 16.1). For each yes-instance (G, ℓ, k), there exists a Hamilton cycle with length at most k. This Hamilton cycle can be used as certificate. There exists a polynomial-time algorithm that, given an instance (G, ℓ, k) and as certificate a set of edges $E' \subseteq E$, checks whether the edges form a Hamilton cycle and whether the sum of the lengths of the edges is at most k. Hence, TSPD is in the class NP. □

Intuitively, the class NP contains those decision problems for which a solution can be verified in polynomial time. In most cases, this more intuitive definition is good enough, because a solution to the problem can be used as certificate (as in the above example). However, there also exist problems in NP for which one needs a certificate that is different from a solution, because a solution does not give enough information. Also note that, although for most problems it is clear what we mean by a "solution", this term is not well defined for decision problems in general.

The name NP stands for "non-deterministic polynomial" (and *not* for "not polynomial"). This names comes from an alternative definition of the class NP, defining it as the class of decision problem that can be solved by a non-deterministic polynomial-time algorithm (or Turing Machine). Such a non-deterministic algorithm basically guesses the certificate and then checks it in polynomial time.

Whether the classes P and NP are equal or not is one of the seven Millennium Prize Problems and it is still open. When you solve the problem, you can win one million US dollars.

The class co-NP is defined similarly as NP, but with the role of yes-instances replaced by no-instances.

Definition 16.3. The class co-NP is the class of decision problems for which there exists a certificate for each no-instance and a polynomial-time algorithm that checks whether a given certificate proves that a given instance is a no-instance.

Example 16.2. We give three examples of problems in co-NP.

No Hamilton Cycle
Given: a graph $G = (V, E)$.
Decide: is there no Hamilton cycle in G?

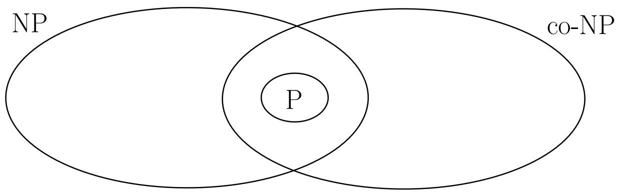


Figure 16.1: The complexity landscape under the assumption NP $\neq$ co-NP.

The problem No Hamilton Cycle is in the class co-NP because each no-instance has a Hamilton cycle that can serve as certificate. Given an instance and a Hamilton cycle, we can verify in polynomial time that the given cycle is a Hamilton cycle and the instance therefore a no-instance. It is not clear, however, whether No Hamilton Cycle is also contained in the class NP.

We turn to the next example.

LP Feasibility
Given: an $m \times n$ matrix A and a vector $\boldsymbol{b} \in \mathbb{Z}^m$.
Decide: does there exist a vector $\boldsymbol{x} \in \mathbb{R}^n$ such that $A\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{x} \geq \boldsymbol{0}$?

To see that LP Feasibility is in the class co-NP, we can use Farkas' Lemma (Theorem 7.1). For each no-instance, there exists a vector $\boldsymbol{y} \in \mathbb{R}^m$ for which $\boldsymbol{y}^\top A \geq \boldsymbol{0}^\top$ and $\boldsymbol{y}^\top \boldsymbol{b} < 0$. This vector $\boldsymbol{y}$ can serve as certificate and the inequalities $\boldsymbol{y}^\top A \geq \boldsymbol{0}^\top$, $\boldsymbol{y}^\top \boldsymbol{b} < 0$ are easily verified in polynomial time.

As a final example, we consider the decision version of Max Flow (see Section 9.1.2).

Max Flow Decision
Given: a directed graph $D = (V, A)$, vertices $s, t \in V$, a capacity function $b : A \to \mathbb{R}_{\geq 0}$ and a target flow $F \in \mathbb{R}_{\geq 0}$.
Decide: does there exist a feasible flow with value at least F?

Max Flow Decision is contained in the class co-NP since a cut with capacity greater than F can serve as certificate for a no-instance by the Max Flow Min Cut theorem (Theorem 9.1). Again, the certificate can easily be checked in polynomial time.

Note that LP Feasibility and Max Flow Decision are both also clearly contained in the class NP. Moreover, since we know that both problems are in fact polynomial-time solvable, we could also immediately have concluded from this fact that they are in NP $\cap$ co-NP.

It is also not known whether the classes NP and co-NP are equal or not. It seems extremely implausible that they are equal, but nobody has managed to prove that they are not.

It is clear that P $\subseteq$ NP $\cap$ co-NP. In fact, most problems that are known to be in NP $\cap$ co-NP are also known to be in P. An exception is the following problem, which is known to be in NP and in co-NP, while it is not known whether it is polynomial-time solvable.

Integer Factorization
Given: natural numbers m, n.
Decide: does m have a divisor $k \in \mathbb{N}$, with $1 < k < n$?

See Figure 16.1 for an illustration of the relationships between the complexity classes P, NP and co-NP.

16.3 Polynomial-time reductions

Reduction are used to show that one problem is at most as hard as another problem. We have actually already seen reductions. When we formulate a problem as an LP, this is actually a reduction. It shows that the problem is at most as hard to solve as a general LP. For example, in Section 9.2, we have formulated the MAX FLOW problem as an LP. Since we know that LPs can be solved in polynomial time, it follows that MAX FLOW can also be solved in polynomial time. In addition, in Section 9.5, we have shown for several problems that they can be reduced to the max flow problem and can therefore also be solved in polynomial-time.

We now formalize this concept.

Definition 16.4. A *polynomial-time reduction* from a decision problem Λ to a decision problem Π is a function that assigns to each instance I of Λ an instance $f(I)$ of Π, such that:

1. there is a polynomial-time algorithm that computes f;
2. for each instance I of Λ, I is a yes-instance of Λ if and only if $f(I)$ is a yes-instance of Π.

We write $\Lambda \propto \Pi$ to denote that there exists a polynomial-time reduction from Λ to Π. This means that Λ is at most as hard as Π or, in other words, that Π is at least as hard as Λ.

Note that we show that a *general* instance of Λ can be transformed into *some* instance of Π. Hence, you can also interpret $\Lambda \propto \Pi$ as meaning that Λ is a special case of Π. Indeed, it is possible that Π is strictly harder than Λ, because there can be other instances of Π that are much harder to solve than the instances created by the reduction from Λ.

Example 16.3. To illustrate the definition, we give a simple polynomial-time reduction between the following two decision problems.

EVEN PATH
Given: a graph $G = (V, E)$ and two vertices $s, t \in V$.
Decide: is there a path from s to t in G that uses an even number of edges?

ODD PATH
Given: a graph $G = (V, E)$ and two vertices $s, t \in V$.
Decide: is there a path from s to t in G that uses an odd number of edges?

We show that EVEN PATH $\propto$ ODD PATH, i.e., we give a polynomial-time reduction from EVEN PATH to ODD PATH.

The reduction is as follows. Given an arbitrary instance $(G = (V, E), s, t)$ of EVEN PATH, we create an instance $(G' = (V', E'), s', t')$ of ODD PATH by introducing a new vertex s' connected by a new edge to s and setting $t' = t$, i.e.:

- $V' = V \cup \{s'\}$;
- $E' = E \cup \{\{s, s'\}\}$;
- $t' = t$.

It is clear that this reduction can be executed in polynomial time. Hence, it remains to prove that (G, s, t) is a yes-instance of EVEN PATH if and only if (G', s', t') is a yes-instance of ODD PATH.

First suppose that (G, s, t) is a yes-instance of EVEN PATH. Hence, there exists a path from s to t in G that uses an even number of edges. This path also exists in G'. Moreover, in G', we can extend the path by the new edge $\{s', s\}$. We now have a path between s' and $t' = t$. Since the original path used an even number of edges, and we have added a single edge, the new path uses an odd number of edges. Hence, (G', s', t') is a yes-instance of ODD PATH.

We still need to show the reverse direction. Hence, suppose that (G', s', t') is a yes-instance of ODD PATH. Then there exists a path from s' to t' in G' that uses an odd number of edges. This path must use the edge $\{s', s\}$ because this is the only edge incident to s'. Moreover, the edge $\{s', s\}$ must be the first edge on the path and removing it gives a path from s to $t' = t$. This path also exists in G. Since the original path used an odd number of edges, and we removed a single edge, the new path uses an even number of edges. Hence, (G, s, t) is a yes-instance of EVEN PATH. This concludes the proof.

It is important to note that, although the two directions of the proof seem similar at first sight, they are not identical. Indeed, the reverse direction only works because we have created G' in such a way that a path from s' to t' *has* to use the edge $\{s', s\}$.

We have shown that EVEN PATH $\propto$ ODD PATH. Hence, if we have an algorithm to solve ODD PATH, then we can use it to solve EVEN PATH too. This means that EVEN PATH is at most as hard as ODD PATH. In this case, it is easy to see that the same reduction can also be used to reduce ODD PATH to EVEN PATH. Hence, both problems are just as hard.

Polynomial-time reductions can be used in two different ways. By the following lemma, whose proof is trivial, we can use polynomial-time reductions to show that a problem is polynomial-time solvable.

Lemma 16.1. If $\Lambda \propto \Pi$ and $\Pi \in$ P, then $\Lambda \in$ P.

The following sections will show how polynomial-time reductions can also be used to prove that a problem is, in a certain sense, hard.

16.4 NP-completeness

We now get to the most important definition of this chapter. Recall that our goal is to provide evidence against the existence of a polynomial-time algorithm for a problem. This is formalized as follows.

Definition 16.5. A decision problem Π is *NP-hard* if $\Lambda \propto \Pi$ for each $\Lambda \in$ NP.

In words, this definition says that a problem Π is NP-hard if it is at least as hard as every problem in the class NP. This means that, if we are able to solve Π in polynomial time, then we can solve every problem from the class NP in polynomial time. Strictly speaking, this does not prove that there exists no polynomial-time algorithm for Π, because it could still be that all problems in NP are polynomial-time solvable, i.e. that P and NP are equal.

Nevertheless, when a problem is NP-hard, this is seen as very strong evidence against the existence of a polynomial-time algorithm for the following reasons. Firstly, it seems extremely implausible that P and NP are equal, because this would mean, intuitively, that each problem for which you can verify a solution in polynomial time you can also find a solution in polynomial time. Secondly, if a problem is NP-hard and you would find a polynomial-time algorithm for it, then you would immediately be able to solve all problems in the class NP, including thousands of problems that have been studied for many years without anyone finding a polynomial-time algorithm for it.

We now define the class NP-complete, which consists of the hardest problems in NP. See Figure 16.2 for an illustration of the relationships between the various complexity classes.

Definition 16.6. A decision problem is *NP-complete* if it is NP-hard and in the class NP.

It is not directly clear that there even exist problems that are NP-complete. The SAT problem, which we describe next, was the first-ever problem shown to be NP-complete (in 1971).

A *boolean variable* is a variable that can take the values true and false. A *boolean expression* is a formula consisting of boolean variables and the symbols $\vee$, $\wedge$ and $\overline{x}$, which have the following meaning:

- $\overline{x}$ is the *negation* of x, which means "not x"; more formally, $\overline{x}$ is true if and only x is false;
- $x \vee y$ is the *disjunction* of x and y, which means "x or y"; more formally, $x \vee y$ is true if and only if at least one of x and y is true;

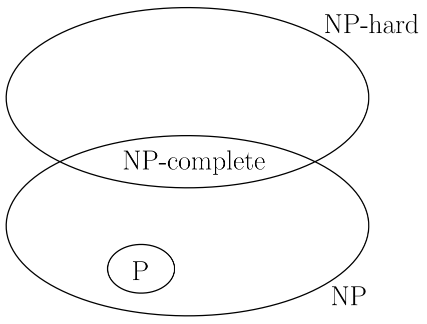


Figure 16.2: The complexity landscape under the assumption P $\neq$ NP.

- $x \wedge y$ is the *conjunction* of x and y, which means "x and y"; more formally, $x \wedge y$ is true if and only if x and y are both true.

A *literal* is either a boolean variable or a negation of a boolean variable. A *clause* is a *disjunction* of literals. A boolean expression is in *Conjunctive Normal Form (CNF)* if it is a conjunction of clauses. The following theorem is well known in logic.

Theorem 16.1. Each boolean expression can be written in conjunctive normal form.

A *truth assignment* for a boolean expression is a function that assigns to each boolean variable appearing in the expression either the value true or false. It is a *satisfying truth expression* if it makes the boolean expression true. When there exists a satisfying truth expression for a boolean expression then the expression is called *satisfiable.*

Example 16.4. The following boolean expression is in conjunctive normal form.

$$(x \vee y \vee \overline{z}) \wedge (\overline{x} \vee q) \wedge (\overline{y} \vee \overline{q} \vee z)$$

A satisfying truth assignment is, for example, $x = \text{true}, y = \text{false}, z = \text{false}, q = \text{true}$. This truth assignment is satisfying because $x = \text{true}$ makes the first clause true, $q = \text{true}$ makes the second clause true and $y = \text{false}$ makes the third clause true. Since all clauses are true, the complete expression is also true.

Another truth assignment is, for example, $x = \text{false}, y = \text{true}, z = \text{false}, q = \text{true}$. This truth assignment is not satisfying because it makes the third clause $\overline{y} \vee \overline{q} \vee z$ false.

We are now ready to define the first problem that was shown to be NP-complete.

Satisfiability (SAT)
Given: a boolean expression S in conjunctive normal form.
Decide: does there exist a satisfying truth assignment for S?

Theorem 16.2 (Cook, Levin, 1971)**.** SAT is NP-complete

We will not prove the Cook-Levin theorem here, because the proof is extremely technical. It shows that any problem $\Lambda \in$ NP can be reduced to SAT by a polynomial-time reduction. This is especially difficult because we know nothing about the problem Λ except that it is in the class NP, i.e., there exists a certificate for each yes-instance and a polynomial-time algorithm that verifies whether a given certificate proves that a given instance is a yes-instance. The main high-level idea of the proof is to encode the certificate using boolean variables and, moreover, to encode the whole verification algorithm as a boolean expression. This eventually gives, for a given instance I of Λ, a boolean expression S_I that is satisfiable if and only if there exists a certificate proving that instance I is a yes-instance. Clearly, there exists a certificate proving that instance I is a yes-instance if and only if I is a yes-instance. Hence, S_I is satisfiable if and only if I is a yes-instance.

Luckily, this complicated way of proving NP-completeness had to be applied only once. Once we knew for one problem that it is NP-complete, we could use this to prove NP-completeness for other problems too. This is explained in the next section.

16.5 Proving NP-completeness

To show that a problem is NP-complete, there are two things to do. We need to prove that it is in the class NP. This is described in Section 16.2 and usually easy. The second thing to do is to show NP-hardness, which is often more difficult. The main idea is that we can use the NP-hardness of one problem to show that another problem is also NP-hard by giving a polynomial-time reduction. This is based on the following lemma, the proof of which is trivial.

Lemma 16.2 (Transitivity of $\propto$)**.** If $\Lambda \propto \Pi$ and $\Pi \propto \Sigma$, then $\Lambda \propto \Sigma$.

The next lemma follows directly from the previous one, and is the key to proving NP-hardness.

Lemma 16.3. If $\Lambda \propto \Pi$ and Λ is NP-hard, then Π is NP-hard.

Hence, we can show that a problem is NP-hard by giving a polynomial-time reduction *from* a problem for which we know it is NP-hard.

We start by showing that Integer Linear Programming is NP-complete. Moreover, we show that it is even NP-complete when all variables are binary and we only need to decide whether there exists a feasible solution, i.e., we consider the following special case of ILP.

BINARY ILP FEASIBILITY
Given: integer $m \times n$ matrix A and vector $\boldsymbol{b} \in \mathbb{Z}^m$.
Decide: does there exist a vector $\boldsymbol{x} \in \{0,1\}^n$ such that $A\boldsymbol{x} \leq \boldsymbol{b}$?

Theorem 16.3. BINARY ILP FEASIBILITY is NP-complete.

Proof. First we show that BINARY ILP FEASIBILITY is in the class NP. We take as certificate a vector $\boldsymbol{x} \in \{0,1\}^n$ for which $A\boldsymbol{x} \leq \boldsymbol{b}$. Such a vector exists for each yes-instance $(A, \boldsymbol{b})$. Given an instance $(A, \boldsymbol{b})$ and a certificate $\boldsymbol{x}$, we can clearly verify in polynomial time whether $A\boldsymbol{x} \leq \boldsymbol{b}$. Hence, BINARY ILP FEASIBILITY is in the class NP.

It remains to show that BINARY ILP FEASIBILITY is NP-hard. We do this by giving a polynomial-time reduction from the problem SAT, which we know to be NP-hard by Theorem 16.2.

The reduction is as follows. Let $C = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ be an arbitrary instance of SAT and let $u_1, \ldots, u_n$ be the boolean variables appearing in C. Then we create an instance of BINARY ILP FEASIBILITY as follows.

For each $i \in \{1, \ldots, n\}$, we create binary variables x_i and y_i and a constraint

$$x_i + y_i = 1.$$

In addition, for each $k \in \{1, \ldots, m\}$, we create a constraint

$$\sum_{i \in I} x_i + \sum_{j \in J} y_j \geq 1,$$

with I the set of indices i for which u_i appears in clause C_k without negation and J the set of indices j for which u_j appears in clause C_k with negation.

This concludes the reduction. It is clear that the constraints can be written in the form $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ and we have obtained an instance of Binary ILP Feasibility.

For example, consider the instance $(u_1 \vee u_2 \vee \overline{u_3}) \wedge (\overline{u_1} \vee u_3 \vee u_4) \wedge (\overline{u_1} \vee \overline{u_2} \vee \overline{u_4})$ of SAT. Then we obtain the following instance of Binary ILP Feasibility:

$$\begin{array}{ccccccccccccccccc}
x_1 & + & y_1 & & & & & & & & & & & & & = & 1 \\
 & & & & x_2 & + & y_2 & & & & & & & & & = & 1 \\
 & & & & & & & & x_3 & + & y_3 & & & & & = & 1 \\
 & & & & & & & & & & & & x_4 & + & y_4 & = & 1 \\
x_1 & & & + & x_2 & & & & & + & y_3 & & & & & \geq & 1 \\
 & & y_1 & & & & & + & x_3 & & & + & x_4 & & & \geq & 1 \\
 & & y_1 & & & + & y_2 & & & & & & & + & y_4 & \geq & 1
\end{array}$$
$$x_1, \ldots, x_4,\ y_1, \ldots, y_4 \in \{0, 1\}$$

We now continue the proof. First assume that we have a yes-instance of SAT. Hence, there exists a satisfying truth assignment t for C. Then we can find a feasible solution to $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ as follows. First, for each boolean variable u_i that is set to true by t, we set $x_i = 1$ and $y_i = 0$. Secondly, for each boolean variable u_i that is set to false by t, we set $x_i = 0$ and $y_i = 1$. It is clear that this solution satisfies the constraints $x_i + y_i = 1$. Now consider a constraint

$$\sum_{i \in I} x_i + \sum_{j \in J} y_j \geq 1.$$

Since the corresponding clause C_k is made true by C, there is either a literal u_i in C_k with u_i set to true by t or a literal $\overline{u_j}$ in C_k with u_j set to false by t. In the first case, a variable x_i with $i \in I$ is set to 1. In the second case, a variable y_j with $j \in J$ is set to 1. In either case, the constraint is satisfied. Hence, we have a yes-instance of Binary ILP Feasibility.

Now assume that we have a yes-instance of Binary ILP Feasibility. Then there exists a binary vector $\boldsymbol{x}$ that satisfies all the constraints. We show that there then exists a satisfying truth assignment for C. By the constraint $x_i + y_i = 1$, either $x_i = 0$ and $y_i = 1$ or $x_i = 1$ and $y_i = 0$, for each $i \in \{1, \ldots, n\}$. In the first case ($x_i = 0$ and $y_i = 1$), we set u_i to false. In the second case ($x_i = 1$ and $y_i = 0$), we set u_i to true. Now consider a clause C_k. By the constraint

$$\sum_{i \in I} x_i + \sum_{j \in J} y_j \geq 1,$$

there either exists an $i \in I$ with $x_i = 1$ or a $j \in J$ with $y_j = 1$. In the first case, u_i appears without negation in C_k and is set to true. In the second case, $\overline{u_j}$ appears in C_k and u_j is set to false. In either case, clause C_k is satisfied. As this holds for each clause, we have shown that C is satisfiable and hence we have a yes-instance of SAT. □

To get back to the example above, a satisfying truth assignment of the SAT instance is, for example, $u_1 = \text{true}, u_2 = \text{false}, u_3 = \text{true}, u_4 = \text{false}$. This corresponds to the following feasible solution to the ILP instance: $x_1 = 1, y_1 = 0, x_2 = 0, y_2 = 1, x_3 = 1, y_3 = 0, x_4 = 0, y_4 = 1$.

To give one more example of an NP-completeness proof, we will show that the following problem is also NP-complete. A *clique* of a graph $G = (V, E)$ is a complete subgraph, i.e., if $U \subseteq V$ and $\{u, v\} \in E$ for all $u, v \in U$ with $u \neq v$ then the subgraph of G induced by U is a clique of G.

Clique
Given: a graph $G = (V, E)$ and a natural number k;
Decide: does G have a clique with at least k vertices?

For an example, see Figure 16.3.

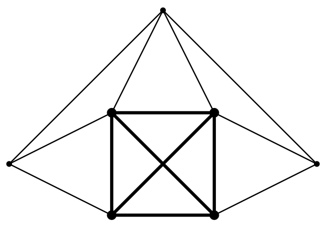


Figure 16.3: Graph with a clique of 4 vertices indicated in bold. There is no clique with more than 4 vertices in this graph.

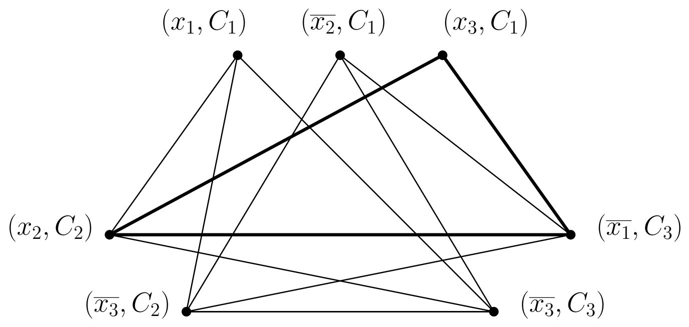


Figure 16.4: Instance of Clique with $k = 3$ obtained from the instance $(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_3})$ of SAT. One clique with $k = 3$ vertices is indicated in bold, giving satisfying truth assignment $x_1 = \text{false}, x_2 = \text{true}, x_3 = \text{true}$ for SAT.

Theorem 16.4. Clique is NP-complete.

Proof. Clique is in NP because given an instance (G, k) and as certificate a set $U \subseteq V$, we can check in polynomial time whether $|U| \geq k$ and $\{u, v\} \in E$ for all $u, v \in U$ with $u \neq v$.

We show that Clique is NP-hard by giving a reduction from SAT to Clique.

The reduction is as follows. Let $C = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ be an arbitrary instance of SAT. Then we create an instance $(G = (V, E), k)$ of Clique as follows:

- $k = m$;
- $V = \{(\sigma, C_i) : \sigma \text{ is a literal in } C_i \text{ and } 1 \leq i \leq m\}$;
- $E = \{(\sigma, C_i), (\tau, C_j) : i \neq j \text{ and } \tau \neq \overline{\sigma}\}$.

For an example, see Figure 16.4.

First suppose we have a yes-instance of SAT, i.e., there exists a satisfying truth assignment for C. Then each clause C_i contains at least one literal that is made true by the truth assignment. We choose one such literal arbitrarily and call it σ_i. Then

$$U = \{(\sigma_i, C_i) : 1 \leq i \leq m\}$$

is a set of $m = k$ vertices of G. Consider any two vertices $(\sigma_i, C_i), (\sigma_j, C_j) \in U$. Then $i \neq j$ and σ_i can't be the negation of σ_j because both are made true by the truth assignment. Hence, there is an edge $((\sigma_i, C_i), (\sigma_j, C_j)) \in E$. Since this is true for any two vertices in U, we can conclude that G has a clique of at least k vertices, i.e, we have a yes-instance of Clique.

Now suppose we have a yes-instance of Clique. Then there exists a clique in G with at least k vertices. Let K be the vertex set of the clique. Then we can find a satisfying truth assignment for C

as follows. For each vertex $(\sigma, C_i) \in K$ we set σ to true. More precisely, if σ is a boolean variable u then we set u to true and if σ is a negated boolean variable $\overline{u}$ then we set u to false. Note that this way we have assigned to each boolean variable at most one value, because there are no edges between two vertices corresponding to a variable and its negation and hence such a pair of vertices can't both be in K. If there are boolean variables that have not yet been assigned a value then we arbitrarily assign them the value true or false.

It remains to show that the described truth assignment is satisfying. First note that there are no edges between vertices corresponding to the same clause. Hence, K can contain at most one vertex per clause. Since K contains at least k vertices, it follows that K contains exactly one vertex (σ, C_i) for each clause C_i. The truth assignment makes σ true because $(\sigma, C_i) \in K$ and hence the truth assignment makes C_i true. Since this holds for each clause, we have shown that there exists a satisfying truth assignment for C, which is therefore a yes-instance of SAT. □

16.6 Exercises

Exercise 16.1. Consider the following SAT instance:

$$S = (x_1 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3} \vee x_4)$$

(a) Does there exist a satisfying truth assignment for S?

(b) Construct a boolean expression S', with exactly three literals per clause, that is true if and only if S is true.

(c) Show that 3-SAT, the variant of SAT with exactly three literals per clause, is NP-complete.

Exercise 16.2. Consider the following two decision problems.

Vertex Cover Decision
Given: a graph G and an integer $k \in \mathbb{Z}$.
Decide: does there exist a vertex cover in G consisting of at most k vertices? (A *vertex cover* is a set of vertices that covers all edges, i.e., each edge has at least one endpoint in the vertex cover.)

Independent Set Decision
Given: a graph G and a number $k \in \mathbb{Z}$.
Decide: does there exist an independent set in G consisting of at least k vertices? (An *independent set* is a set of vertices that is not connected by edges, i.e., no edge has both endpoints in the independent set.)

(a) Show that for a graph $G = (V, E)$ and $W \subseteq V$ the following statements are equivalent:

(1) W is a vertex cover;

(2) $V \setminus W$ is an independent set;

(3) $V \setminus W$ is a clique in the complement $\overline{G}$ of G (this is the graph with the same vertices as G and an edge $\{u, v\}$ precisely if $\{u, v\} \notin E$).

(b) Show that Vertex Cover Decision and Independent Set Decision are NP-complete.

Exercise 16.3. Prove Lemma 16.3.

Exercise 16.4. Prove that the following problem DIRECTED TSP DECISION is NP-complete.

DIRECTED TSP DECISION
Given: a complete directed graph $D = (V, A)$, a length function $\ell : A \to \mathbb{Z}^+$ and an integer $k \in \mathbb{Z}$.
Decide: does there exist a directed cycle in D that visits each vertex exactly once and has total length as most k?

You may use that the following problem DIRECTED HAMILTONIAN CYCLE is NP-complete.

DIRECTED HAMILTONIAN CYCLE
Given: a directed graph D.
Decide: does there exist a directed cycle in D that visits each vertex exactly once?

Exercise 16.5. Harry Potter is looking for a bowtruckle that is hiding in a graph and has made itself invisible. Harry tries to find the bowtruckle by casting the spell *rivilio trullio* while aiming his wand at a vertex of the graph. The first time he hits the bowtruckle it will stay invisible but move to a neighbouring vertex of the graph. However, Harry will not be able to notice this because the bowtruckle moves silently. The second time that Harry hits the bowtruckle it will become visible. Harry's wand has just enough energy left to cast the spell *rivilio trullio* k times. Therefore, Harry wants to know whether he can choose at most k vertices of the graph such that he is sure that he will find the bowtruckle when he casts *rivilio trullio* on each of these vertices. Show that this problem is NP-complete.

Exercise 16.6. Consider the following two decision problems.

PARTITION
Given: positive integers $a_1, \ldots, a_n$.
Decide: does there exist a subset S of $\{1, \ldots, n\}$ such that

$$\sum_{j \in S} a_j = \sum_{j \notin S} a_j?$$

0,1-KNAPSACK
Given: positive integers $c_1, \ldots, c_m$ and k.
Decide: does there exist a vector $x \in \{0, 1\}^m$ such that

$$\sum_{j=1}^{m} c_j x_j = k?$$

Given that PARTITION is NP-complete. Show that 0,1-KNAPSACK is also NP-complete.

Exercise 16.7. Suppose you study n patients that suffer from a certain disease and m genes that might be related to this disease. You suspect that, for each of the patients, the disease is caused by a mutation of one of these genes. Your data consists of an $n \times m$ matrix M with $M_{i,j} = 1$ if in patient i gene j is mutated and $M_{i,j} = 0$ otherwise. You want to find out which genes cause the disease. To find this out, you want to find a smallest possible set J of genes, such that in each patient at least one of the genes from J is mutated. The decision version of this problem is as follows:

GENES
Given: $n \times m$ matrix M with $M_{i,j} \in \{0, 1\}$ and $k \in \mathbb{Z}$.
Decide: does there exist a $J \subseteq \{1, \ldots, m\}$ with $|J| \leq k$ such that, for all $i \in \{1, \ldots, n\}$, there exists at least one $j \in J$ with $M_{i,j} = 1$?

Prove that GENES is NP-complete.

Exercise 16.8. Prove that the following decision problem is NP-complete.

NEIGHBOUR INCIDENT
Given: graph $G = (V, E)$ and natural number $k \in \mathbb{N}$.
Decide: does there exist a subset V' of V with $|V'| \leq k$ such that, for each edge $e \in E$, there exists a vertex $v \in V'$ for which e is incident to v or to a neighbour of v?

Chapter 17

Approximation Algorithms

In the previous chapter, we have seen that for NP-hard problems it is unlikely that we will ever find a polynomial-time algorithm. Unfortunately, many practical problems are NP-hard. However, for optimization problems, NP-hardness only means that there is no polynomial-time algorithm that always returns an optimal solution (unless P = NP). What if we want to find a solution that is perhaps not optimal, but "close" to being optimal? This chapter discusses approximation algorithms, which are polynomial-time algorithms that guarantee to always return a solution that is at most a certain factor worse than an optimal solution. These solutions given an upper or lower bound on the optimal value and can therefore very well be used in exponential-time methods that do search for an optimal solution, such as, for example, branch & bound methods.

17.1 Definitions

Definition 17.1. A *ρ-approximation algorithm* for an optimization problem Π is a polynomial-time algorithm that produces, for each instance I of Π, a solution, such that:

$$\begin{aligned} ALG(I) &\leq \rho \cdot OPT(I) \quad \text{if } \Pi \text{ is a minimization problem} \\ ALG(I) &\geq \rho \cdot OPT(I) \quad \text{if } \Pi \text{ is a maximization problem,} \end{aligned}$$

with $OPT(I)$ the value of an optimal solution and $ALG(I)$ the value of the solution returned by the algorithm, for instance I.

The factor ρ is called the *approximation ratio*. For minimization problems, $\rho \geq 1$ and we prefer an algorithm with ρ as small as possible. For maximization problems, $\rho \leq 1$ and we prefer an algorithm with ρ as large as possible. In both cases, $\rho = 1$ if and only if the algorithm always returns an optimal solution (i.e. it is a polynomial-time algorithm for the problem). Note that ρ is not necessarily a constant, there also exist, e.g., $O(\log(n))$-approximation algorithms. Approximation algorithms with ρ constant (*constant-factor approximation algorithms*) are greatly preferred over other approximation algorithms though.

It is important to understand the difference between approximation algorithms and *heuristics*. The latter type of algorithms also aims at producing solutions that are close to being optimal. However, the fundamental difference is that heuristics to not give any guarantee. Their performance is usually tested by doing experiments on simulated or real-life data. Approximation algorithms, on the other hand, do have a guaranteed performance. Nevertheless, approximation algorithms are rarely used by themselves in practice because heuristics typically perform better on practical data. The performance guarantees of approximation algorithms can be especially useful for computing bounds on the optimal value, e.g. in a branch & bound framework.

17.2 Vertex cover

The first problem for which we study approximation algorithms is the Vertex Cover problem, which you might recall from Exercise 16.2. A *vertex cover* is a set of vertices that covers all edges, i.e., each edge has at least one endpoint in the vertex cover. The associated optimization problem is defined as follows.

Vertex Cover
Given: a graph G.
Find: a vertex cover of G with minimum cardinality.

17.2.1 Greedy algorithm

Our first attempt at an approximation algorithm is the greedy algorithm displayed in Algorithm 7.

```
   Data: A graph $G = (V, E)$.
   Result: a vertex cover of $G$
1  Initialize $C := \emptyset$;
2  while $E \neq \emptyset$ do
3      Choose a vertex $v \in V$ with maximum degree $d(v)$;
4      Add $v$ to $C$;
5      Delete $v$ (and hence all incident edges) from $G$;
6  end
7  return $C$;
```

Algorithm 7: Greedy algorithm for Vertex Cover

The algorithm makes a lot of sense. We want to cover all edges, so why not choose a vertex that covers as many edges as possible in each step? Nevertheless, we show below that the greedy algorithm is a very bad approximation algorithm.

Theorem 17.1. The greedy algorithm is not a c-approximation algorithm, for any constant c.

Proof. Consider, as an instance of Vertex Cover, a bipartite graph $G = (U \cup W, E)$ with $|U| = n$ and $W = W_1 \cup \ldots \cup W_n$ with

$$\begin{aligned} |W_1| &= n \\ |W_2| &= \left\lfloor \frac{n}{2} \right\rfloor \\ |W_3| &= \left\lfloor \frac{n}{3} \right\rfloor \\ &\cdots \\ |W_n| &= \left\lfloor \frac{n}{n} \right\rfloor = 1 \end{aligned}$$

The edges are as follows. Each vertex from W_i is incident to i vertices from U, in such a way that each vertex from U is incident to at most one vertex from each W_i. See an example in Figure 17.1.

We first analyse what the greedy algorithm might do. Note that each vertex in U has degree $d(u) \leq n$, while each vertex in W_i has degree $d(w) = i$. The greedy algorithm will choose a vertex with maximum degree. Since the maximum degree is n, the algorithm either chooses a vertex from U or the vertex in W_n. The algorithm chooses a vertex with maximum degree arbitrarily, so it could choose the vertex in W_n. Suppose it does. Then the algorithm deletes the vertex in W_n and all incident edges, and adds this vertex to C.

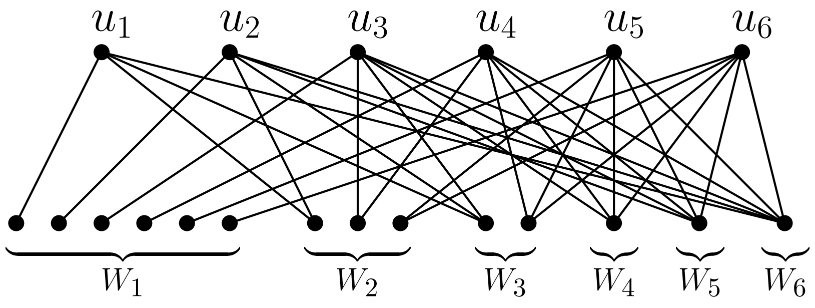


Figure 17.1: An instance of VERTEX COVER where the greedy algorithm may perform very poorly.

In the next iteration, we have that $d(u) \leq n-1$ for all $u \in U$ and still that $d(w) = i$ for each $w \in W_i$, with $1 \leq i \leq n-1$. Hence the vertices in W_{n-1} have maximum degree. Suppose the algorithm chooses a vertex in W_{n-1}. After deleting it, the remaining vertices (if any) in W_{n-1} still have maximum degree. Hence, the algorithm may choose them one by one untill all vertices from W_{n-1} have been deleted (and added to C).

The algorithm may continue like this, in each iteration choosing a vertex from W, until all vertices from W have been deleted (and added to C). At this point, only the vertices from U remain and all edges are gone, so the algorithm stops and returns C, which contains all vertices from W.

It is clear that $C = W$ is a vertex cover. In that sense the algorithm did a good job. However, look at the number of vertices in the vertex cover:

$$|C| = n + \lfloor\frac{n}{2}\rfloor + \lfloor\frac{n}{3}\rfloor + \ldots + \lfloor\frac{n}{n}\rfloor = \Omega(n\ln(n))$$

This is very bad, seeing that $C = U$ is also a vertex cover and contains only n vertices. It is also not so difficult to see that U is indeed an optimal solution. We now analyse the approximation ratio of the greedy algorithm.

$$\frac{ALG(G)}{OPT(G)} = \Omega\left(\frac{n\ln(n)}{n}\right) = \Omega(\ln(n))$$

Hence, the greedy algorithm is not a c-approximation algorithm, for any constant c. □

So the greedy algorithm is not a good approximation algorithm for the VERTEX COVER problem. Fortunately, we can do much better if we use Linear Programming.

17.2.2 LP-rounding algorithm

We now present a much smarter algorithm for VERTEX COVER, which can also be applied to the weighted variant of the problem.

WEIGHTED VERTEX COVER
Given: a graph $G = (V, E)$ and a weight $c_v \geq 0$ for each vertex $v \in V$.
Find: a vertex cover C of G with minimum total weight

$$\sum_{v \in C} c_v.$$

First, we formulate the problem as an ILP. For each vertex $v \in V$, we have a decision variable x_v defined as follows:

$$x_v = \begin{cases} 1 & \text{if } v \text{ is in the vertex cover} \\ 0 & \text{otherwise.} \end{cases}$$

The ILP formulation is as follows.

$$\begin{aligned} \min \quad & z = \sum_{v \in V} c_v x_v \\ \text{s.t.} \quad & x_u + x_v \geq 1 \quad \forall \{u,v\} \in E \\ & x_v \in \{0,1\} \quad \forall v \in V \end{aligned} \tag{17.1}$$

Since we cannot solve the ILP in polynomial time, we consider the LP relaxation:

$$\begin{aligned} \min \quad & z_{LP} = \sum_{v \in V} c_v x_v \\ \text{s.t.} \quad & x_u + x_v \geq 1 \quad \forall \{u,v\} \in E \\ & x_v \geq 0 \quad \forall v \in V \end{aligned} \tag{17.2}$$

Observe that, although we could require $x_v \leq 1$, this is not necessary since this will hold automatically in any optimal solution.

The LP-rounding algorithm for Weighted Vertex Cover is described in Algorithm 8.

```
Data: A graph $G = (V, E)$ and a weight $c_v$ for each $v \in V$.
Result: a vertex cover of $G$
1 Solve the LP-relaxation (17.2) and let $x^*$ the an optimal solution found;
2 return $C := \{v \in V \mid x_v^* \geq \frac{1}{2}\}$;
```

Algorithm 8: LP-rounding algorithm for Weighted Vertex Cover

We will now show that, unlike the greedy algorithm, the LP-rounding algorithm does give a good approximation ratio.

Theorem 17.2. The LP-rounding algorithm is a 2-approximation algorithm for Weighted Vertex Cover.

Proof. Since we know that an LP instance can be solved in polynomial time, it is clear that the LP-rounding algorithm runs in polynomial time.

We now check that the algorithm always returns a vertex cover. Consider any edge $\{u, v\}$. Because of the restriction $x_u + x_v \geq 1$, we know that at least one of x_u^*, x_v^* has value at least $\frac{1}{2}$. Hence, the LP-rounding algorithms puts at least one of u and v in C. Since this holds for each edge, it follows that the returned C is a vertex cover.

It remains to find the approximation ratio of the algorithm. For any instance I, we have

$$ALG(I) = \sum_{v \in C} c_v = \sum_{v \in V \mid x_v^* \geq \frac{1}{2}} c_v \leq 2 \sum_{v \in V} c_v x_v^* = 2 z_{LP} \leq 2z = 2OPT(I)$$

The first inequality holds because $c_v \leq 2 c_v x_v^*$ for each vertex with $x_v^* \geq \frac{1}{2}$. The second inequality holds in general because the LP-relaxation is a relaxation of the ILP.

Therefore, we can conclude that the LP-rounding algorithm is a 2-approximation algorithm. □

17.3 Traveling salesman problem

In this section, we analyse approximation algorithms for the traveling salesman problem (TSP). We first consider the general problem on a weighted graph and then turn to a more restricted variant, which will turn out to be easier to approximate.

17.3.1 General TSP

We first consider TSP with any length function allowed.

Traveling Salesman Problem (TSP)
Given: a complete graph $G = (V, E)$ and a length function $\ell : E \to \mathbb{R}_{\geq 0}$.
Find: a Hamilton cycle C of G with minimum length

$$\sum_{e \in C} \ell(e).$$

Unfortunately, TSP is very hard to approximate, as we will show in the next theorem.

We basically give a polynomial-time reduction from the following NP-hard problem Hamilton Cycle to TSP.

Hamilton Cycle
Given: a graph $G = (V, E)$.
Decide: does G have a Hamilton cycle?

The difference with the polynomial-time reductions in Chapter 16 is that TSP is an optimization problem and we are interested in approximation algorithms, while in Chapter 16 we have only considered reductions between decision problems. Nevertheless, the same technique turns out to work here.

Theorem 17.3. There is no c-approximation algorithm for TSP, for any constant c, unless P = NP.

Proof. Suppose there exists a c-approximation algorithm for TSP. Then we can solve Hamilton Cycle in polynomial-time as follows.

Let $G = (V, E)$ be an arbitrary instance of Hamilton Cycle. Then we construct an instance $(G' = (V', E'), \ell)$ of TSP as follows. Graph G' is a complete graph with the same vertices as G, so $V' = V$. The length function ℓ is defined as follows. For each $e \in E'$,

$$\ell(e) = \begin{cases} 1 & \text{if } e \in E \\ 2 + (c-1)|V| & \text{if } e \notin E \end{cases}$$

We claim that the c-approximation algorithm for TSP returns a tour with length $|V|$ if and only if there exists a hamilton cycle in G.

The "only if" direction of the claim is easy to see. If the c-approximation algorithm returns a tour of length $|V|$, then this tour must consist of only edges of length 1. Hence, by the definition of ℓ, these edges are all edges of G. Since each TSP tour is a Hamilton cycle, and both graphs have the same vertices, this means that the tour is a Hamilton cycle for G.

To show the "if" direction, assume that there exists a Hamilton cycle in G. Hence, $OPT(G', \ell) = |V|$. Suppose now that the c-approximation algorithm for TSP does not return a tour with length $|V|$. Then it returns a tour with length greater than $|V|$, which must therefore use at least one of the edges with length $2 + (c-1)|V|$. Hence, the total length of the tour is

$$ALG(G', \ell) \geq |V| - 1 + 2 + (c-1)|V| = 1 + c|V|$$

and therefore,

$$\frac{ALG(G', \ell)}{OPT(G', \ell)} \geq \frac{1 + c|V|}{|V|} > c,$$

which is not possible because the algorithm is a c-approximation algorithm. This concludes the proof of the claim.

By this claim, we can use the c-approximation algorithm for TSP to solve Hamilton Cycle in polynomial time. Since Hamilton Cycle is NP-hard, this implies that P = NP. □

17.3.2 TSP with triangle inequality

Since we have seen that there exists no constant-factor approximation algorithm for general TSP, we now study a restricted variant. Fortunately, many practical instances satisfy the following restriction on on the distances (edge lengths).

Definition 17.2. A distance function $d : V \times V \to \mathbb{R}_{\geq 0}$ satisfies the *triangle inequality* if

$$d(u, v) \leq d(u, w) + d(w, v) \quad \forall u, v, w \in V.$$

We now consider TSP restricted to distances that satisfy the triangle inequality, which is formally defined as follows.

$\triangle$TSP
Given: a complete graph $G = (V, E)$ and a length function $\ell : E \to \mathbb{R}_{\geq 0}$ satisfying the triangle inequality.
Find: a Hamilton cycle C of G with minimum length

$$\sum_{e \in C} \ell(e).$$

Although $\triangle$TSP is still NP-hard, it turns out to be much easier to approximate. An approximation algorithm is described in Algorithm 9. It uses algorithms for finding a minimum spanning tree (see Chapter 11) and an Euler tour (Appendix A).

```
Data: An instance $(G = (V, E), \ell)$ of $\triangle$TSP.
Result: a Hamilton cycle of $G$
1 Find a minimum spanning tree $T$ of $(G, \ell)$;
2 Replace each edge $e$ of $T$ by two parallel edges, both with length $\ell(e)$. Let $M$ be the resulting
   multigraph;
3 Find an Euler tour $W$ in $M$;
4 Let $C$ be the cycle in $G$ obtained from $W$ by skipping vertices that have already been visited;
5 return $C$;
```

Algorithm 9: Tree algorithm for $\triangle$TSP

Theorem 17.4. The tree algorithm for $\triangle$TSP is a 2-approximation algorithm.

Proof. The running time is clearly polynomial because a minimum spanning tree can be found in polynomial time (see Chapter 11) as well as an Euler tour (see Appendix A).

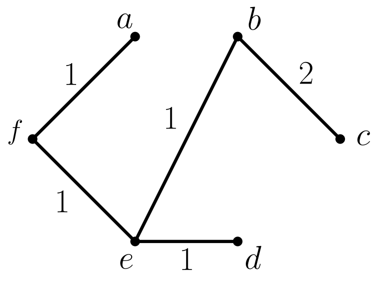
We now show that the algorithm always outputs a Hamilton cycle. Since G is a complete graph, it is connected and therefore has a minimum spanning tree. Replacing each edge by two parallel edges gives a multigraph M in which all degrees are even. Hence, there exists an Euler tour in M. This Euler tour visits all vertices at least one since M is connected. If we follow the tour in G but skip vertices that have already been visited (which is possible because G is complete) we get a Hamilton cycle.

It remains to find the approximation ratio of the algorithm. Let C^* be a Hamilton cycle with length $OPT(G, \ell)$. Deleting a single edge from C^* gives a spanning tree T' of G, which has total length at most $OPT(G, \ell)$. Therefore, the minimum spanning tree T found by the algorithm also has length at most $OPT(G, \ell)$. Since M is obtained from T by doubling each edge, it has total length at most $2OPT(G, \ell)$, and so also the Euler tour W has at most this length. The cycle C is obtained by skipping vertices and therefore, by the triangle inequality, C also has length at most $2OPT(G, \ell)$. Therefore, $ALG(G, \ell) \leq 2OPT(G, \ell)$, which shows that the approximation ratio is 2. This concludes the proof that the algorithm is a 2-approximation algorithm. $\square$

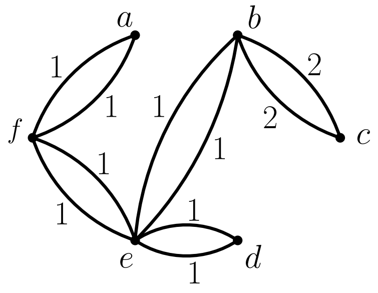
Example 17.1. Below is an example symmetric distance matrix satisfying the triangle inequality.

$$\begin{array}{c|cccccc} & a & b & c & d & e & f \\ a & & 2 & 4 & 3 & 2 & 1 \\ b & & & 2 & 2 & 1 & 2 \\ c & & & & 2 & 3 & 4 \\ d & & & & & 1 & 2 \\ e & & & & & & 1 \end{array}$$

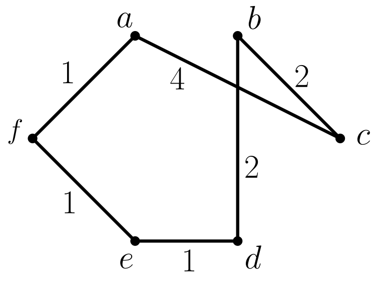
We can find a minimum spanning tree by, for example, using the Kruskal algorithm. We select all length-1 edges because this does not create a cycle. Then there are two length-2 edges that do not create a cycle. We choose one of them arbitrarily. This gives the following minimum spanning tree, with total length 6.



Doubling each edge of the minimum spanning tree gives the following multigraph, with total length 12.



An Euler tour in the multigraph is, for example, $(a, f, e, d, e, b, c, b, e, f, a)$, which also has length 12. Skipping already-visited vertices gives the Hamilton cycle (a, f, e, d, b, c, a), which has length 11.



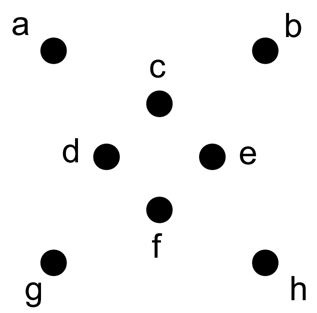
□

Finally, to find an Euler tour in Algorithm 9, we do not need to use the method described in the proof of Theorem A.1 but can use a simpler algorithm described in the following lemma (see Exercise 17.6).

Lemma 17.1. If M is a multigraph obtained by replacing each edge of a tree by two parallel edges, then an Euler tour of M can be obtained as follows. Start in an arbitrary vertex and repeat the following until all edges have been used. If there is a neighbouring vertex that has not been visited yet, walk to an arbitrary such neighbour. Otherwise, walk to an arbitrary neighbour that has already been visited over an edge that has not been used yet.

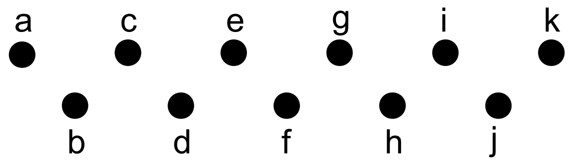
17.4 Exercises

Exercise 17.1. Find a TSP tour through the points below using the tree algorithm. The distances between the points are given in the table below and satisfy the triangle inequality.



$$\begin{array}{c|cccccccc} & a & b & c & d & e & f & g & h \\ \hline a & & 4 & 3 & 3 & 5 & 5 & 4 & 8 \\ b & & & 3 & 5 & 3 & 5 & 8 & 4 \\ c & & & & 2 & 2 & 2 & 5 & 5 \\ d & & & & & 2 & 2 & 3 & 5 \\ e & & & & & & 2 & 5 & 3 \\ f & & & & & & & 3 & 3 \\ g & & & & & & & & 4 \end{array}$$

Exercise 17.2. Find a TSP tour through the points below using the tree algorithm. The distances between the vertices are the Euclidian distances. Find an optimal tour by inspection and calculate the ratio ALG/OPT.



Exercise 17.3. Consider the following problem.

BIN PACKING
Given: n items with sizes $a_1, a_2, \ldots, a_n \leq 1$.
Find: the minimum number of size-1 bins that are needed to pack the n items?

Consider the following algorithm:

FIRST FIT

- Process the items in an arbitrary order.
- Add each item to the first bin in which it still fits. If the item does not fit in any bin, open a new bin.

Show that this FIRST FIT algorithm is a 2-approximation algorithm for BIN PACKING.

Exercise 17.4. Consider the knapsack problem:

$$z = \max \sum_{j=1}^{n} c_j x_j, \text{ s.t.} \sum_{j=1}^{n} a_j x_j \leq b, \ x \in \mathbb{Z}_{\geq 0}\,.$$

Assume that:

$$\begin{aligned} &a_j \le b, \quad j = 1, \dots, n \\ &a_j \in \mathbb{Z}_{\ge 0}, \quad j = 1, \dots, n \\ &b \in \mathbb{Z}_{\ge 0} \\ &\frac{c_1}{a_1} \ge \frac{c_j}{a_j}, \quad j = 2, \dots, n\,. \end{aligned}$$

The "greedy solution" for this problem is $(x^G)^T = (\lfloor (b/a_1) \rfloor, 0, \dots, 0)$ with value $z^G = c_1 \lfloor (b/a_1) \rfloor$. Show that $z^G \ge (1/2)z$.

Hint: the optimal solution of the LP-relaxation, under the given assumptions, is: $(x_{LP})^T = ((b/a_1), 0, \dots, 0)$ with value $z_{LP} = c_1(b/a_1)$. In addition, we know that $z^G \le z \le z_{LP}$, and hence that $z^G/z \ge z^G/z_{LP}$. Now show that $z^G/z_{LP} \ge 1/2$.

Exercise 17.5. Consider the VERTEX COVER problem in a graph with a weight c_v for each vertex $v \in V$. The LP-relaxation of this problem is given by

$$\begin{aligned} z_{LP} = \min \quad & \sum_{v \in V} c_v x_v \\ s.t. \quad & x_u + x_v \ge 1 && \forall \{u, v\} \in E \\ & x_v \ge 0 && \forall v \in V \end{aligned}$$

The corresponding dual problem is:

$$\begin{aligned} z_D = \max \quad & \sum_{e \in E} y_e \\ s.t. \quad & \sum_{e \in \delta(v)} y_e \le c_v && \forall v \in V && (*) \\ & y_e \ge 0 && \forall e \in E, \end{aligned}$$

with $\delta(v)$ the set of edges that are incident to v. Consider the following algorithm for VERTEX COVER:

```
1 Input: a graph $G = (V, E)$ with a weight $c_v$ for each $v \in V$
2 Output: a vertex cover of $G$
3 Initialisation: $y_e := 0, \forall e \in E, E' := E, C := \emptyset$;
4 while $E' \neq \emptyset$ do
5     choose an $e \in E'$ arbitrarily;
6     Increase $y_e$ until (*) is satisfied with equality for an endpoint of $e$, say $v$;
7     $C := C \cup \{v\}, E' := E' \setminus \delta(v)$;
8 end
9 return $C$
```

a) Prove that this algorithm indeed returns a vertex cover.

b) Prove that this algorithm is a 2-approximation algorithm for VERTEX COVER.

Hints:

$$\sum_{e \in \delta(v)} y_e = c_v \quad \forall v \in C$$

$$z_{ALG}(C) = \sum_{v \in C} c_v = \sum_{v \in C} (\sum_{e \in \delta(v)} y_e)$$

Use the weak duality theorem.

Exercise 17.6. Prove Lemma 17.1.

Part VI

Nonlinear Programming

Chapter 18

Some Basic Definitions and Results

All topics treated so far have been related to linear and integer linear optimization. The field of optimization is, of course, much broader than that and includes, among others, stochastic, robust, and non-linear optimization, and between many of the topics there is an active interface. In this syllabus, we cannot treat all topics, but we will give a very brief introduction to non-linear optimization. The interested reader is referred to Bazaraa et al. [1], Fletcher [6], Nash and Sofer [7], and Peressini et al. [10] for more on this topic.

We study problems of the following form:

$$\begin{aligned} &\min f(\boldsymbol{x}) \\ &\text{s.t. } g_i(\boldsymbol{x}) \quad \substack{\geq \\ = \\ \leq} \quad 0, \quad i = 1, \dots, m\,, \end{aligned}$$

where the sign $\substack{\geq \\ = \\ \leq}$ symbolizes $\geq$, or $=$, or $\leq$. In linear optimization the functions f and g_i, $i = 1, \dots, m$ are linear.

In linear optimization, we know that if an instance has an optimal solution, then it has an optimal solution in an extreme point of the feasible solution set, which is a polyhedron. In nonlinear optimization, the optimum may well be an interior point of the set of feasible solutions. To understand the algorithms that will be presented in the following chapters we need to review some basic results from calculus and analysis. We start with results for functions of one variable in Section 18.1, and proceed to the multi-variable case in Section 18.2. After introducing the basic results we proceed to so-called coercive functions in Section 18.3 and to convex and concave functions in Section 18.4.

18.1 Functions of one variable

The result that we will build on through this chapter is the following

Theorem 18.1 (Taylor's formula)**.** Suppose that $f(x)$, $f'(x)$, and $f''(x)$ exist on the closed interval $[a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$. If x^* and x are any two different points on the interval $[a, b]$, then there exists a point z strictly between x^* and x such that

$$f(x) = f(x^*) + f'(x^*)(x - x^*) + \frac{f''(z)}{2}(x - x^*)^2\,. \tag{18.1}$$

Proof. □ □

Notice that

$$f(x^*) + f'(x^*)(x - x^*) + \frac{f''(x^*)}{2}(x - x^*)^2$$

is an approximation of $f(x)$ when x is close to x^*.

Example 18.1. Consider the function $f(x) = 3x^4 - 4x^3 + 1$, see Figure 18.1. We have

$$\begin{aligned} f'(x) &= 12x^3 - 12x^2 = 12x^2(x-1) \\ f''(x) &= 12x(3x-2)\,. \end{aligned}$$

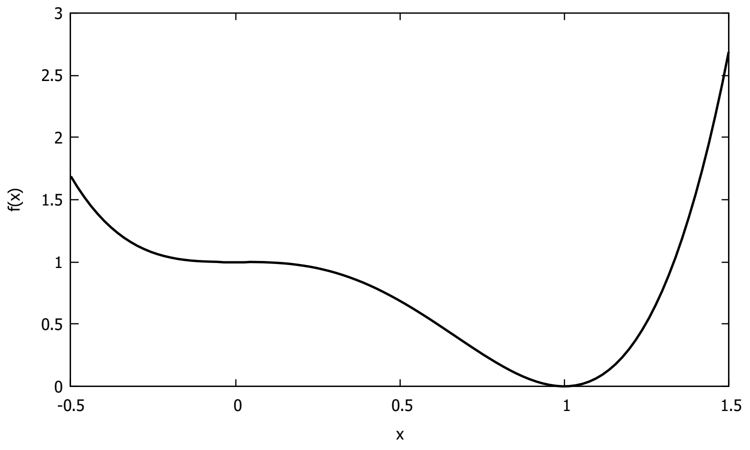


Figure 18.1: The function $f(x) = 3x^4 - 4x^3 + 1$.

Consider the interval $[a, b] = [-1/2,\ 3/2]$ and the points $x = 0,\ x^* = 3/2$. Then we can compute that the point $z \approx 1.088$ is the point z that satisfies Equation (18.1) in Theorem 18.1. $\square$

Instead of taking "arbitrary" points $x,\ x^*$ on an interval, as we did in Example 18.1, it is more interesting to consider x^* as a point for which $f'(x^*) = 0$. As you recall from calculus, such a point is a *critical point* of the function in which we may be able to find a (local) maximum or minimum. Before considering Taylor's formula in the case that x^* is a critical point, we introduce some basic definitions.

Definition 18.1. Suppose $f(x)$ is a real-valued function defined on some interval I. A point x^* in I is:

(a) a *global minimizer* for $f(x)$ on I if $f(x^*) \le f(x)$ for all $x \in I$.

(b) a *strict global minimizer* for $f(x)$ on I if $f(x^*) < f(x)$ for all $x \in I,\ x \ne x^*$.

(c) a *local minimizer* for $f(x)$ if there exists a number $\delta > 0$ such that $f(x^*) \le f(x)$ for all $x \in I$ that satisfy $x^* - \delta < x < x^* + \delta$.

(d) a *strict local minimizer* for $f(x)$ if there exists a number $\delta > 0$ such that $f(x^*) < f(x)$ for all $x \in I$ that satisfy $x^* - \delta < x < x^* + \delta$ and $x \ne x^*$.

(e) a *critical point* of $f(x)$ if $f'(x^*)$ exists and $f'(x^*) = 0$.

The following definition tells us the characteristics of a critical point of a function in relation to the value of the second derivative of that function.

Definition 18.2. Suppose that $f(x),\ f'(x)$, and $f''(x)$ are all continuous on an interval I, and that x^* is a critical point of $f(x)$.

(a) If $f''(x) \ge 0$ for all $x \in I$, then x^* is a *global minimizer* of $f(x)$ on I.

(b) If $f''(x) > 0$ for all $x \in I$ such that $x \neq x^*$, then x^* is a *strict global minimizer* of $f(x)$ on I.

(c) If $f''(x^*) > 0$, then x^* is a *strict local minimizer* of $f(x)$.

Note that these conditions are sufficient but not necessary. In other words, a critical point x^* can be a (strict) global minimizer, even if $f''(x)$ is not (strictly) greater than zero for all $x \in I$ (such that $x \neq x^*$); and a critical point can be a strict local minimizer even when $f''(x^*) \not> 0$.

Let us return to Taylor's formula in Theorem 18.1, and let us now take $x^* \in I$ to be a critical point of $f(x)$, i.e. $f'(x^*) = 0$. We repeat Equation (18.1) here for the purpose of readability.

$$f(x) = f(x^*) + f'(x^*)(x - x^*) + \frac{f''(z)}{2}(x - x^*)^2 . \tag{18.1}$$

If $f''(x) \geq 0$ for all $x \in I$, then Taylor's formula says that

$$f(x) = f(x^*) + 0 + \text{ nonnegative number},$$

in which case we can conclude that x^* is a minimum of $f(x)$ on I. This gives a nice relationship with the second derivative test.

Example 18.2. Consider again the function $f(x) = 3x^4 - 4x^3 + 1$, as depicted in Figure 18.2, and choose $x^* = 1$. $f'(1) = 0$ and $f''(1) = 12 > 0$, so x^* is a strict local minimum, and since $f''(x) \geq 0$ for all $x \geq 2/3$ we know that $x^* = 1$ is a minimizer of $f(x)$ on the interval $[2/3, +\infty)$. It is also interesting to note that $f'(x) = 0$ also for $x = 0$. If we evaluate the second derivative at $x = 0$, we see that $f''(0) = 0$, so we do not know the characteristics of $x = 0$ without investigating f around that point. $f(0) = 1$, and for small $\varepsilon > 0$, $f(\varepsilon) < 1$ and $f(-\varepsilon) > 1$, so $x = 0$ is a so-called *inflection point* for $f(x)$. □

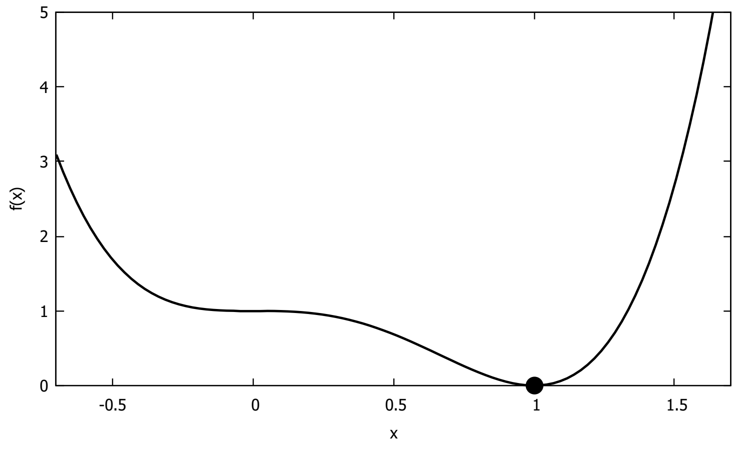


Figure 18.2: The function $f(x) = 3x^4 - 4x^3 + 1$ and the critical point $x^* = 1$.

18.2 Gradient, Hessian, minimizers, and maximizers of functions of several variables

The definitions presented in the previous section generalize naturally to the multi-variable case. Instead of the derivative of the function $f(x)$, we now consider the gradient of the function $f(\boldsymbol{x})$, where

$\boldsymbol{x}$, as in the previous parts of this syllabus, is a column vector

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}.$$

Let

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

be the *gradient* of $f(\boldsymbol{x})$ and $\partial f/\partial x_j$ the partial derivative of f with respect to x_j. The second derivative $f''(x)$ generalizes to the *Hessian* of $f(\boldsymbol{x})$,

$$Hf(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_n} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n x_1} & \frac{\partial^2 f}{\partial x_n x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

where $\partial^2 f/\partial x_i x_j$ are the second partial derivatives. Recall that H is symmetric if all second partial derivatives of f are continuous.

Example 18.3. Consider the function $f(\boldsymbol{x}) = x_1^3 - 12x_1x_2 + 8x_2^3$. The gradient of $f(\boldsymbol{x})$ is

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} 3x_1^2 - 12x_2 \\ -12x_1 + 24x_2^2 \end{pmatrix}.$$

The Hessian of $f(\boldsymbol{x})$ is

$$Hf(\boldsymbol{x}) = \begin{pmatrix} 6x_1 & -12 \\ -12 & 48x_2 \end{pmatrix}.$$

□

Let $\|\boldsymbol{x}\|$ denote the Euclidean norm of the vector $\boldsymbol{x}$, i.e., $\|\boldsymbol{x}\| = (\sum_{j=1}^n x_j^2)^{1/2}$. The (open) *ball* centered at $\boldsymbol{x}$ of radius r, is the set

$$B(\boldsymbol{x},\ r) = \{\boldsymbol{y} \in \mathbb{R}^n \mid \|\boldsymbol{y} - \boldsymbol{x}\| < r\}.$$

A point $\boldsymbol{x}$ in a subset D of $\mathbb{R}^n$ is an *interior point* of D if there is an $r > 0$ such that the ball $B(\boldsymbol{x},\ r)$ is contained in D.

Now we are ready for the definition that generalizes Definition 18.1

Definition 18.3. Suppose that $f(\boldsymbol{x})$ is a real-valued function defined on a subset $D \subseteq \mathbb{R}^n$. A point $\boldsymbol{x}^*$ is a

- *global minimizer* for $f(\boldsymbol{x})$ on D if $f(\boldsymbol{x}^*) \le f(\boldsymbol{x})$ for all $\boldsymbol{x} \in D$,
- *strict global minimizer* for $f(\boldsymbol{x})$ on D if $f(\boldsymbol{x}^*) < f(\boldsymbol{x})$ for all $\boldsymbol{x} \in D,\ \boldsymbol{x} \neq \boldsymbol{x}^*$,
- *local minimizer* for $f(\boldsymbol{x})$ if there exists a number $\delta > 0$ such that $f(\boldsymbol{x}^*) \le f(\boldsymbol{x})$ for all $\boldsymbol{x} \in D$ for which $\boldsymbol{x} \in B(\boldsymbol{x}^*,\ \delta)$,
- *critical point* for $f(\boldsymbol{x})$ if the first partial derivatives of $f(\boldsymbol{x})$ exist at $\boldsymbol{x}^*$ and $\frac{\partial f}{\partial x_j}(\boldsymbol{x}^*) = 0$ for $j = 1, \ldots, n$.

Example 18.3, cont. Let us find the critical points of $f(\boldsymbol{x}) = x_1^3 - 12x_1x_2 + 8x_2^3$. To do that we need to solve the equations $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$:

$$\begin{aligned} 3x_1^2 - 12x_2 &= 0 \\ -12x_1 + 24x_2^2 &= 0\,. \end{aligned}$$

From the second equation we obtain $x_1 = 2x_2^2$, which we can substitute in the first equation, from which we then can deduce that $x_2(x_2^3 - 1) = 0$. This results in the critical points

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 2 \\ 1 \end{pmatrix}.$$

□

Theorem 18.2 (Taylor's formula in the multi-variable case)**.** Suppose that $\boldsymbol{x}^*$, $\boldsymbol{x}$ are points in $\mathbb{R}^n$ and that $f(\boldsymbol{x})$ is a function of n variables with continuous first and second partial derivatives on some open set containing the line segment $[\boldsymbol{x}^*,\ \boldsymbol{x}] = \{\boldsymbol{v} \in \mathbb{R}^n \mid \boldsymbol{v} = \boldsymbol{x}^* + t(\boldsymbol{x} - \boldsymbol{x}^*),\ 0 \le t \le 1)\}$ joining $\boldsymbol{x}^*$ and $\boldsymbol{x}$. Then, there exists a point $\boldsymbol{z} \in [\boldsymbol{x}^*,\ \boldsymbol{x}]$ such that

$$f(\boldsymbol{x}) = f(\boldsymbol{x}^*) + \nabla f(\boldsymbol{x}^*)^\mathsf{T}(\boldsymbol{x} - \boldsymbol{x}^*) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^*)^\mathsf{T} Hf(\boldsymbol{z})(\boldsymbol{x} - \boldsymbol{x}^*)\,.$$

Let A be any $n \times n$ symmetric matrix. Then, $Q_A(\boldsymbol{y}) = \boldsymbol{y}^\mathsf{T} A\boldsymbol{y}$ for $\boldsymbol{y} \in \mathbb{R}^n$ is called the *quadratic form* associated with A. The quadratic form $Q_{Hf(\boldsymbol{z})}$ associated with $Hf(\boldsymbol{z})$ evaluated at $\boldsymbol{x} - \boldsymbol{x}^*$ is

$$Q_{Hf(\boldsymbol{z})}(\boldsymbol{x} - \boldsymbol{x}^*) = (\boldsymbol{x} - \boldsymbol{x}^*)^\mathsf{T} Hf(\boldsymbol{z})(\boldsymbol{x} - \boldsymbol{x}^*)\,,$$

which occurs in Taylor's formula.

Definition 18.4. Suppose that A is an $n \times n$ symmetric matrix and that $Q_A(\boldsymbol{y}) = \boldsymbol{y}^\mathsf{T} A\boldsymbol{y}$ is the quadratic form associated with A. Then A and Q_A are called

- *positive semidefinite* if $Q_A(\boldsymbol{y}) \ge \boldsymbol{0}$ for all $\boldsymbol{y} \in \mathbb{R}^n$.
- *positive definite* if $Q_A(\boldsymbol{y}) > \boldsymbol{0}$ for all $\boldsymbol{y} \in \mathbb{R}^n$, $\boldsymbol{y} \ne \boldsymbol{0}$.
- *negative semidefinite* if $Q_A(\boldsymbol{y}) \le \boldsymbol{0}$ for all $\boldsymbol{y} \in \mathbb{R}^n$.
- *negative definite* if $Q_A(\boldsymbol{y}) < \boldsymbol{0}$ for all $\boldsymbol{y} \in \mathbb{R}^n$, $\boldsymbol{y} \ne \boldsymbol{0}$.
- *indefinite* if $Q_A(\boldsymbol{y}) > \boldsymbol{0}$ for some $\boldsymbol{y} \in \mathbb{R}^n$ and $Q_A(\boldsymbol{y}) < \boldsymbol{0}$ for other $\boldsymbol{y} \in \mathbb{R}^n$.

Theorem 18.3. Suppose that $\boldsymbol{x}^*$ is a critical point of a function $f(\boldsymbol{x})$ with continuous first and second partial derivatives on $\mathbb{R}^n$, and that $Hf(\boldsymbol{x})$ is the Hessian of $f(\boldsymbol{x})$. Then, $\boldsymbol{x}^*$ is a

- global minimizer for $f(\boldsymbol{x})$ if $Hf(\boldsymbol{x})$ is positive semidefinite on $\mathbb{R}^n$.
- strict global minimizer for $f(\boldsymbol{x})$ if $Hf(\boldsymbol{x})$ is positive definite on $\mathbb{R}^n$.
- global maximizer for $f(\boldsymbol{x})$ if $Hf(\boldsymbol{x})$ is negative semidefinite on $\mathbb{R}^n$.
- strict global maximizer for $f(\boldsymbol{x})$ if $Hf(\boldsymbol{x})$ is negative definite on $\mathbb{R}^n$.

Even though it is always interesting to know the global minimizer (or maximizer) of a function, it is also relevant to know whether or not a point is a *local* minimizer. In fact, if we minimize a function *over a feasible set*, the optimum can very well be a local minimizer of the function, as the global minimum of the function may be outside of the feasible set.

Theorem 18.4. Suppose that $f(\boldsymbol{x})$ is a function with continuous first and second partial derivatives on some set $D \subset \mathbb{R}^n$. Moreover, suppose that $\boldsymbol{x}^*$ is an interior point of D and that $\boldsymbol{x}^*$ is a critical point of $f(\boldsymbol{x})$. Then, $\boldsymbol{x}^*$ is a

- strict local minimizer of $f(\boldsymbol{x})$ if $Hf(\boldsymbol{x}^*)$ is positive definite,
- strict local maximizer of $f(\boldsymbol{x})$ if $Hf(\boldsymbol{x}^*)$ is negative definite.

So, to determine the characteristic of a critical point we need to check whether the Hessian is positive or negative (semi)definite or indefinite. To check if a symmetric matrix A is positive or negative (semi)definite or indefinite, we can either check the eigenvalues of A, or we can check the sign of the principle minors of A.

Definition 18.5. Suppose A is an $n \times n$ symmetric matrix. Define Δ_k to be the determinant of the upper left-hand corner $k \times k$ submatrix of A for $1 \leq k \leq n$. The determinant Δ_k is called the kth *principal minor* of A.

Theorem 18.5. If A is an $n \times n$ symmetric matrix, and if Δ_k is the kth principal minor of A for $1 \leq k \leq n$, then A is

- positive definite if and only if $\Delta_k > 0$ for all $1 \leq k \leq n$,
- negative definite if and only if $(-1)^k \Delta_k > 0$ for all $1 \leq k \leq n$.

Notice that it is *not true* in general that if A is an $n \times n$ symmetric matrix, then A is positive semidefinite if and only if $\Delta_k \geq 0$ for all k. However, the following is true:

- If A is an $n \times n$ symmetric matrix and $\Delta_k > 0,\ k = 1, \ldots, n-1$, and $\Delta_n = 0$, then A is positive semidefinite.
- If A is an $n \times n$ symmetric matrix and $(-1)^k \Delta_k > 0,\ k = 1, \ldots, n-1$, and $\Delta_n = 0$, then A is negative semidefinite.

For eigenvalues, the following holds.

Theorem 18.6. If A is an $n \times n$ symmetric matrix, then A is

- positive definite (negative definite) if and only if the eigenvalues of A are positive (negative).
- positive semidefinite (negative semidefinite) if and only if the eigenvalues of A are nonnegative (nonpositive).
- indefinite if and only if A has at least one positive eigenvalue and one negative eigenvalue.

Example 18.4. Determine the value of the eigenvalues of the matrix

$$A = \begin{pmatrix} 2 & -4 & 0 \\ -4 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}.$$

Solve $|A - \lambda I| = \mathbf{0}$.

$$\begin{vmatrix} 2-\lambda & -4 & 0 \\ -4 & 2-\lambda & 0 \\ 0 & 0 & 2-\lambda \end{vmatrix} = \mathbf{0}$$

Expand by column 3: $(2-\lambda)((2-\lambda)(2-\lambda)-16) = (2-\lambda)(\lambda^2-4\lambda-12) = 0$ gives $\lambda = 2,\ \lambda = 6,\ \lambda = -2$. From Theorem 18.6 we can conclude that the matrix A is indefinite.

Determine the values of the principal minors of the matrix

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

$$\Delta_1 = |2| = 2, \quad \Delta_2 = \begin{vmatrix} 2 & -1 \\ -1 & 2 \end{vmatrix} = 4 - 1 = 3, \quad \Delta_3 = |B| = 4\,.$$

From Theorem 18.5 we conclude that B is positive definite. □

Example 18.3, cont. Let us examine the character of the critical point $\boldsymbol{x}^\mathsf{T} = (2,1)$ of $f(\boldsymbol{x}) = x_1^3 - 12x_1x_2 + 8x_2^3$. Recall that the Hessian of $f(\boldsymbol{x})$ is: The Hessian of $f(\boldsymbol{x})$ is

$$Hf(\boldsymbol{x}) = \begin{pmatrix} 6x_1 & -12 \\ -12 & 48x_2 \end{pmatrix}.$$

Examining the Hessian of $f(\boldsymbol{x})$ at the point $\boldsymbol{x}^\mathsf{T} = (2,1)$. The Hessian becomes

$$\begin{pmatrix} 12 & -12 \\ -12 & 48 \end{pmatrix}.$$

The principal minors are $\Delta_1 = 12$, $\Delta_2 = 12 \cdot 48 - (-12) \cdot (-12) = 432$. Since both Δ_1 and Δ_2 are positive, the Hessian evaluated at this point is positive definite and the critical point is a strict local minimizer. In fact, this point is a minimizer on the domain $D = \{\boldsymbol{x} \in \mathbb{R}^2 \mid x_1 > 0, x_1x_2 \geq 1/2\}$. □

From Theorem 18.3 we know that if $\boldsymbol{x}^*$ is a critical point and the Hessian is positive (negative) definite or semidefinite, then $\boldsymbol{x}^*$ is a global minimizer (maximizer). A relevant question is what we know about a critical point $\boldsymbol{x}^*$ in the case that H is indefinite. The following theorem gives an answer.

Theorem 18.7. If $f(\boldsymbol{x})$ is a function with continuous second partial derivatives in a set $D \subseteq \mathbb{R}^n$, if $\boldsymbol{x}^*$ is an interior point of D that is a critical point of $f(\boldsymbol{x})$, and if the Hessian $Hf(\boldsymbol{x}^*)$ is indefinite, then $\boldsymbol{x}^*$ is a *saddle point*, see Figure 18.3.

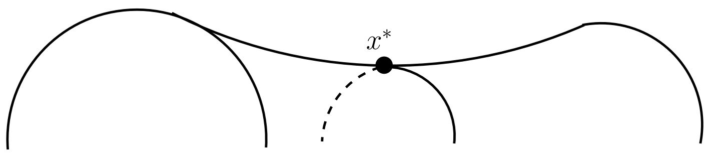


Figure 18.3: Saddle point.

Example 18.3, cont. Let us examine the character of the second critical point of $f(\boldsymbol{x}) = x_1^3 - 12x_1x_2 + 8x_2^3$, namely $\boldsymbol{x}^\mathsf{T} = (0,0)$. The Hessian becomes

$$\begin{pmatrix} 0 & -12 \\ -12 & 0 \end{pmatrix}.$$

Now, compute the eigenvalues of the Hessian:

$$\begin{vmatrix} -\lambda & -12 \\ -12 & -\lambda \end{vmatrix} = \mathbf{0}\,,$$

which yields $\lambda = \pm 12$. From Theorem 18.6 we can conclude that the Hessian is indefinite at $\boldsymbol{x}^\mathsf{T} = (0,0)$, so this point is a saddle point. □

18.3 Coercive functions

We know that if $f(\boldsymbol{x})$ has a critical point and if $Hf(\boldsymbol{x})$ is always positive definite, then the critical point is a global minimizer. What about global minimization for a function $f(\boldsymbol{x})$ in the case in which $Hf(\boldsymbol{x})$ is not known to be always positive definite?

Theorem 18.8. Let D be a closed bounded subset of $\mathbb{R}^n$. If $f(\boldsymbol{x})$ is a continuous function defined on D, then $f(\boldsymbol{x})$ has a global maximizer and a global minimizer on D.

We now consider a type of functions that is "easily" handled.

Definition 18.6. A continuous function $f(\boldsymbol{x})$ that is defined on all of $\mathbb{R}^n$ is called *coercive* if

$$\lim_{\|\boldsymbol{x}\|\to\infty} f(\boldsymbol{x}) = +\infty\,,$$

i.e., for any constant M, there must be a positive number R_M such that $f(\boldsymbol{x}) \geq M$ whenever $\|\boldsymbol{x}\| \geq R_M$.

Example 18.5. Consider the function

$$f(x_1,x_2) = x_1^4 + x_2^4 - 3x_1x_2 = (x_1^4+x_2^4)\left(1 - \frac{3x_1x_2}{x_1^4+x_2^4}\right)\,.$$

If $\|\boldsymbol{x}\|$ is large, then $\frac{3x_1x_2}{x_1^4+x_2^4}$ is very small, hence

$$\lim_{\|\boldsymbol{x}\|\to\infty} f(\boldsymbol{x}) = +\infty\,,$$

Thus, $f(\boldsymbol{x})$ is coercive □

Notice that for $f(\boldsymbol{x})$ to be coercive, it is *not* sufficient that $f(\boldsymbol{x}) \to \infty$ as each coordinate tends to infinity; $f(\boldsymbol{x})$ must become infinite along any path for which $\|\boldsymbol{x}\|$ becomes infinity.

Theorem 18.9. Let $f(\boldsymbol{x})$ be a continuous function defined on all $\mathbb{R}^n$. If $f(\boldsymbol{x})$ is coercive, then $f(\boldsymbol{x})$ has at least one global minimizer. If, in addition, the first partial derivatives of $f(\boldsymbol{x})$ exist on all of $\mathbb{R}^n$, then these global minimizers can be found among the critical points of $f(\boldsymbol{x})$.

Example 18.6. Consider the function $f(x_1,x_2) = x_1^4 - 4x_1x_2 + x_2^4$. The gradient and Hessian are:

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} 4x_1^3 - 4x_2 \\ -4x_1 + 4x_2^3 \end{pmatrix}\,, \quad Hf(\boldsymbol{x}) = \begin{pmatrix} 12x_1^2 & -4 \\ -4 & 12x_2^2 \end{pmatrix}$$

To determine the critical points, we solve the system of equations $\nabla f(\boldsymbol{x}) =$:

$$4x_1^3 - 4x_2 = 0 \tag{18.2}$$
$$-4x_1 + 4x_2^3 = 0 \tag{18.3}$$

From Equation (18.2) we obtain $x_2 = x_1^3$, and from Equation (18.3) we obtain $x_1 = x_2^3$. Combining the two yields $x_1 = x_1^9$, or $x_1(x_1^8 - 1) = 0$. Evaluating yields

$$\boldsymbol{x}^1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}\,, \quad \boldsymbol{x}^2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}\,, \quad \boldsymbol{x}^3 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$$

The Hessian is not always positive definite as for instance $Hf(\boldsymbol{x})(\frac{1}{2},\ \frac{1}{2}) < 0$. But, $f(\boldsymbol{x})$ is coercive, which we can show in a very similar ways as in Example 18.5. Evaluating $f(\boldsymbol{x})$ at the three critical points gives

$$\begin{aligned} f(0,0) &= 0 \\ f(1,1) &= -2 \\ f(-1,-1) &= -2 \end{aligned}$$

Therefore, the points $\boldsymbol{x}^2$ and $\boldsymbol{x}^3$ are global minimizers. □

18.4 Convex and concave functions

In Chapter 3, we studied convex sets. Here we will focus on functions and will present results related to convex and concave functions.

Definition 18.7. Let $f(\boldsymbol{x})$ be a real-valued function defined over points $\boldsymbol{x} \in D \subseteq \mathbb{R}^n$, where D is either $\mathbb{R}^n$ or a convex subset of $\mathbb{R}^n$. Then, $f(\boldsymbol{x})$ is said to be a *convex* function if and only if for any points $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ in D and $0 \le \alpha \le 1$, we have

$$f(\alpha\boldsymbol{x}^1 + (1-\alpha)\boldsymbol{x}^2) \le \alpha f(\boldsymbol{x}^1) + (1-\alpha)f(\boldsymbol{x}^2)\,. \tag{18.4}$$

The function $f(\boldsymbol{x})$ is *strictly convex* if the inequality in (18.4) holds with strict inequality for $0 < \alpha < 1$.

What the definition is saying is that when two points on the graph of the function are joined by a chord, then the function itself lies underneath the chord on the interval joining the points $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$, see Figure 18.4.

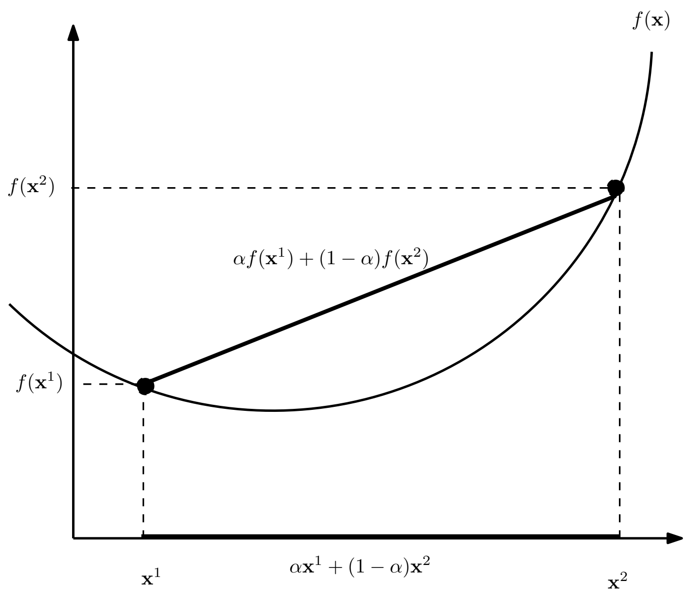

$f(\mathbf{x})$
$f(\mathbf{x}^2)$
$\alpha f(\mathbf{x}^1) + (1-\alpha)f(\mathbf{x}^2)$
$f(\mathbf{x}^1)$
$\mathbf{x}^1$
$\alpha\mathbf{x}^1 + (1-\alpha)\mathbf{x}^2$
$\mathbf{x}^2$

Figure 18.4: A convex function.

Notice that $\alpha\boldsymbol{x}^1 + (1-\alpha)\boldsymbol{x}^2$ in Definition 18.7 is a convex combination of the points $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ and that $\alpha f(\boldsymbol{x}^1) + (1-\alpha)f(\boldsymbol{x}^2)$ is a convex combination of $f(\boldsymbol{x}^1)$ and $f(\boldsymbol{x}^2)$, see Chapter 3.

We have the following similar definition of a concave function.

Definition 18.8. Let $g(\boldsymbol{x})$ be a real-valued function defined over points $\boldsymbol{x} \in D \subseteq \mathbb{R}^n$, where D is either $\mathbb{R}^n$ or a convex subset of $\mathbb{R}^n$. Then, $g(\boldsymbol{x})$ is said to be a *concave* function if and only if for any points $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ in D and $0 \le \alpha \le 1$, we have

$$g(\alpha\boldsymbol{x}^1 + (1-\alpha)\boldsymbol{x}^2) \ge \alpha g(\boldsymbol{x}) + (1-\alpha)g(\boldsymbol{x})\,. \tag{18.5}$$

The function $g(\boldsymbol{x})$ is *strictly concave* if the inequality in (18.5) holds with strict inequality for $0 < \alpha < 1$.

Theorem 18.10.

- If $f_1(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x})$ are convex functions, then $f(\boldsymbol{x}) = f_1(\boldsymbol{x}) + f_2(\boldsymbol{x}) + \cdots f_k(\boldsymbol{x})$ is convex. Moreover, if at least one of the functions $f_i(\boldsymbol{x})$ is strictly convex, then $f(\boldsymbol{x})$ is strictly convex.

- If $f(\boldsymbol{x})$ is (strictly) convex on a convex set $D \subseteq \mathbb{R}^n$ and if $\alpha > 0$, then $\alpha f(\boldsymbol{x})$ is (strictly) convex.
- If $f(\boldsymbol{x})$ is a (strictly) convex function defined on a convex set $D \subseteq \mathbb{R}^n$ and if $g(\boldsymbol{y})$ is a (strictly) increasing convex function defined on the range of $f(\boldsymbol{x})$ in $\mathbb{R}^n$, then the composite function $g(f(\boldsymbol{x}))$ is (strictly) convex.
- If $f(\boldsymbol{x})$ is a concave function, then $-f(\boldsymbol{x})$ is a convex function.

Example 18.7. An illustration of Theorem 18.10, first bullet point is: $f_1(x) = x^2$ and $f_2(x) = x^4$ are convex functions. Hence, the function $f_3(x) = f_1(x) + f_2(x) = x^2 + x^4$ is convex. Third bulletpoint: $g(y) = e^y$ and $f(x) = x^2$ are convex. Therefore $g(f(x)) = e^{x^2}$ is convex. □

Theorem 18.11. Any local minimizer of a convex function $f(\boldsymbol{x})$ defined on a convex subset C of $\mathbb{R}^n$ is also a global minimizer. Any local minimizer of a strictly convex function $f(\boldsymbol{x})$ defined on a convex subset C in $\mathbb{R}^n$ is the unique strict global minimizer of $f(\boldsymbol{x})$ on C.

Proof. Suppose that $\boldsymbol{x}^*$ is a local minimizer for the convex function $f(\boldsymbol{x})$ on C. Then there is a positive number r such that $f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*)$ whenever $\boldsymbol{x} \in C$ and $\|\boldsymbol{x} - \boldsymbol{x}^*\| < r$.

Given any $\boldsymbol{y} \in C$, we want to show that $f(\boldsymbol{y}) \geq f(\boldsymbol{x}^*)$. Select λ such that $0 < \lambda < 1$, and such that

$$\|\lambda \boldsymbol{y} + (1-\lambda)\boldsymbol{x}^* - \boldsymbol{x}^*\| < r\,.$$

Notice that since C is convex, $\lambda \boldsymbol{y} + (1-\lambda)\boldsymbol{x}^* \in C$, see also Figure 18.5. Accordingly,

$$f(\boldsymbol{x}^*) \leq f(\lambda \boldsymbol{y} + (1-\lambda)\boldsymbol{x}^*) \leq \lambda f(\boldsymbol{y}) + (1-\lambda) f(\boldsymbol{x}^*)\,, \tag{18.6}$$

where the first inequality follows from $\boldsymbol{x}^*$ being a local minimizer, and the second inequality from $f(\boldsymbol{x})$ being convex. The last inequality is strict if $\boldsymbol{y} \neq \boldsymbol{x}$ and $f(\boldsymbol{x})$ is strictly convex.

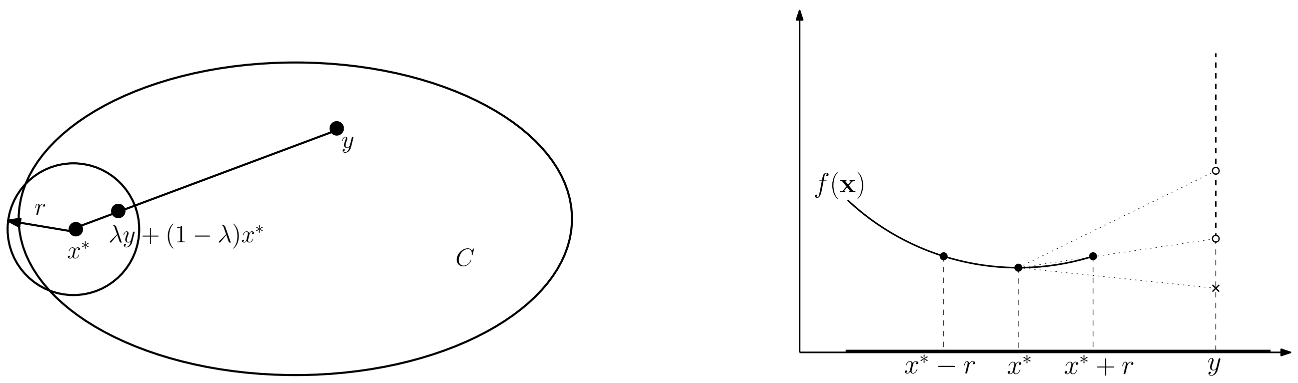


Figure 18.5: Left: The points $\boldsymbol{x}^*$, $\boldsymbol{y}$, and $\lambda \boldsymbol{y} + (1-\lambda)\boldsymbol{x}^*$. Right: The graph of $f(\boldsymbol{x})$ restricted to the line between $\boldsymbol{x}$ and $\boldsymbol{y}$. For the function to be convex $f(\boldsymbol{y})$ must be at or above the line through $f(\boldsymbol{x}^*)$ and $f(\boldsymbol{x}^* + r)$. Otherwise, the $f(\boldsymbol{x}^* + r)$ lies above the line between $f(\boldsymbol{x}^*)$ and $f(\boldsymbol{y})$.

Now, subtract $f(\boldsymbol{x}^*)$ from both the left-hand side and the right-hand side of (18.6). This gives

$$0 \leq \lambda f(\boldsymbol{y}) - \lambda f(\boldsymbol{x}^*)\,.$$

Dividing by λ gives $0 \leq f(\boldsymbol{y}) - f(\boldsymbol{x}^*)$, or equivalently

$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{y})\,,$$

where the inequality is strict if $f(\boldsymbol{x})$ is strictly convex and if $\boldsymbol{y} \neq \boldsymbol{x}^*$. □

Theorem 18.12. Suppose that $f(\boldsymbol{x})$ has continuous first partial derivatives on a convex set $D \subseteq \mathbb{R}^n$. Then

- the function $f(\boldsymbol{x})$ is convex on D if and only if
$$f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^\mathsf{T}(\boldsymbol{y} - \boldsymbol{x}) \le f(\boldsymbol{y})$$
for all $\boldsymbol{x},\ \boldsymbol{y} \in D$.
- the function $f(\boldsymbol{x})$ is strictly convex on D if and only if
$$f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^\mathsf{T}(\boldsymbol{y} - \boldsymbol{x}) < f(\boldsymbol{y})$$
for all $\boldsymbol{x},\ \boldsymbol{y} \in D$ with $\boldsymbol{x} \neq \boldsymbol{y}$.

Notice that $f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^\mathsf{T}(\boldsymbol{y} - \boldsymbol{x})$ is the tangent hyperplane to the graph of $f(\boldsymbol{x})$ at $\boldsymbol{x}$.

Corollary 18.1. If $f(\boldsymbol{x})$ is a convex function with continuous first partial derivatives on some convex set $D \subseteq \mathbb{R}^n$, then any critical point of $f(\boldsymbol{x})$ in D is a global minimizer of $f(\boldsymbol{x})$.

Theorem 18.13. Suppose that $f(\boldsymbol{x})$ has continuous second partial derivatives on some open convex set $D \subset \mathbb{R}^n$. If the Hessian $Hf(\boldsymbol{x})$ of $f(\boldsymbol{x})$ is positive semidefinite (positive definite) on D, then $f(\boldsymbol{x})$ is convex (strictly convex) on D.

Notice that if $f(\boldsymbol{x})$ is a convex function with continuous second partial derivatives on some convex set D in $\mathbb{R}^n$, then $Hf(\boldsymbol{x})$ is positive semidefinite on D. However, if $f(\boldsymbol{x})$ is strictly convex on D, $Hf(\boldsymbol{x})$ is *not* necessarily strictly positive definite.

18.5 Exercises

Exercise 18.1. Classify the following matrices according to whether they are positive or negative definite or semidefinite or indefinite:

(a)
$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{pmatrix}$$

(b)
$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & -2 \end{pmatrix}$$

(c)
$$\begin{pmatrix} 7 & 0 & 0 \\ 0 & -8 & 0 \\ 0 & 0 & 5 \end{pmatrix}$$

(d)
$$\begin{pmatrix} 2 & -4 & 0 \\ -4 & 8 & 0 \\ 0 & 0 & -3 \end{pmatrix}$$

Exercise 18.2. Give an example of a strictly convex function whose Hessian is not strictly positive definite. (Hint: There is a polynomial function of one variable that suffices.)

Exercise 18.3. Evaluate the Hessian to determine (if possible) the nature of the critical points of the following functions:

(a) $f(x_1, x_2) = x_1^3 + x_2^3 - 3x_1 - 12x_2 + 20.$

(b) $f(x_1, x_2, x_3) = 3x_1^2 + 2x_2^2 + 2x_3^2 + 2x_1x_2 + 2x_2x_3 + 2x_1x_3.$

(c) $f(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2 - 4x_1x_2.$

(d) $f(x_1, x_2) = x_1^4 + x_2^4 - x_1^2 - x_2^2 + 1.$

(e) $f(x_1, x_2) = 12x_1^3 - 36x_1x_2 - 2x_2^3 + 9x_2^2 - 72x_1 + 60x_2 + 5.$

Exercise 18.4. Find the global maximizers and minimizers, if they exist, for the following functions:

(a) $f(x_1, x_2) = x_1^2 - 4x_1 + 2x_2^2 + 7.$

(b) $f(x_1, x_2, x_3) = (2x_1 - x_2)^2 + (x_2 - x_3)^2 + (x_3 - 1)^2.$

Exercise 18.5. Which of the following functions are coercive?

(a) On $\mathbb{R}^3$, let $f(x_1, x_2, x_3) = x_1^3 + x_2^3 + x_3^3 - x_1x_2.$

(b) On $\mathbb{R}^3$, let $f(x_1, x_2, x_3) = x_1^4 + x_2^4 + x_3^2 - 3x_1x_2 - x_3.$

(c) On $\mathbb{R}^3$, let $f(x_1, x_2, x_3) = x_1^4 + x_2^4 + x_3^2 - 7x_1x_2x_3^2.$

(d) On $\mathbb{R}^3$, let $f(x_1, x_2, x_3) = x_1^4 + x_2^4 + x_3^2 - 2x_1x_2^2.$

Exercise 18.6. Taylor's formula.

(a) Write out in full Taylor's formula of $f(x, y) = 3x^2y + 2xy + x^2 + 2$ in $(x^*, y^*) = (0, 0)$.

(b) Which terms from $f(x, y)$ do you recognize?

(c) Conclude that it gives an approximation of $f(x, y)$ when (x, y) is close to (x^*, y^*)

Exercise 18.7. Indicate whether the following statements are true or false:

1. Each strictly convex function has at most one local minimum.
2. Suppose $f(\boldsymbol{x})$ has continuous second partial derivatives on an open convex set D of $\mathbb{R}^n$. If the Hessian $Hf(\boldsymbol{x})$ of $f(\boldsymbol{x})$ is strictly negative definite on D, then $f(\boldsymbol{x})$ has a unique global maximum.
3. There exists a coercive function $f(\boldsymbol{x})$ such that for each $M > 0$, $f(\boldsymbol{x})$ has a local minimum $(\boldsymbol{x}^*)$ with $||\boldsymbol{x}^*|| > M$.

Chapter 19

Iterative Methods for Unconstrained Optimization

By an iterative method, we mean a computational routine that produces a sequence $\{\boldsymbol{x}^{(k)}\}$ in $\mathbb{R}^n$ that we can expect to converge to a minimizer of a given function $f(\boldsymbol{x})$, under reasonably broad conditions on $f(\boldsymbol{x})$. The sequence $\{\boldsymbol{x}^{(k)}\}$ should be numerically reliable and not too costly to compute. A natural question to ask is why we need iterative methods. Why not simply solve the system of equations

$$\nabla f(\boldsymbol{x}) = \mathbf{0} \tag{19.1}$$

to identify the critical points? Usually, it is not practical, and often it is impossible, to locate the critical points of $f(\boldsymbol{x})$ by solving the system (19.1), and to analyze the Hessian $Hf(\boldsymbol{x})$.

In this chapter, we will present two classical iterative methods, namely Newton's method, and the method of Steepest Descent.

19.1 Newton's method

Suppose that $f(\boldsymbol{x})$ is a twice continuously differentiable real-valued function of n variables, and suppose that $\boldsymbol{x}^{(0)} \in \mathbb{R}^n$. Then the Newton's method sequence $\{\boldsymbol{x}^{(k)}\}$, with initial point $\boldsymbol{x}^{(0)}$, for minimizing $f(\boldsymbol{x})$, is defined by the following recurrence formula:

$$Hf(\boldsymbol{x}^{(k)})(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}) = -\nabla f(\boldsymbol{x}^{(k)}) \quad \text{for } k \geq 0\,, \tag{19.2}$$

or, equivalently, by

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - [Hf(\boldsymbol{x}^{(k)})]^{-1}\nabla f(\boldsymbol{x}^{(k)}) \quad \text{for } k \geq 0\,. \tag{19.3}$$

The motivation comes from Taylor's theorem that tells us that the function

$$f_k(\boldsymbol{x}) = f(\boldsymbol{x}^{(k)}) + \nabla f(\boldsymbol{x}^{(k)})^\mathsf{T}(\boldsymbol{x} - \boldsymbol{x}^{(k)}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(k)})^\mathsf{T} Hf(\boldsymbol{x}^{(k)})(\boldsymbol{x} - \boldsymbol{x}^{(k)})$$

is the quadratic function that best fits the function $f(\boldsymbol{x})$ at the point $\boldsymbol{x}^{(k)}$ in the sense that $f_k(\boldsymbol{x})$ and $f(\boldsymbol{x})$ as well as their first and second derivatives agree at $\boldsymbol{x}^{(k)}$.

We look for a minimizer $\boldsymbol{x}^*$ of $f(\boldsymbol{x})$, hence it is reasonable to choose $\boldsymbol{x}^{(k+1)}$ as the minimizer of $f_k(\boldsymbol{x})$. If $f_k(\boldsymbol{x})$ has a minimizer $\boldsymbol{y}^*$, then $\boldsymbol{y}^*$ is a critical point of $f_k(\boldsymbol{x})$, which yields the following system of equations:

$$\mathbf{0} = \nabla f_k(\boldsymbol{y}^*) = \nabla f_k(\boldsymbol{x}^{(k)}) + Hf(\boldsymbol{x}^{(k)})(\boldsymbol{y}^* - \boldsymbol{x}^{(k)})\,.$$

If $Hf(\boldsymbol{x})$ is positive definite at $\boldsymbol{x}^{(k)}$, then $f_k(\boldsymbol{x})$ is strictly convex and does have a strict global minimizer $\boldsymbol{y}^*$. The point $\boldsymbol{y}^*$ can then be found as the unique solution to

$$\nabla f_k(\boldsymbol{y}^*) = \mathbf{0}\,,$$

or equivalently to

$$Hf(\boldsymbol{x}^{(k)})(\boldsymbol{y}^* - \boldsymbol{x}^{(k)}) = -\nabla f_k(\boldsymbol{x}^{(k)}) \,,$$

which is the same expression as (19.2) with $\boldsymbol{y}^* = \boldsymbol{x}^{(k+1)}$.

Theorem 19.1. Suppose that $\{\boldsymbol{x}^{(k)}\}$ is the Newton Method sequence for minimizing a function $f(\boldsymbol{x})$. If the Hessian $Hf(\boldsymbol{x}^{(k)})$ of $f(\boldsymbol{x})$ at $\boldsymbol{x}^{(k)}$ is positive definite and if $\nabla f(\boldsymbol{x}^{(k)}) \neq \boldsymbol{0}$, then the direction

$$\boldsymbol{p}^{(k)} = -[Hf(\boldsymbol{x}^{(k)})]^{-1}\nabla f(\boldsymbol{x}^{(k)})$$

from $\boldsymbol{x}^{(k)}$ to $\boldsymbol{x}^{(k+1)}$ is a descent direction for $f(\boldsymbol{x})$ in the sense that there is an $\epsilon > 0$ such that

$$f(\boldsymbol{x}^{(k)} + t\boldsymbol{p}^{(k)}) < f(\boldsymbol{x}^{(k)})$$

for all t such that $0 < t < \epsilon$.

Example 19.1. Consider the function $f(x_1, x_2) = x_1^4 + 2x_1^2x_2^2 + x_2^4$. Compute $\boldsymbol{x}^{(1)}$ using Newton's Method starting at $\boldsymbol{x}^{(0)^\mathsf{T}} = (1, 1)$.

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} 4x_1^3 + 4x_1x_2^2 \\ 4x_1^2x_2 + 4x_2^3 \end{pmatrix}, \quad Hf(\boldsymbol{x}) = \begin{pmatrix} 12x_1^2 + 4x_2^2 & 8x_1x_2 \\ 8x_1x_2 & 4x_1^2 + 12x_2^2 \end{pmatrix}$$

$$\nabla f\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 8 \\ 8 \end{pmatrix}, \quad Hf\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 16 & 8 \\ 8 & 16 \end{pmatrix}$$

Equation (19.2) gives

$$\begin{aligned} 16(x_1 - 1) + 8(x_2 - 1) &= -8 \\ 8(x_1 - 1) + 16(x_2 - 1) &= -8 \,, \end{aligned}$$

or

$$\begin{aligned} 2x_1 + x_2 &= 2 \\ x_1 + 2x_2 &= 2 \,, \end{aligned}$$

which yields $x_1 = x_2 = 2/3$, so $\boldsymbol{x}^{(1)^\mathsf{T}} = (2/3,\ 2/3)$. If we continue, we see that the sequence converges to

$$\boldsymbol{x}^* = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

□

An important question is what happens if the function $f(\boldsymbol{x})$ is not convex (or concave) in the interval that we consider. In that case the method might converge to a local minimum (or maximum). Newton's method can also produce a sequence that does not converge at all, even if the function under consideration is convex. An example of this phenomenon is illustrated by applying Newton's method to the function $f(x) = \frac{2}{3}|x|^{3/2}$ with the initial point $x^{(0)} = 1$, see also Exercise 19.5.

19.2 The method of Steepest Descent

The method of steepest descent is based on the following property: At a given point $\boldsymbol{x}^{(0)}$, the vector $-\nabla f(\boldsymbol{x}^{(0)})$ points in the direction of the most rapid decrease for $f(\boldsymbol{x})$, and the rate of decrease of $f(\boldsymbol{x})$ at $\boldsymbol{x}^{(0)}$ in this direction is $-\|\nabla f(\boldsymbol{x}^{(0)})\|$. Suppose that $f(\boldsymbol{x})$ is a function with continuous first partial derivatives on $\mathbb{R}^n$. Then the Steepest Descent sequence $\{\boldsymbol{x}^{(k)}\}$ with initial point $\boldsymbol{x}^{(0)}$ for minimizing $f(\boldsymbol{x})$ is defined by the following recurrence formula:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - t_k \cdot \nabla f(\boldsymbol{x}^{(k)}) \,, \tag{19.4}$$

where t_k is the value of $t \geq 0$ that minimizes the function

$$\varphi(t) = f(\boldsymbol{x}^{(k)} - t \cdot \nabla f(\boldsymbol{x}^{(k)})), \quad \text{for } t \geq 0 \,, \tag{19.5}$$

that is, we find the stationary point of $\varphi(t)$, if such a point exists, by solving

$$\varphi'(t) = 0 \,.$$

Example 19.2. Find the first iteration of the Steepest Descent sequence for the function $f(x_1, x_2) = 4x_1^2 - 4x_1x_2 + 2x_2^2$ with the initial point $\boldsymbol{x}^{(0)^\mathsf{T}} = (2, 3)$.

We first determine the gradient:

$$\nabla f \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 8x_1 - 4x_2 \\ -4x_1 + 4x_2 \end{pmatrix}, \quad \nabla f(\boldsymbol{x}^{(0)}) = \begin{pmatrix} 4 \\ 4 \end{pmatrix} .$$

Now, minimize the function

$$\begin{aligned} \varphi_0(t) &= f\left(\begin{pmatrix} 2 \\ 3 \end{pmatrix} - t \begin{pmatrix} 4 \\ 4 \end{pmatrix}\right) = f\begin{pmatrix} 2-4t \\ 3-4t \end{pmatrix} \\ &= 4(2-4t)^2 - 4(2-4t)(3-4t) + 2(3-4t)^2 \\ &= 4(2-4t)^2 - 4(6-20t+16t^2) + 2(3-4t)^2 \\ &= 32t^2 - 32t + 10 \,. \end{aligned}$$

We obtain

$$\varphi_0'(t) = 64t - 32 \,.$$

Setting $\varphi_0'(t) = 0$ yields $t_0 = 1/2$, which is a global minimizer since $\varphi_0''(t) = 64 > 0$. The next iterate $\boldsymbol{x}^{(1)}$ is

$$\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)} - t_0 \nabla f(\boldsymbol{x}^{(0)}) = \begin{pmatrix} 2 \\ 3 \end{pmatrix} - \frac{1}{2}\begin{pmatrix} 4 \\ 4 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \end{pmatrix} - \begin{pmatrix} 2 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} .$$

If we continue the procedure another two steps, we get the following sequence, see Figure 19.1.

□

Theorem 19.2. The method of Steepest Descent moves in perpendicular steps. More precisely, if $\{\boldsymbol{x}^{(k)}\}$ is a Steepest Descent sequence for a function $f(\boldsymbol{x})$, then for each $k \in \mathbb{N}$, the vector joining $\boldsymbol{x}^{(k)}$ to $\boldsymbol{x}^{(k+1)}$ is orthogonal to the vector joining $\boldsymbol{x}^{(k+1)}$ to $\boldsymbol{x}^{(k+2)}$.

Proof. The step length t_k is chosen as the minimizer of the function $\varphi(t) = f(\boldsymbol{x}^{(k)} - t_k \nabla f(\boldsymbol{x}^{(k)}))$ for $t \geq 0$, see Equation (19.5). From the Chain rule we obtain

$$\varphi'(t) = -\nabla f(\boldsymbol{x}^{(k)} - t\nabla f(\boldsymbol{x}^{(k)}))^\mathsf{T} \nabla f(\boldsymbol{x}^{(k)}) \,. \tag{19.6}$$

Since $x^{(k+1)} = \boldsymbol{x}^{(k)} - t_k \nabla f(\boldsymbol{x}^{(k)})$, we can rewrite $\varphi'(t)$ in (19.6) as

$$\varphi'(t) = -\nabla f(\boldsymbol{x}^{(k+1)})^\mathsf{T} \nabla f(\boldsymbol{x}^{(k)}) \,.$$

Solving the equation $\varphi'(t) = 0$ yields

$$\nabla f(\boldsymbol{x}^{(k+1)})^\mathsf{T} \nabla f(\boldsymbol{x}^{(k)}) = 0 \,. \tag{19.7}$$

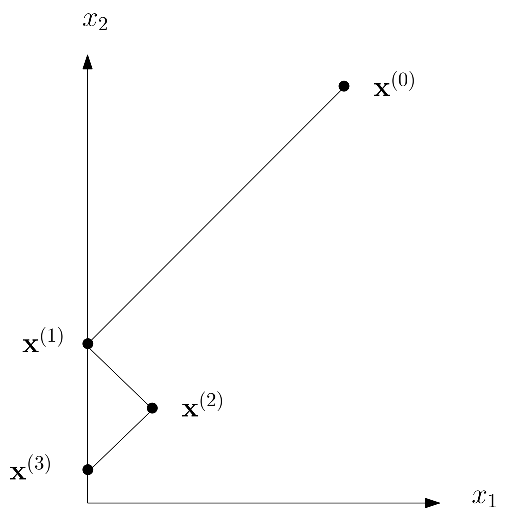


Figure 19.1: Steepest Descent sequence.

Next, we look at the inner product between two subsequent steps in the Steepest Descent sequence:

$$(\boldsymbol{x}^{(k+2)} - \boldsymbol{x}^{(k+1)})^\mathsf{T}(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)})\,.$$

From the recurrence formula in Equation (19.4) we obtain

$$(\boldsymbol{x}^{(k+2)} - \boldsymbol{x}^{(k+1)})^\mathsf{T}(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}) = t_{k+1}t_k\nabla f(\boldsymbol{x}^{(k+1)})^\mathsf{T}\nabla f(\boldsymbol{x}^{(k)})$$

Using (19.7) in the equation above yields $(\boldsymbol{x}^{(k+2)} - \boldsymbol{x}^{(k+1)})^\mathsf{T}(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}) = 0$ and hence that the two consecutive directions are perpendicular. □

Theorem 19.3. If $\{\boldsymbol{x}^{(k)}\}$ is a Steepest Descent sequence for a function $f(\boldsymbol{x})$ and if $\nabla f(\boldsymbol{x}^{(k)}) \neq \mathbf{0}$ for some k, then $f(\boldsymbol{x}^{(k+1)}) < f(\boldsymbol{x}^{(k)})$.

Theorem 19.4. If $f(\boldsymbol{x})$ is a strictly convex coercive function with continuous first partial derivatives on $\mathbb{R}^n$, then for any initial point $\boldsymbol{x}^{(0)}$, the Steepest Descent sequence converges to the unique global minimizer of $f(\boldsymbol{x})$.

There are some drawbacks with the Steepest Descent method and we mention the most important ones here:

- The method might converge very slowly due to its movement by perpendicular steps.
- It might require long computing times to compute the step length t_k.

19.3 Exercises

Exercise 19.1. Compute the Steepest decent iterations $\boldsymbol{x}^{(2)}$ and $\boldsymbol{x}^{(3)}$ from Example 19.2.

Exercise 19.2. Compute the first point $\boldsymbol{x}^{(1)}$ of the Steepest Descent sequence for minimizing the function

$$f(x_1, x_2) = 2x_1^4 + x_2^2 - 4x_1x_2 + 5x_2$$

with initial point $x^{(0)^\mathsf{T}} = (0, 0)$.

Exercise 19.3. Given the function $f(\boldsymbol{x}) = 3x_1^2 + \frac{5}{2}e^{x_2^2} - 3$, will the Steepest Descent sequence with initial point $\boldsymbol{x}^{(0)}$ converge to a unique global minimizer of $f(\boldsymbol{x})$?

Exercise 19.4. Consider the following function: Suppose that A is a symmetric positive definite $n \times n$ matrix, that $\boldsymbol{b} \in \mathbb{R}^n$, $\boldsymbol{x}^{(0)} \in \mathbb{R}^n$, and that $a \in \mathbb{R}^1$. Show that $f(\boldsymbol{x})$ defined by

$$f(\boldsymbol{x}) = a + \boldsymbol{b}^\mathsf{T}\boldsymbol{x} + \frac{1}{2}\boldsymbol{x}^\mathsf{T} A\boldsymbol{x}$$

has a unique global minimizer at the point $\boldsymbol{x}^*$, which is the unique solution of the system

$$A\boldsymbol{x} = -\boldsymbol{b}\,.$$

Also, show that the Newton's method sequence $\{\boldsymbol{x}^{(k)}\}$ with initial point $\boldsymbol{x}^{(0)}$ for minimizing $f(\boldsymbol{x})$ reaches $\boldsymbol{x}^*$ in one step, i.e. $\boldsymbol{x}^{(1)} = \boldsymbol{x}^*$.

Exercise 19.5. Consider the function $f(x) = \frac{2}{3}|x|^{3/2}$. Show that the Newton's method sequence with starting point $x^{(0)} = 1$ does not converge.

Part VII

Appendices

Appendix A

Basic Graph Theory

A.1 (Undirected) graphs

A *graph* is an ordered pair of finite sets (V, E), where each element of E is a size-2 subset of V. The elements of V are called *vertices* and the elements of E *edges*. Graphs can be illustrated by drawing the vertices as dots and the edges as lines connecting the dots. See Figure A.1 for an example.

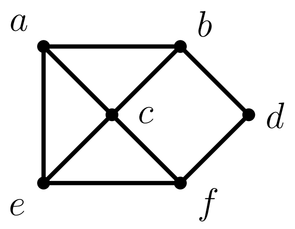


Figure A.1: A graph $G = (V, E)$ with vertex set $V = \{a, b, c, d, e, f\}$ and edge set $E = \{\{a, b\}, \{a, c\}, \{b, c\}, \{a, e\}, \{b, d\}, \{c, e\}, \{c, f\}, \{e, f\}, \{d, f\}\}$.

When there is an edge $\{u, v\}$, we say that u and v are *neighbours*. The vertices u, v are called the *endpoints* of edge $\{u, v\}$. We say that an edge e is *incident* to each of its endpoints. The *degree* $d(v)$ of a vertex v is the number of edges that are incident to v. For example, in Figure A.1, $d(a) = 3$ and $d(c) = 4$.

A *walk* in a graph $G = (V, E)$ is an ordered list of vertices $(v_0, v_1, v_2, \dots, v_k)$ such that $\{v_{i-1}, v_i\} \in E$, for all $1 \leq i \leq k$. The walk is *closed* if $v_0 = v_k$. For example, (a, b, c, a, e, f, c, a) is a closed walk in the graph in Figure A.1. If, in addition, $v_1, \dots, v_k$ are all distinct, then the closed walk is also called a *cycle*. The *length* of the cycle is the number k of dintinct vertices on the cycle. For example, (a, b, d, f, c, a) is a cycle of length 5 in the graph in Figure A.1.

A cycle is called a *Hamilton cycle* if it visits each vertex of the graph. Since a cycle is never allowed to visit vertices more than once, a Hamilton cycle visits each vertex exactly once. For example (a, c, b, d, f, e, a) is a Hamilton cycle of the graph in Figure A.1.

A *path* is a walk that does not visit any vertex more than once. It is called a *u*-*v* path if u is the first vertex and v the last vertex on the path. A graph is *connected* if it contains a *u*-*v* path for each pair of vertices $u, v \in V$. A *connected component* of a graph is a maximal connected subgraph, i.e. a subgraph that is connected and not contained in a larger connected subgraph.

A graph $G' = (V', E')$ is a *subgraph* of a graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. It is an *induced subgraph* if, in addition, all edges $\{u, v\} \in E$ with $u, v \in V'$ are in E'.

Remark A.1. Although we have defined paths, walks and cycles as ordered lists of vertices, we could as well have defined them as ordered lists of edges. We will see them in the latter way whenever that is useful.

A.1.1 Complete graphs

A graph $G = (V, E)$ is *complete* if E contains an edge for each pair of vertices in V. See Figure A.2 for examples.

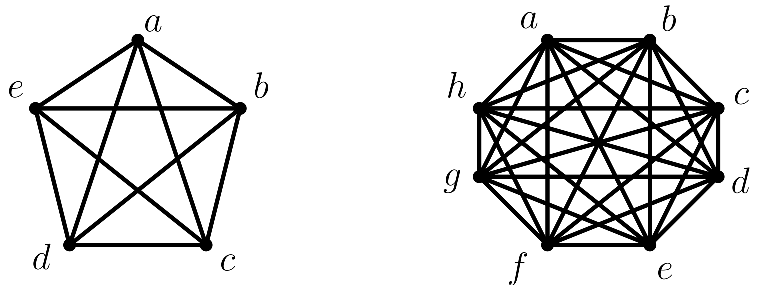


Figure A.2: The complete graph with 5 vertices and the complete graph with 8 vertices.

A.1.2 Trees

A *tree* is a connected graph without cycles, see Figure A.3 for an example. A graph without cycles is called a *forest*. The following lemma is often useful.

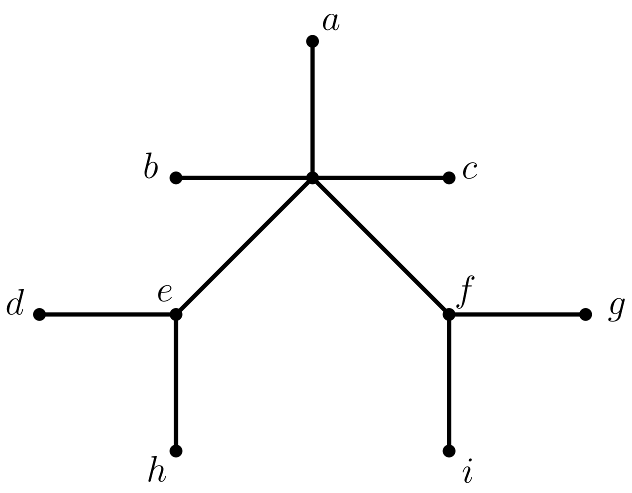


Figure A.3: A tree.

Lemma A.1. Let $G = (V, E)$ be a graph, then the following are equivalent:

1. G is a tree;
2. G is connected and $|E| = |V| - 1$;
3. G contains no cycle and $|E| = |V| - 1$;
4. G contains a unique path between each pair of vertices.

A.1.3 Bipartite graphs

A graph $G = (V, E)$ is *bipartite* if V can be partitioned into two sets U, W such that each edge in E has one endpoint in U and one endpoint in W. Sometimes, the bipartition is already indicated in the definition of the graph: in the bipartite graph $G = (U \cup W, E)$ each edge in E is of the form $\{u, w\}$ with $u \in U$ and $w \in W$. See Figure A.4 for an example.

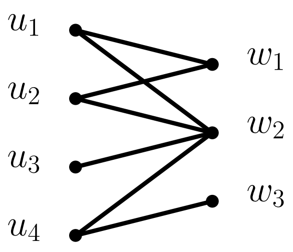


Figure A.4: A bipartite graph $G = (U \cup W, E)$ with $U = \{u_1, u_2, u_3, u_4\}$ and $W = \{w_1, w_2, w_3\}$.

A.1.4 Multigraphs

A *multigraph* is an ordered pair (V, E), where V is a finite set and E is a finite multiset, and each element of E is a size-2 subset of V. Hence, in a multigraph, there can be more than one edge between a given pair of vertices. Such edges are called *parallel edges*. See Figure A.5 for an example. All definitions in the previous section for graphs also apply to multigraphs. For example, the degree of b is $d(b) = 5$ in the multifraph in Figure A.5.

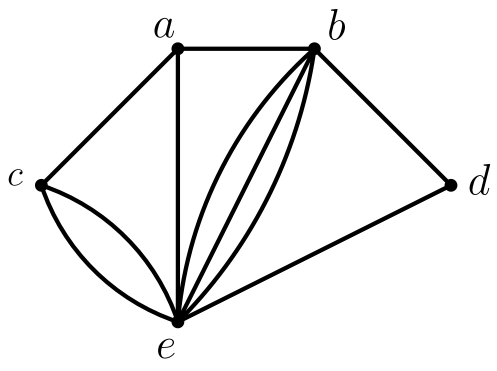


Figure A.5: A multigraph $M = (V, E)$ with vertex set $V = \{a, b, c, d, e\}$ and edge multiset $E = \{\{a, b\}, \{a, c\}, \{b, d\}, \{d, e\}, \{a, e\}, \{c, e\}, \{c, e\}, \{b, e\}, \{b, e\}, \{b, e\}\}$. There are two parallel edges between c and e and three parallel edges between b and e.

A.2 Directed graphs

A *directed graph* is an ordered pair (V, A) where V is a finite set and each element of A is an ordered pair of elements of V. The elements of V are called *vertices* and the elements of A *arcs*. Directed graphs can be illustrated by drawing the vertices as dots and each arc as an arrow from one dot to another dot. See Figure A.6 for an example. A directed graph is also called a *digraph* for short.

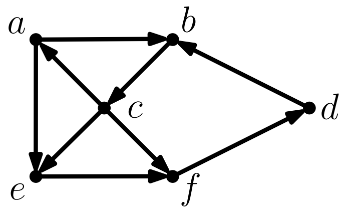


Figure A.6: A directed graph $D = (V, A)$ with vertex set $V = \{a, b, c, d, e, f\}$ and arc set $A = \{(a, b), (c, a), (b, c), (a, e), (c, e), (c, f), (e, f), (f, d), (d, b)\}$.

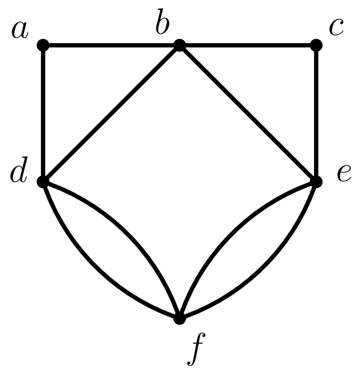


Figure A.7: A Eulerian multigraph $M = (V, E)$.

Vertex v is called the *head* and vertex u the *tail* of arc (u, v). We say that an arc (u, v) is *leaving* u and *entering* v. The *indegree* $d^-(v)$ of a vertex v is the number of arcs entering v and the *outdegree* $d^+(v)$ of a v is the number of arcs leaving v. For example, in Figure A.6, $d^-(b) = 2$ and $d^+(b) = 1$.

A *(directed) walk* in a directed graph $D = (V, A)$ is an ordered list of vertices $(v_0, v_1, v_2, \ldots, v_k)$ such that $(v_{i-1}, v_i) \in A$, for all $1 \leq i \leq k$. It is closed if $v_0 = v_k$. For example, $(a, b, c, a, e, f, d, b, c, a)$ is a closed walk in the directed graph in Figure A.6. If, in addition, $v_1, \ldots, v_k$ are all distinct, then the closed walk is also called a *(directed) cycle*. For example, (b, c, e, f, d, b) is a cycle in the graph in Figure A.6. A directed graph is called *acyclic* if it has no directed cycle.

A cycle in a directed graph is called a *Hamilton cycle* if it visits each vertex of the graph. For example (a, e, f, d, b, c, a) is a Hamilton cycle of the directed graph in Figure A.6.

A *directed path* in a directed graph is a directed walk that does not visit any vertex more than once. A directed graph is *strongly connected* if there exists a directed path from any vertex to any other vertex.

A.3 Euler tours

An *Euler tour* in a graph $G = (V, E)$ is a closed walk $(v_0, v_1, v_2, \ldots, v_k)$ that uses each edge exactly once, i.e., for each $e \in E$ there is exactly one $i \in \{1, \ldots, k\}$ such that $e = \{v_{k-1}, v_k\}$.

For example, an Euler tour in the complete graph on five vertices in Figure A.2 is $(a, b, c, d, e, a, c, e, b, d, a)$.

Euler tours are also interesting when considering multigraphs. An *Euler tour* in a multigraph $M = (V, E)$ is a closed walk $(v_0, v_1, v_2, \ldots, v_k)$ that uses each edge exactly once, i.e., for each $e \in E$ that appears k times in E, there are exactly k indices $i \in \{1, \ldots, k\}$ such that $e = \{v_{k-1}, v_k\}$.

A (multi)graph is *Eulerian* if it has an Euler tour.

For example, $(a, b, c, e, f, d, f, e, b, d, a)$ is an Euler tour in the multigraph in Figure A.7.

The following theorem shows that one can easily find out whether a (multi)graph is Eulerian or not. The proof also describes how to find an Euler tour if one exists.

Theorem A.1 (Euler). A (multi)graph is Eulerian if and only if it is connected and each vertex has an even degree.

Proof. The "only if" direction of the proof is trivial. If a graph has an Euler tour, then it must be connected. Moreover, each time the tour enters a vertex it must also leave again, so all degrees are even.

To show the "if" direction, suppose that M is connected and all degrees are even. Then we can find an Euler tour as follows. We start the walk in an arbitrary vertex v_0 and keep "walking" to a neighbouring vertex as long as there is an incident edge that we have not used yet. Since all degrees are even, we can always keep walking except when we get back to v_0 and have already used all incident edges. Let $W = (v_0, v_1, v_2, \ldots, v_0)$ be our walk so far.

If we have used all edges then we are done. Otherwise, since M is connected, there exists at least one vertex v_i on W of which we have not yet used all incident edges (see Exercise A.10). Then we construct a new walk, starting in v_i, not using any edge more than once and not using any edges from W. When there is no incident edge left to continue our walk, we must be back in v_i, again because all degrees are even. Let $X = (v_i, x_1, x_2, \ldots, v_i)$ be the constructed walk.

Now we combine the two walks W and X into a single bigger walk

$$(v_0, v_1, \ldots, v_i, x_1, x_2, \ldots, v_i, v_{i+1}, v_{i+2}, \ldots, v_0)$$

If this bigger walk uses all edges then we are done. If not, we keep repeating the above procedure until we have used all edges. This gives an Euler tour of M. □

A.4 Exercises

Exercise A.1. What is the number of edges of a complete graph with n vertices?

Exercise A.2. What is the number of edges of a tree with n vertices?

Exercise A.3. Prove that a graph is a tree if and only if there exists a unique path between each pair of vertices.

Exercise A.4. Let G be a graph obtained from a tree by adding a single edge. Prove that G has exactly one cycle.

Exercise A.5. Prove Lemma A.1.

Exercise A.6. Prove that a graph is bipartite if and only if it has no cycle of odd length.

Exercise A.7. Prove or give a counter example for the following statements:

(a) There is an n such that there exists a connected graph with n edges and at least $n + 2$ nodes.

(b) There is a directed graph in which there exists a path of any given integer length $n > 0$.

(c) There is a directed graph in which there exists a walk of any given integer length $n > 0$.

Exercise A.8. Prove that each acyclic directed graph has at least one vertex with indegree 0.

Exercise A.9. Prove that each acyclic directed graph has at least one vertex with outdegree 0.

Exercise A.10. Prove the following statement from the proof of Theorem A.1. If $W = (v_0, v_1, v_2, \ldots, v_0)$ is a closed walk in a connected (multi)graph and W is not an Euler tour, then there exists a vertex v_i on W with an incident edge that is not used by W.

Exercise A.11. An *Euler walk* is a walk that uses each edge exactly once (but does not need to end where it started). Prove that a graph has an Euler walk if and only if it is connected and has at most two vertices with odd degree.

Exercise A.12. An *Euler tour* in a directed graph is a closed directed walk that uses each arc exactly once. Prove that a strongly-connected directed graph has an Euler tour if and only if the outdegree of each vertex is equal to its indegree.

Acknowledgements

We thank Alexander van Amesfoort for many helpful comments on a previous version of these lecture notes.

Index

Bibliography

[1] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming.* Wiley-Interscience. John Wiley & Sons, Inc., 1979.

[2] R. G. Bland. New finite pivoting rules for the simplex method. *Mathematics of Operations research*, 2:103–107, 1977.

[3] S. P. Bradley, A. C. Hax, and T. L. Magnanti. *Applied Mathematical Programming.* Addison-Wesley Publishing Company, Reading, MA, 1977.

[4] V. Chvátal. *Linear Programming.* W. H. Freeman and Company, New York, NY, 1983.

[5] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In Th. C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 339–347, New York, NY, 1951. Wiley.

[6] R. Fletcher. *Practical Methods of Optimization.* Wiley-Interscience. John Wiley & Sons, Ltd., 1980.

[7] S. G. Nash and A Sofer. *Linear and Nonlinear Programming.* Industrial ENgineering Series. McGraw-Hill International editions, 1996.

[8] M. Padberg. *Linear Optimization and Extensions.* Springer-Verlag, Berlin Heidelberg, 1995.

[9] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization, Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, NJ, 1982.

[10] A. L. Peressini, F. E. Sullivan, and J. J. Uhl. *The Mathematics of Nonlinear Programming.* Undergraduate Texts in Mathematics. Springer Verlag, 1988.

[11] W. L. Winston. *Operations Research: Applications and Algorithms.* Duxbury Press, 1994.