Olena Gavriliouk

The Essentials of Optimization:
Linear, Integer, Multicriteria, and Network Optimization

LECTURE NOTES

Preface

I do not pretend to be an expert on teaching Optimization (and I know better than to think that I am an Optimization guru), but *some* of what I know I pass onto you.

I want this course to give you the essentials of Optimization, and, most importantly, to show you how Optimization can be used in production, planning, logistics, supply chain, etc. Therefore, I do not present proofs of theorems and propositions (if you are keen, you can find them in the main references for these Lecture Notes), but I do try to explain why the results of the stated theorems, propositions and corollaries are important and how they can be used in practice (which, in my humble opinion, is most important for practitioners of Optimization).

These Lecture Notes are meant to be supplemented with tutorials and computer exercises. Nowadays, knowing *only* the theory of Optimization is useless if you want to solve optimization problems arising in the real world.

I am not going to pretend that the examples presented in these Lecture Notes are Olena's originals (some are). Most of them are taken from Bazaraa et al. (1990), Wolsey (1998), Hamacher and Klamroth (2000) and Pochet and Wolsey (2000).

Chapter 1 is based on Bazaraa et al. (1990) and Hamacher and Klamroth (2000). Material in chapters 2 and 3 is taken from Wolsey (1998), Nemhauser and Wolsey (1999) and Hamacher (2006). Chapter 4 is an agglomeration from Bazaraa et al. (1990), Hamacher and Klamroth (2000), Drezner and Hamacher (2004) and Ehrgott (2005).

Contents

CHAPTER 1

Linear Programming

"*Linear programming* [(LP), Integer Programming (IP), and Mixed Integer Programming (MIP)] are concerned with the optimization (minimization or maximization) of a linear function while satisfying a set of linear equality and /or inequality constraints or restrictions." (This appended quote is from (Bazaraa et al. 1990, p.1).)

Also, according to Bazaraa et al. (1990), the reason for the terminology *program* instead of *problem* (which I believe would be much clearer in most uses) is due to one of the original applications of LP in a planning software, i.e., program.

Operations Research (OR) became the term of the field of Mathematics that is concerned with LP, IP, MIP, Location theory, etc. In my terminology, and for the purpose of these Lecture Notes, OR is equivalent to Optimization. [1]

[1] Some mathematicians and practitioners might argue with it, but that's not the argument I want to waste my time on.

1.1 Introduction

A linear program[2] (LP) can be represented by a set of **constraints**, **variables**, and an **objective function**. For example,

$$\begin{aligned}
\text{minimize} \quad & c_1x_1 + c_2x_2 + \ldots + c_nx_n && (1.1)\\
\text{subject to} \quad & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 && (1.2)\\
& a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2 && (1.3)\\
& \quad \vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots && \\
& a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m && (1.4)\\
& x_1,\ x_2,\ \ldots,\ x_n \ \geq 0, && (1.5)
\end{aligned}$$

where (1.1) represents a minimization *objective function*, inequalities (1.2)–(1.5) are *constraints*. *Variables* are defined and constrained to be non-negative in constraint (1.5).

Alternatively, an LP is represented as follows:

$$\max\{\mathbf{cx} : A\mathbf{x} \leq \mathbf{b}, \quad \mathbf{x} \geq \mathbf{0}\},$$

where A is an $m \times n$ matrix. [3]

An LP can take on various forms, e.g., maximization and minimization, the constraints can be $\geq$ or $\leq$. Most importantly, however, every LP can be expressed in the following **standard form**:

$$\begin{aligned}
\min \quad & \mathbf{cx} && (1.6a)\\
\text{s.t.} \quad & A\mathbf{x} = \mathbf{b} && (1.6b)\\
& \mathbf{x} \geq \mathbf{0}. && (1.6c)
\end{aligned}$$

Example 1.1
[4] "Ozi Choco" produced hot chocolate mix (item P1) and dark chocolate truffles (item P2). The company has three production facilities, F1, F2, and F3, where the ingredients for the two items are produced.[5] Because "Ozi Choco" produces the best chocolate in Australia, we assume that there is a demand for all that the company produces. The profit per unit of P1 is 3 monetary units and 5 monetary units for a unit of P2. Because "Ozi Choco" must share the facilities with other producers, it only has a certain number of available hours in each facility. Table 1.1 gives the number of hours a unit of each product takes to produce in each facility. What should be produced and in what quantity in order to maximize profit?

[2] Linear program and Linear Programming have the same abbreviation but, clearly, denote two different things: a problem itself, and the entire field of Linear Programming. You will learn to differentiate the two based on a context.

[3] If one desires to be very proper, like some algebraists do, then one writes $\mathbf{c}^T\mathbf{x}$ instead of $\mathbf{cx}$. But I omit such details herein and presume that we can all work out when a vector must be transposed in order to be multiplied with another vector or a matrix.

[4] A modified example from Hamacher and Klamroth (2000).

[5] This is a very fictitious example since cocoa mass is not produced in Australia but is imported—the interesting fact for those of us who are chocolate lovers.

	P1	P2	available capacity
F1	3	2	18
F2	1	0	4
F3	0	2	12

Table 1.1: Hourly production requirements and facilities' capacities for example 1.1.

Let x_1 and x_2 be the amount of product P1 and P2 to be produced. This problem can be formulated as follows:

$$\begin{aligned} \max \quad & 3x_1+5x_2 && \text{(1.7a)} \\ \text{s.t.} \quad & 3x_1+2x_2 \leq 18 && \text{(1.7b)} \\ & x_1 \leq 4 && \text{(1.7c)} \\ & 2x_2 \leq 12 && \text{(1.7d)} \\ & x_1 \geq 0 && \text{(1.7e)} \\ & x_2 \geq 0 && \text{(1.7f)} \end{aligned}$$

△

Obviously, one does not require a computational tool/software/program of any kind or knowledge of LP theory to find the optimal solution to example 1.1. (The famous trial-and-error method can do the job on this small example.) The purpose of example 1.1 is to illustrate graphically in a two-dimensional space the crux of the LP theory.

Inequalities (also referred to as **hyperplanes** (1.7b)–(1.7f) can be plotted as in figure 1.1. Each hyperplane separates the space into two regions: feasible and infeasible. The points in the feasible region satisfy the equation or the inequality of a given hyperplane. The intersection of the feasible regions of each hyperplane is the **feasible region** of the LP under consideration.

Any point within a feasible region is called a **feasible solution** to an LP. One of the feasible solutions is optimal. An **optimal solution** is not necessarily unique.

From equations (1.7a)–(1.7f), one should note the difference between space of the objective function (1.7a) and the space of the constraints (1.7b)–(1.7f). The objective function "lives" in a three-dimensional (3D) space, since it can formally be written as

$$z = 3x_1 + 5x_2.$$

For a given value of z, the **projection** of the objective function can be represented in the space of one dimension lower than the objective function itself, e.g., in example 1.1 the projection of the objective function is a line in 2D.

Figure 1.2 depicts the gradient of the projection of the objective function. Figure 1.3 shows the "movement" of the projection (in dashed lines) in the space of the feasible region. The objective function projection crosses the feasible region when it takes on values between 0 and 36.

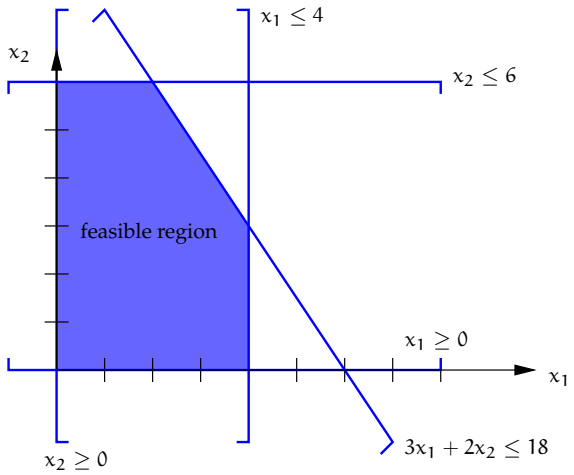


Figure 1.1: The feasible region for example 1.1.

The objective function value (OFV) of 0 is obtained at point $(x_1, x_2) = (0, 0)$ (well, producing nothing results in making no money[6]). The largest OFV is 36 and is achieved at point $(x_1, x_2) = (2, 6)$, which means that the production manager of "Oz Choco" should produce 2 units of P1 and 6 units of P2 in order to make the maximum profit possible.

The largest (for a maximization LP) and the smallest (for a minimization LP) objective function values are termed **optimal** and often denoted by z^{OPT}.

Because a feasible region is formed by intersecting hyperplanes, it is always **convex**. However, a feasible region does not have to be bounded. For an example of an **unbounded feasible region** see figures 1.4(a) and 1.4(b). However, just because a feasible region of an LP is unbounded, it does not necessarily mean that its minimum/maximum OFV is also unbounded (compare figures 1.4(a) and 1.4(b)). An LP is said to be **unbounded** if $z^{\text{OPT}} = \infty$ or $z^{\text{OPT}} = -\infty$.

In addition to being unbounded, a feasible region could be empty. In such a case, an LP is said to be **infeasible** (see figure 1.5).

1.2 Duality

This is one of my favorite topics not only because I remember finding it challenging when I was first introduced to it, but also because it provides a new lens through which one can view LP.

The idea of duality is ubiquitous in mathematics and in real life: addition and subtraction, multiplication and division, positive and negative, etc.

Translating duality into LP terms: for every maximization problem there is a minimization one. As I already mentioned, there are different ways to repre-

[6] As they say it in Germany, "Von nichts kommt nichts."

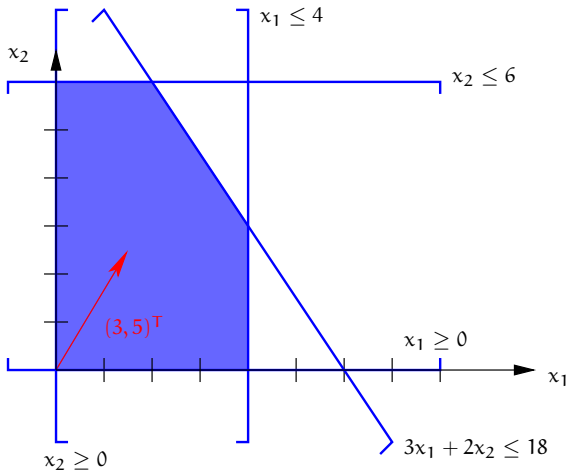


Figure 1.2: The feasible region for example 1.1 with a gradient vector $(3,5)^T$ of the objective.

sent an LP (e.g., the form of the objective function,—min and max,—the form of the constraints,—equality and inequality,—variables).

Based on the form of a **primal** LP, there are several variations of the duality theorem. For example,

Theorem 1.2 (Duality Theorem) *If an LP is feasible and bounded, then*

$$\underbrace{\max\{\mathbf{cx} : A\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\}}_{\textit{primal LP}} = \underbrace{\min\{\mathbf{vb} : \mathbf{v}A \geq \mathbf{c},\ \mathbf{v} \geq \mathbf{0}\}}_{\textit{dual LP}} \tag{1.8}$$

Another important result of duality theory is the following theorem:

Theorem 1.3 (Complementary Slackness Theorem) *A primal-dual feasible pair* $(\mathbf{x}, \mathbf{v})$ *is optimal if and only if* $\mathbf{v}(\mathbf{b} - A\mathbf{x}) = 0$ *and* $(\mathbf{v}A - \mathbf{c})\mathbf{x} = 0$.

To illustrate duality theory, consider the following example:

Example 1.4

$$\begin{aligned}
\max \quad & x_1 + 2x_2 && && (1.9a)\\
\text{s.t.} \quad & x_1 && \leq 3 && (1.9b)\\
& && x_2 \leq 4 && (1.9c)\\
& x_1, && x_2 \geq 0 && (1.9d)
\end{aligned}$$

What is the dual to this LP?

The feasible region of the LP along with the gradient of the objective and the projection of the objective at the optimal point are depicted in figure 1.6(a). The

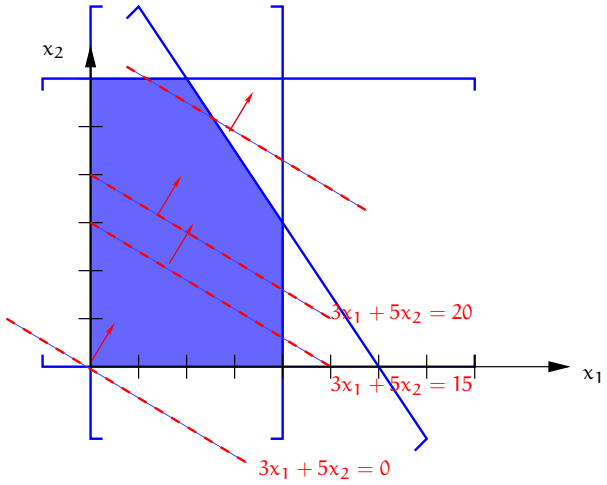
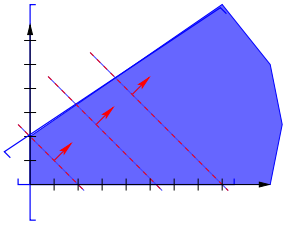
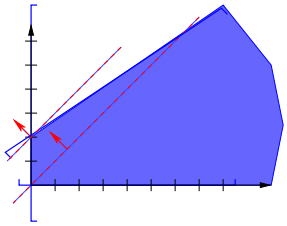


Figure 1.3: The feasible region for example 1.1 with a projection of the objective function. The value of the objective function changes as its projection onto the feasible region "moves" in the direction of its gradient.



(a) An unbounded feasible region with an unbounded objective function value.



(b) An unbounded feasible region with a finite objective function value.

Figure 1.4: Examples of unbounded feasible regions.

optimal solution is at point $(3, 4)$ with $z^{\text{OPT}} = 11$. From figure 1.6(a), note how at the optimal point the gradient of the objective fits in the cone created by the gradients of constraints (1.9b) and (1.9c), which are the **active** constraints at the optimal point.

△

In terms of examples 1.1 and 1.4,theorem 1.2 means the following:
Since both LPs are feasible, i.e., polyhedron $P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\} \neq \emptyset$, then $\exists\ \mathbf{v} \in \mathbb{R}^m : \mathbf{v}A \geq \mathbf{c}$, i.e., the gradient vector $\mathbf{c}$ of the objective function can be expressed as a non-negative linear combination of the gradients of the constraints (which, with a little imagination, one can already see in figures 1.3 and 1.4(b).

The next theorem, known as the *Lemma of Farkas*, gives an insight into duality theory in case of infeasible and unbounded LPs.

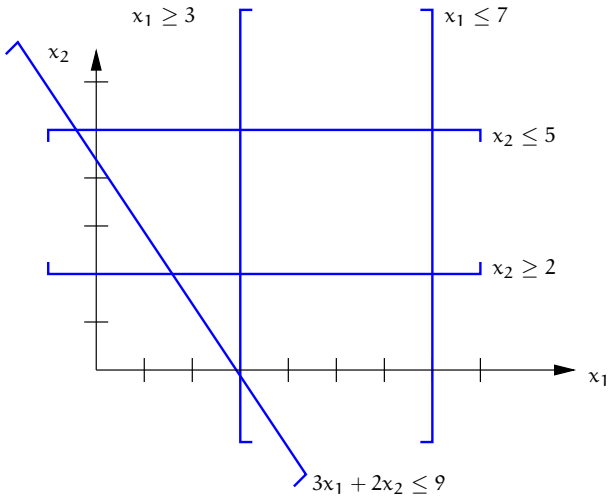
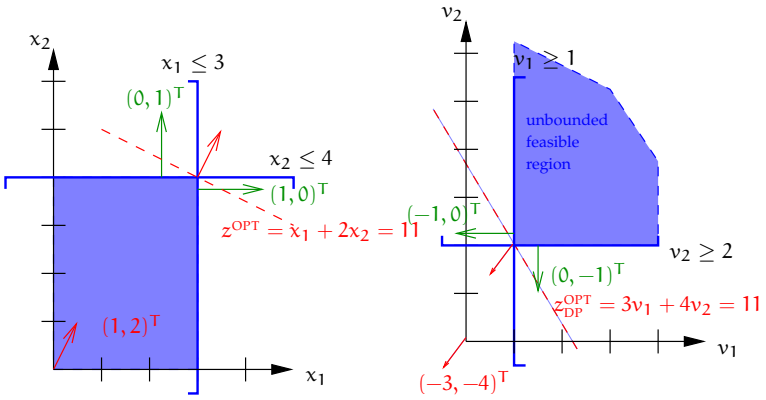


Figure 1.5: An example of an empty feasible region.



(a) The region of the primal LP defined by (1.9a)–(1.9d). The red dashed line is the projection of the objective function at optimal solution $(3,4)$.

(b) The region of the dual LP defined by (1.9a)–(1.9d). The red dashed line is the projection of the objective function at optimal solution $(1,2)$.

Figure 1.6: A portrait of duality. A red arrow is the gradient of the objective function projection. Green arrows are the gradients of the constraints. In the two pictures, at each respective optimal point the gradient of the objective function projection lies within a cone created by the gradients of the constraints active at the optimal point.

Theorem 1.5 (Lemma of Farkas) *The following exclusive alternatives hold for given* $A \in M_{m,n}(\mathbb{R})$ *and* $\mathbf{b} \in \mathbb{R}^m$

1. *Either* $\{\mathbf{x} \in \mathbb{R}^n_+ : A\mathbf{x} \leq \mathbf{b}\} \neq \emptyset$
   or $\{\mathbf{v} \in \mathbb{R}^m_+ : \mathbf{v}A \geq \mathbf{0} \quad \textit{and} \quad \mathbf{v}\mathbf{b} < 0\} \neq \emptyset$

2. *Either* $\{\mathbf{x} \in \mathbb{R}^n_+ : A\mathbf{x} = \mathbf{b}\} \neq \emptyset$
   or $\{\mathbf{v} \in \mathbb{R}^m : \mathbf{v}A \leq \mathbf{0} \quad \textit{and} \quad \mathbf{v}\mathbf{b} > 0\} \neq \emptyset$
3. *Either* $\{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\} \neq \emptyset$
   or $\{\mathbf{v} \in \mathbb{R}^m_+ : \mathbf{v}A = \mathbf{0} \quad \textit{and} \quad \mathbf{v}\mathbf{b} < 0\} \neq \emptyset$
4. *For* $P = \{\mathbf{r} \in \mathbb{R}^n_+ : A\mathbf{r} = \mathbf{0}\}$*: Either* $P \backslash \{\mathbf{0}\} \neq \emptyset$ *or* $\{\mathbf{v} \in \mathbb{R}^m : \mathbf{v}A > \mathbf{0}\} \neq \emptyset$

Proof: Consider an LP $\max\{\mathbf{0} \cdot \mathbf{x} : \mathbf{x} \in P\}$, where P is any of the sets in the "either" options. Items (1)-(3) of the theorem follow from the duality theory and the fact that $\mathbf{v} = \mathbf{0}$ provides a dual feasible solution.

1. If $P = \{\mathbf{x} \in \mathbb{R}^n_+ : A\mathbf{x} \leq \mathbf{b}\} \neq \emptyset$, then $0 = \max\{\mathbf{0} \cdot \mathbf{x} : \mathbf{x} \in P\} \leq \mathbf{v}\mathbf{b} \ \ \forall \mathbf{v} \in \mathbb{R}^m_+$ with $\mathbf{v}A \geq \mathbf{0}$. Then $Q = \{\mathbf{v} \in \mathbb{R}^m_+ : \mathbf{v}A \geq \mathbf{0}, \mathbf{v}\mathbf{b} < 0\} = \emptyset$.
   By the same argument, if $Q \neq \emptyset$, then $P = \emptyset$. And since $\mathbf{0} \in \mathbb{R}^m_+$ satisfies $\mathbf{0}A \geq \mathbf{0}$, one of the alternatives always holds.
2. Exercise.
3. Exercise.
4. Consider an LP $\max\{\mathbf{1}\mathbf{r} : A\mathbf{r} = \mathbf{0}, \mathbf{r} \geq \mathbf{0}\}$. Since $\mathbf{0}$ is feasible for this LP, then $\{A\mathbf{r} = \mathbf{0}, \mathbf{r} \geq \mathbf{0}\} \backslash \{\mathbf{0}\} \neq \emptyset \Leftrightarrow$ this LP is unbounded $\Leftrightarrow \min\{\mathbf{0}\mathbf{v} : \mathbf{v}A \geq \mathbf{1}\}$ infeasible $\Leftrightarrow \{\mathbf{v} : \mathbf{v}A > \mathbf{0}\} = \emptyset$.

■

Duality theory introduced herein (i.e., theorems 1.2, 1.3, and 1.5) basically states the following:

1. If an LP is not unbounded or infeasible, then the optimal OFV of a primal LP (i.e., z^{OPT}) equals the optimal OFV of the dual LP (denoted by $z^{\text{OPT}}_{\text{DP}}$) . (Due to theorem 1.2.)
2. If an LP has an optimal solution, then the gradient of the objective can be written as a non-negative linear combination of the gradients of the constraints which are active at that corner point of the feasible region of the LP. (Due to theorem 1.2.)
3. The Complementary Slackness theorem 1.3, says that if an optimal solution to a primal (dual) LP is known, then one can easily find an optimal solution to its dual (primal) LP. (Just solve the equations in the statement of theorem 1.3.)
4. The dual of the dual is the original primal.
5. If a primal (dual) LP is unbounded, its dual (primal) is infeasible. (Due to theorem 1.5.)

1.3 A few Words about Simplex

If you have taken an introductory OR course, you have heard about the **Simplex**—method for solving LPs. There is a two-fold reason why I am not presenting the gruesome details of Simplex in these Lecture Notes:

- there are many excellent presentations of Simplex in literature (see Hamacher and Klamroth (2000) and Bazaraa et al. (1990)); and
- all real world optimization problems I have encountered are either too trivial or too large to attempt solving them with Simplex by hand.

However, keeping in the spirit of these Lecture Notes, I present to you the insight and the idea behind Simplex.

Because each LP can be written in the standard form, Simplex consists of a sequence of algebraic operations on a system of linear equations.

There is one key difference between Simplex and solving a system of linear equations (like what you have probably seen in a course on Linear Algebra): during Simplex the variables must remain non-negative at all times.

Here is my stab at trying to explain the Simplex method in layman's terms:

1. Simplex method always starts at a corner point (e.g., point $\mathbf{x^0}$ in figure 1.7).
2. The corner points of a simplex (i.e., the convex polyhedron which is the feasible region, and hence the name of the method) satisfy the definition of what is known as a **basic feasible solution**.
3. The algebraic operations involved in Simplex method are designed in such a way that each step (called a **Simplex iteration**) goes from one basic feasible solution to the next (i.e., from one corner point to the next as shown by arrows in figure 1.7).
4. Moreover, from one iteration to the next the value of the objective function either improves or at least does not worsen. (If special rules are observed, Simplex will never get stuck in an intermediate corner point on its way to an optimal corner point.)
5. Simplex gives an indication to whether an LP is unbounded or infeasible. Alternatively, it shows when an optimal solution is reached.
6. There are at most $\begin{pmatrix} n \\ m \end{pmatrix}$ basics feasible solutions, and in the worst-case scenario Simplex must iterate (visit) all of them. Hence, if special rules for avoiding getting stuck in a corner point are observed, then Simplex will terminate after at most $\begin{pmatrix} n \\ m \end{pmatrix}$ iterations.

Because in the worst-case scenario Simplex must go through

$$\begin{pmatrix} n \\ m \end{pmatrix} = \frac{n!}{m!(n-m)!} > \left(\frac{n}{m}\right)^m$$

basic feasible solutions. Note that for $n \geq 2m$, $\left(\frac{n}{m}\right)^m \geq 2^m$, hence Simplex is exponential in the order of variables (n) and constraints (m). For a more detailed discussion see section 8.2 of Bazaraa et al. (1990).

The exponential effort of Simplex was confirmed by the famous *Klee-Minty* example on which Simplex performed $2^n - 1$ iterations through all the vertices of

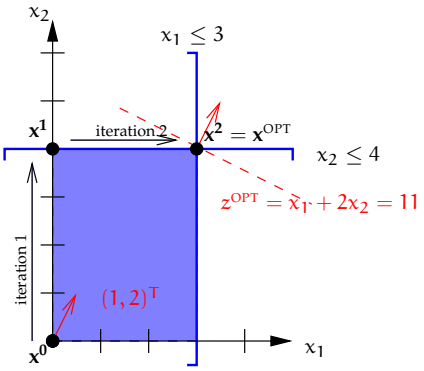


Figure 1.7: An rough illustration of Simplex method.

the feasible region of the polyhedron of the Klee-Minty LP example. But this was an example especially designed by academics.

Statistical analysis on the number of the Simplex iterations necessary to get to optimality resulted in an average of $3m/2$ iteration and rarely more than $3m$ iterations.

In 1979, L. G. Khachian developed an *ellipsoid algorithm* which solves LPs in polynomial time. Then came *Karmakar's projective algorithm* which is also polynomial. Hence the *theoretical* question of whether each LP can be solved in polynomial time (with respect to the number of variables and constraints) was answered affirmatively.

However, to my best knowledge, the best solvers on the market (e.g., CPLEX, Gurobi) still use different flavors of Simplex (e.g., primal Simplex, dual Simplex, primal-dual Simplex).

1.4 Gurobi Basics (on a Linux Operating System)

Gurobi Optimizer (or simply Gurobi) is a commercial solver for OR problems—a black box which solves linear, integer, and nonlinear programs. I.e., Gurobi is a library of encoded routines/algorithms which can solve optimization problems. One can use C, C++, Java, Python, VB and C# to "talk" to Gurobi.[7]

A free version of Gurobi that handles at most 300 variables can be downloaded from www.gurobi.com (Obviously, you cannot try it on your ABS computer.)

You all have an account on "Elise," one of ABS's Linux machines. When you enter *Elise*, you will have a *terminal* window waiting for your commands (terminal window is also referred to as *xterm*) as well as the first prompt `elise:~>`

[7] Analogy: Gurobi as a library of books (procedures), if you want to get information (use) out of a book (procedure), you must know the language in which it is written. A book (procedure) which you need does not automatically put the information you need into your brain (solve your optimization problem), you must do the time consuming work of reading a book (extracting the necessary procedure from Gurobi.

where you can start typing your first command.

In these Lecture Notes, I give you directions on how to execute a Linux command as follows:
elise: commandname
After you typed a command into a prompt (i.e., after elise:~>) in a terminal window, press the "Enter" key.

To find Gurobi examples, look into its directory as follows:
elise:~> ls /opt/gurobi301/linux64/examples/

Directory /opt/gurobi301/linux64/examples/ contains directories with examples in different programming languages.

Explore what is in directory /opt/gurobi301/linux64/ by typing:
elise:~> ls /opt/gurobi301/linux64/
There are a few directories here, e.g., bin, docs, examples, include, lib. We are most interested in docs and examples. However, look into directories bin, include and lib to familiarize yourself with what is in them.

If you need to return to your *home directory* at any point type:
elise:~> cd

It is useful for you to have your own local copy of /opt/gurobi301/linux64/examples/ and /opt/gurobi301/linux64/docs/. In order to accomplish that, make directory GurobiStuff in your home directory as follows:
elise:~> mkdir GurobiStuff
Now copy /opt/gurobi301/linux64/docs into the new directory:
elise:~> cp -r /opt/gurobi301/linux64/docs GurobiStuff/. and
elise:~> cp -r /opt/gurobi301/linux64/examples GurobiStuff/.

Now a good idea is to copy my .profile into your home directory, i.e.,
elise:~> cp /home/gavrol/.profile .
Now update your profile with
elise:~> source .profile

.profile contains path settings, plus a couple other useful settings (view it with one of the editors).

Now we are close to start trying things. Copy my ORLectures, i.e.,
elise:~> cp -r /home/gavrol/ORLectures .

You will be using kwrite editor. If you are familiar with vi or emacs feel free to use those.

To set the size of your editor window, do as follows:
elise:~> kwrite 1000x900 &

To open any file in kwrite do:
elise:~> kwrite [filename] &
for example: elise:~> kwrite .profile &

Having done all that, we are ready to implement the next example with C++ and Gurobi.

Example 1.6 (Using object-oriented programming and Gurobi)
A corporation has $30 million (M) available for the coming year to allocate to its three subsidiaries. The corporation has established a minimal level of funding for each subsidiary. These funding levels are $3M, $5M and $8M respectively. Subsidiary 2 cannot utilize more than $17M without major new capital expansion. Each subsidiary has the opportunity to conduct various projects with the funds it receives. A rate of return has been established for each project. Moreover, certain projects permit only limited investments. The data are given in table 1.2. How should investment be made in order to maximize profit while simultaneously satisfying all requirements?

Subsidiary	Project	Rate of return	Upper limit of investment
1	1	0.08	$6M
	2	0.06	$5M
	3	0.07	$9M
2	4	0.05	$7M
	5	0.08	$10M
	6	0.09	$9M
3	7	0.10	$6M
	8	0.06	$3M

Table 1.2: Project information for example 1.6.

△

1.5 Exercises

Exercise 1.1 *I stated in this section that any form of LP can be converted into any other. Take example 1.1 and rewrite the objective function* (1.7a) *as a minimization. Then re-write the example in the standard form.*

Exercise 1.2 *Can you change the objective function of example 1.1 so that there are multiple optimal solutions?*

Exercise 1.3 *Use duality (i.e., theorem 1.5) to conclude that if a primal LP is infeasible then its dual must be unbounded.*

Exercise 1.4 *Show how the Complementary Slackness theorem is satisfied in example 1.4.*

Exercise 1.5 *A small joinery makes two different sizes of boxwood chess sets. The small set requires 2 hours of machining on a lathe, and the large set requires only 1 hour. There are three lathes with skilled operators who each*

work 40 hours per week. The small chess set requires 2 kg of boxwood, and the large one needs 5 kg. There are only 200 kg of boxwood available per week. Also it is known that at most 30 of large sets can be sold each week. A profit from the small set is \$3 and from the large one \$1. How many sets of each kind should the shop produce to maximize profit? What is the maximal profit?

1. *Set up a linear model for this problem.*
2. *Solve the problem graphically.*
3. *How does the solution change when the profits are \$4 and \$2 for the small and the large sets, respectively? Compare the objective values if the solution from part (2) is used for these profits.*
4. *How should the profit change so that the unique optimal solution is to produce 50 small and 20 large sets?*

Exercise 1.6 *Consider the following linear programming problem:*

$$\begin{aligned}
\max \quad & x_1 + x_2 + 3x_3 & \\
s.t \quad & x_1 + 2x_2 + x_3 \leq 14 & (1.10)\\
& x_1 + 2x_2 + 4x_3 \geq 5 & (1.11)\\
& x_1 + x_2 - x_3 \leq 3 & (1.12)\\
& x_1 \text{ unrestricted} & (1.13)\\
& x_2 \geq 0 & (1.14)\\
& x_3 \leq 0 & (1.15)
\end{aligned}$$

1. *Reformulate the problem so that it is in standard form.*

Exercise 1.7 *Consider the problem* $\{\min cx,\ s.t\ Ax \geq b,\ x \geq 0\}$. *Consider the following situations:*

1. *A new constraint,* $A_{m+1.}$, *is added to the problem.*
2. *A new variable,* x_{n+1} *with coefficients in* $A_{.\,n+1}$, *is added to the problem.*
3. *Constraint* $A_{i.}x = b_i$ *for some* $i \in \{1, \ldots, m\}$ *is deleted from the problem.*
4. *Variable* x_j *for some* $j \in \{1, \ldots, n\}$ *is deleted from the problem.*

For each of the above cases answer the following two questions 1) What happens to the feasible region? 2) What happens to the optimal objective value z^{OPT}*?*

Exercise 1.8 (adapted from Bazaraa et al. (1990)) *Consider the problem of locating a new machine on an existing layout consisting of four machines located at the following* x_1 *and* x_2 *coordinates:* $\binom{3}{0}$, $\binom{0}{-3}$, $\binom{-2}{1}$, *and* $\binom{1}{4}$. *Let the coordinates of the new machine be* $\binom{x_1}{x_2}$. *The* ℓ_1 *metric (also called street or Manhattan distance) defined by*

$$\ell_1\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}\right) = |x_1 - a_1| + |x_2 - a_2|$$

is used to measure the distance between the new and the existing machines. Formulate the problem of finding an optimal location as a linear program for each of the following cases:

1. *Minimize the sum of the weighted distances, where the weights corresponding to the four machines are* 5, 7, 3, *and* 1*, respectively (reflecting the travel frequencies between the old and the new machines). Solve the problem using Gurobi or any software of your choice and provide the output of the solution.*
2. *Minimize the maximum distance between any of the existing machines and the new one. (Remember that the objective function must be linear!)*

Exercise 1.9 (from Bazaraa et al. (1990)) *Fed has* M *amount of money to invest over the next five years. At the beginning of each year he can invest money in one- or two-year certified deposits (CDs). The bank pays* r_1 *and* r_2 *interest on the one-year and the two-year CD, respectively. In addition, the bank will offer three-year CDs at the beginning of the second year; this three-year CD will pay* r_3 *in total interest. If a CD is terminated prematurely no interest is paid. How should Fed invest his money in order to maximize his total cash at the end of the fifth year?*

1. *Formulate as an LP. Write C++/Python program to solve the problem using Gurobi. (Parameters* r_1, r_2, r_3 *and* M *should be the input to your program.)*
2. *Experiment with changes to* r_1, r_2, r_3 *and with the same* M *to determine the volatility of the investment strategies.*
3. *How does changing* M *reflect on the investment strategy?*

Exercise 1.10 (modified from Bazaraa et al. (1990)) *Employ C++ object-oriented capability (like we did in the example 1.6) to solve the following problem. We need to develop a computerized menu-planning system. Menu is divided into three major categories: vegetables, meat, and fruit. At least one serving unit of each category is required (so, vegetarians are out of the picture). The cost per serving unit of some suggested items as well as their content of carbohydrates, vitamins, protein, and fats are summarized in table 1.3. Suppose that the minimal requirements of carbohydrates, vitamins, protein, and fat per meal are, respectively, 5, 10, 10, and 2. The sugar content should not exceed 20. Formulate the menu-planning problem as a linear program. Solve with Gurobi.*

	Carbs	Vitamins	Protein	Fat	Sugar	Cost in $/serving
Vegetables						
Peas	2	3	1	0	4	0.1
Green beans	2	2	1	0	2	0.1
Okra	2	3	1	0	2	0.13
Corn	2	3	1	0	5	0.09
Rice	3	2	1	1	2	0.1
Potatoes	4	1	1	1	3	0.07
Meat						
Chicken	2	5	4	3	1	0.75
Beef	3	4	6	6	2	0.90
Fish	2	8	5	3	1	1.20
Lamb	3	6	6	6	1	1.00
Fruit						
Citrus	1	5	0	0	6	0.40
Apples	1	3	0	0	5	0.25
Melons	1	3	0	0	4	0.60
Grapes	1	4	0	0	4	0.55

Table 1.3: Diet data for exercise 1.10.

CHAPTER 2

Integer Programming: Theory

In this chapter I scrape through the seemingly bottomless theory of Integer Programming (IP). Even though by definition Mixed Integer Programming (MIP) contains IP, in my eyes the two are equivalent enough so that there is no need to make a special chapter about the topics of MIP.

To stress that optimization is all around us, here is the first example of an integer program.

Example 2.1 (0-1 KNAPSACK)
You need some groceries, so you take your sturdy shopping bag and go along the High St to "Thomas Dux" (there is no way you are driving down the High Street in Armadale on Saturday around brunch time, that street is an impenetrable chaos, so you walk). While you are in the store you decide what are the most essential things that you need to buy (because your bag is not of infinite capacity). Each item j ($j = 1, \ldots, n$) has value c_j to you in terms of, for example, the probability that you want to eat/cook with it on that day. Simultaneously, each item has cost a_j in terms of space it takes up in your shopping bag. Obviously, you want to choose items in such a way so that the total value is maximized but yet they all fit in your bag.

Hence your decision with respect to each item is: to choose or not to choose.[1] Let variable x_j represent that decision for item j, hence each x_j is a binary variable that could take value 0 or 1.

$$\max \quad \sum_{j=1}^{n} c_j x_j \tag{2.1}$$
$$\text{s.t.} \quad \sum_{j=1}^{n} a_j x_j \leq b \tag{2.2}$$
$$x_j \in \mathbb{B} \qquad j = 1, \ldots, n \tag{2.3}$$

△

[1] Even though Hamlet might have not realized it then, his problem was also an optimization one.

Note, that finding the most efficient way to pack a grocery bag is yet another optimization problem.

Definition 2.2 Integer program (IP): $\max\{\mathbf{cx} : A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}, \mathbf{x} \in \mathbb{Z}^n\}$, *where* A *is an* $m \times n$ *matrix,* $m \geq n$.

Mixed integer program (MIP): $\max\{\mathbf{cx}+\mathbf{hy} : A\mathbf{x}+G\mathbf{y} \leq \mathbf{b}, \mathbf{x},\mathbf{y} \geq \mathbf{0}, \mathbf{x} \in \mathbb{Z}^n\}$.

0-1 or binary program (BIP): $\max\{\mathbf{cx} : A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \{0,1\}^n := \mathbb{B}^n\}$.

Combinatorial optimization problem (COP): *Given a finite set* $\mathcal{N} = \{1,\dots,n\}$, *weights* c_i *with* $i \in \mathcal{N}$ *and a family* $\mathbb{F}$ *of feasible subsets of* $\mathcal{N}$, *a COP can be described as* $\min\{\sum_{i\in S} c_i : S \in \mathbb{F}\}$.

I do not differentiate between a BIP and a COP, because the latter can be expressed as a BIP as follows: Characterizing each $S \in \mathbb{F}$ by its *incidence vector* $\mathbf{x} = \mathbf{x}_s \in \mathbb{B}^n$ defined by

$$x_j = \begin{cases} 1 & \text{if} \quad j \in S \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, one can show that an IP and a BIP are equivalent as well (but I leave it to you as an exercise). Therefore, unless crucial, I do not differentiate between BIP, COP, and IP, but refer to all of them as IP in these Lecture Notes[2].

In many cases, e.g., the famous *Traveling Salesman Problem*, m is exponential in n, so that $A\mathbf{x} \leq \mathbf{b}$ cannot be written out *explicitly* (see example 2.3).

Example 2.3 ((Asymmetric) Traveling Salesman Problem (TSP))
Given a set $\mathcal{N}$ of n cities with costs c_{ij} of traveling from i to j ($c_{ij} \neq c_{ji}$ is possible!), find a tour of a salesman visiting each city exactly once and minimizing the overall cost. (Variations of TSP: courier pick-up and delivery.) TSP is usually referred to as COP with $\mathbb{F}$ being a set of all possible tours.

$$x_{ij} = \begin{cases} 1 & \text{if salesman goes directly from } i \text{ to } j \\ 0 & \text{else.} \end{cases}$$

$$\min \quad \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{2.4a}$$

$$\text{s.t.} \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\, i = 1,\dots,n \tag{2.4b}$$

$$\sum_{j=1}^{n} x_{ji} = 1 \qquad \forall\, i = 1,\dots,n \tag{2.4c}$$

$$\sum_{i\in S}\sum_{j\notin S} x_{ij} \geq 1 \qquad \forall\, \emptyset \subsetneqq S \subsetneqq \mathcal{N} \tag{2.4d}$$

$$x_{ij} \in \mathbb{B} \qquad \forall\, i,j = 1,\dots,n$$

[2] In my mind BIP, COP, IP, and MIP are all the same. And I think you are safe if you think about it that way too.

Constraints (2.4b) ensure that a salesman leaves a city exactly once. Constraints (2.4c) ensure that a salesman enters a city exactly once. Constraints (2.4b) and (2.4c) are known as the **assignment constraints** [3]. Constraints (2.4d) are known as the **subtour elimination constraints**. As the name suggests, (2.4d) ensure that a salesman has one continuous tour, i.e., prevent a situation as in figure 2.1.

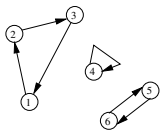


Figure 2.1: If subtour elimination constraints for TSP are not included, the depicted situation can occur.

How many subtour elimination constraints are there? (For $n = 6$ write out explicitly the subtour elimination constraints.) The number of the subtour elimination constraints is exponential in the number of cities to be visited. △

An obvious way to try to handle IPs is to relax them, i.e., instead of solving

$$\max \{\mathbf{cx} : A\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq \mathbf{0},\ \mathbf{x} \in \mathbb{Z}^n\} \tag{2.5}$$

solve the **relaxed IP**

$$\max \{\mathbf{cx} : A\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\} \tag{2.6}$$

where integrality constraints on the variables $\mathbf{x}$ are omitted.

This way of handling an IP delivers an optimal LP solution, $\mathbf{x}_{\text{LP}}^{\text{OPT}}$. However, $\mathbf{x}_{\text{LP}}^{\text{OPT}}$ could be infeasible because it is not an *integral* (i.e., integer) solution (see figure 2).

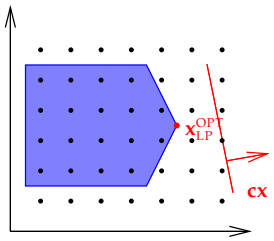


Figure 2.2: Infeasible optimal solution under relaxation.

Another idea is to solve a relaxed IP and then round the obtained optimal solution $\mathbf{x}_{\text{LP}}^{\text{OPT}}$. However, this rounded solution is most often sub-optimal to the original IP. (Find an example in 2-dimensions (2D) where such rounding fails.)

[3] We encounter assignment constraints on a few more occasions throughout these Lecture Notes, so make note of them.

2.1 Polyhedral Analysis

To lead up to the explanation about the difficulty of solving IPs, we must get through some definitions and facts (a.k.a. theorems).

Definition 2.4 Convexity: *Set* $T \subseteq \mathbb{R}^n$ *is* ***convex*** *if* $\forall\, \mathbf{x}, \mathbf{y} \in T$ *and* $0 \leq \mu \leq 1$ $\mu\mathbf{x} + (1-\mu)\mathbf{y} \in T$.

Convex combination: *For* $\mathbf{x}^1, \ldots, \mathbf{x}^t \in \mathbb{R}^n$ *any* $\mathbf{x} = \sum_{i=1}^t \lambda_i \mathbf{x}^i$ *with* $\sum_{i=1}^t \lambda_i = 1$ *and* $\lambda_i \geq 0$ *for* $1 \leq i \leq t$ *is called a* ***convex combination*** *of* $\mathbf{x}^1, ..., \mathbf{x}^t$.

Convex hull: *For* $X \subseteq \mathbb{R}^n$

$$\operatorname{conv}(X) := \left\{ \sum_{i=1}^t \lambda_i \mathbf{x}^i : \; t \in \mathbb{N},\; \lambda_i \geq 0,\; \sum_{i=1}^t \lambda_i = 1, \mathbf{x}^i \in X \text{ for } i = 1, \ldots, t \right\} \tag{2.7}$$

is the ***convex hull*** *of* X *(i.e,* $\operatorname{conv}(X)$ *is the set of all points that are convex combinations of points in* X*).*

If you feel like doing a proof, show that $\operatorname{conv}(X)$ is a convex set.

Definition 2.5 (Polyhedron) $P = P(A, \mathbf{b}) = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\} \subseteq \mathbb{R}^n$ *where* A *is an* $m \times n$ *matrix and* $\mathbf{b} \in \mathbb{R}^m$ *is a* ***polyhedron****.* P *is a* ***rational polyhedron*** *if* $(A, \mathbf{b})$ *is a rational matrix. A bounded polyhedron is called a* ***polytope****.*

Obviously[4], every polyhedron is a convex set. Polyhedra may also be of type $\mathbf{x} \in \mathbb{R}^n_+$ instead of $\mathbb{R}^n$, $A\mathbf{x} = \mathbf{b}$ instead of $A\mathbf{x} \leq \mathbf{b}$ or $A\mathbf{x} \geq \mathbf{b}$.

Definition 2.6 (Formulation) *Polyhedron* $P = P(A, \mathbf{b}) \subseteq \mathbb{R}^n$ *is a* ***formulation*** *for set* $X \subseteq \mathbb{Z}^n$ *if* $X = P \cap \mathbb{Z}^n$.

Generalization to situations of mixed integer programs: $P \subseteq \mathbb{R}^{n+q}$ is a formulation for $X \subseteq \mathbb{Z}^n \times \mathbb{R}^q$ if and only if $X = P \cap (\mathbb{Z}^n \times \mathbb{R}^q)$.

Unfortunately, the word *formulation* is also used to describe the set of constraints that define an IP, e.g., we say that (2.1)–(2.3) are a *formulation* of a KNAPSACK. Such a nomenclature is a bit imprecise, but there is a reason why it has stuck.

For the example of the KNAPSACK, the formulation resulting from (2.2)–(2.3) is the polytope described by

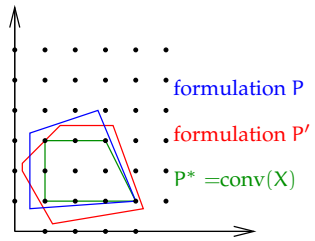
$$\{\sum_{j=1}^n a_j x_j \leq b, \text{ and } 0 \leq x_j \leq 1 \; \forall\, j = 1, \ldots, n\}. \tag{2.8}$$

So formulation (2.8) somewhat resembles what is described by (2.2)–(2.3), hence the alternative use of the word *formulation*.

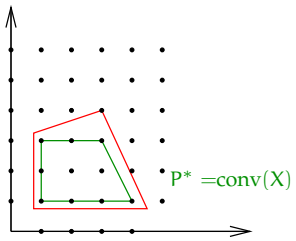
When a context is not clear, I use the phrase **formulation polyhedron** (or *formulation polytope*) instead of just *formulation*.

[4] If it is not obvious, do a proof.

Example 2.7
For $n = 2$, let $X = \{(i,j) : \ i = 1,2,3, \ j = 1,2,3\} \cup \{(4,1)\}$. Figure 2.3(a) shows three different formulations of X.



(a) Different formulation polyhedra. Note that none of the formulation polytopes includes integer points that are *not* in X. Formulation polyhedra include surrounding space in $\mathbb{R}^2$, some more than others.



(b) Red polytope in this figure is not a formulation of the given X as it includes point $(3,4)$ which is not X.

Figure 2.3: Formulation polyhedra and invalid formulation polyhedra.

△

As demonstrated in example 2.7, an IP can have more than one formulation. In general, formulations can not be compared (see formulations P and P′ in example 2.7), hence, one cannot state that one formulation is better than another.

Why is the idea of a formulation such an important one? Well, for each IP there is a formulation that is the Holy Grail[5] of all formulations—*the ideal formulation*.

Definition 2.8 (Better and Ideal Formulations) *Given* $X \subseteq \mathbb{R}^n$, *formulation* P_1 *for* X *is* ***better*** *than formulation* P_2 *for* X *if* $P_1 \subsetneqq P_2$. *Formulation* P^* *is the* ***ideal formulation*** *if* $P^* \subseteq P$ *for any other formulation* P *for* X.

Since polyhedra are convex sets, and by definition 2.6 $X \subset P$, then $\text{conv}(X) \subseteq P$ for any formulation P of X (provided a formulation of X exists). Thus, $\text{conv}(X)$ could be that *ideal* formulation *if and only if* $\text{conv}(X)$ is a polyhedron (because, by definition 2.5, a formulation must be a polyhedron).

Thus, in order to show that $\text{conv}(X)$ is the ideal formulation, we must show that it is a polyhedron. Our intuition, based on the mental pictures in 2D (like that in example 2.7), says, "How could it not be?" But it must be proven.

Bear with a couple more definitions.

Definition 2.9 (Extreme point) $\mathbf{x} \in P = P(A, \mathbf{b})$ *is an* ***extreme point*** *of* P *if it cannot be written as a convex combination of any other point* $\mathbf{x}^1, \mathbf{x}^2 \in P$, *i.e.*

$$\mathbf{x} = \lambda \mathbf{x}^1 + (1-\lambda)\mathbf{x}^2 \text{ for } 0 < \lambda < 1 \text{ implies } \mathbf{x}^1 = \mathbf{x}^2 = \mathbf{x}. \tag{2.9}$$

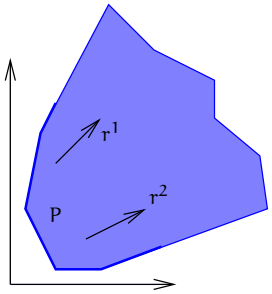
[5]However, as with every Holy Grail, this one turns out to be a bit disappointing at times.

Definition 2.10 (Rays and extreme rays)

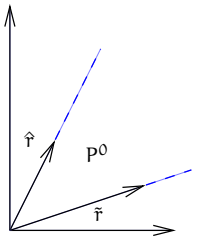
- *Any* $\mathbf{r} \in P^0 := \{\mathbf{r} \in \mathbb{R}^n : A\mathbf{r} \leq 0\}$ *with* $\mathbf{r} \neq \mathbf{0}$ *is called a* ***ray*** *of* $P = P(A, \mathbf{b}) = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\}$ *(where we assume* $P \neq \emptyset$*).*
- $\mathbf{r} \in P^0 \setminus \{\mathbf{0}\}$ *is an* ***extreme ray*** *of* $P(A, \mathbf{b})$ *if there do not exist rays* $\mathbf{r}^1, \mathbf{r}^2 \in P^0$ *such that* $\mathbf{r} = \frac{1}{2}\mathbf{r}^1 + \frac{1}{2}\mathbf{r}^2$ *unless* $\mathbf{r}^1 = \theta\mathbf{r}^2$ *for some* $\theta \in \mathbb{R}_+$.

The notion "ray" is justified, since

$$\mathbf{r} \in P^0,\ \mathbf{x} \in P \text{ implies } A(\mathbf{x} + \mu\mathbf{r}) = \underbrace{A\mathbf{x}}_{\leq \mathbf{b}} + \mu \underbrace{A\mathbf{r}}_{\leq \mathbf{0}} \leq \mathbf{b}\ \forall\ \mu \geq 0.$$



(a) Polyhedron P with two rays $\mathbf{r}^1$ and $\mathbf{r}^2$.



(b) Two extreme rays $\tilde{\mathbf{r}}$ and $\hat{\mathbf{r}}$ of polyhedron P^0.

Figure 2.4: Rays. Every unbounded polyhedron contains rays.

The polyhedral theory of IP contains many[6] theorems, propositions, lemmas, and corollaries that build on each other to allow us to conclude the following:

- **Corollary 2.11** *Every polyhedron has only a finite number of extreme points and extreme rays.*

  This corollary[7] states an important property of a polyhedron. It is a bit overdue since we have already drawn on it when discussing how Simplex method "moves" from one extreme point to another and eventually terminates.

- **Corollary 2.12** *If* $P = P(A, \mathbf{b}) = \{\mathbf{x} \in \mathbb{R}^n_+ : A\mathbf{x} \leq \mathbf{b}\}$ *is bounded, then* $\text{conv}(X) = \text{conv}(P \cap \mathbb{Z}^n)$ *is a rational polytope.*

- **Corollary 2.13** *The extreme rays of* P *and* $\text{conv}(X) = \text{conv}(P \cap \mathbb{Z}^n)$ *coincide.*

[6] I really mean *many*. In fact, so many, that when I was studying the polyhedral theory and then assisting in teaching it, I was constantly losing sight of the end goal behind the plethora of theorems and propositions. I compare it to the long, strenuous hikes I've done around Mont Blanc. You keep climbing and climbing, and by the time you get to the top you are so buggered you forget the purpose—to enjoy the most breathtaking view.

[7] I often find that corollaries are the most useful. Keeping up with my analogy of hiking up a mountain, when you are exhausted from hiking (proving all those theorems), a corollary is that last, tiny effort needed to lift your head and behold the view (understand the powerful consequence of the string of the theorems you have just proved).

- **Theorem 2.14 (Convex hull is a polyhedron)** *Let* $A, \mathbf{b}$ *be integral and* $P = P(A, \mathbf{b}) = \left\{\mathbf{x} \in \mathbb{R}^n_+ : A\mathbf{x} \leq \mathbf{b}\right\}$*, then* $\text{conv}(X) = \text{conv}(P \cap \mathbb{Z}^n)$ *is a rational polyhedron.*

  Bingo! This is the result we are after. Note that corollary 2.12 only allows me to conclude that $\text{conv}(X)$ is a rational polyhedron if $P = P(A, \mathbf{b})$ is *bounded*. Theorem 2.14 extends the result of the corollary to the unbounded cases of $P = P(A, \mathbf{b})$. The importance of this result is spelled out in the next theorem.

- **Theorem 2.15** *Given integral* A *and* $\mathbf{b}$*,* $P = P(A, \mathbf{b}) = \left\{\mathbf{x} \in \mathbb{R}^n_+ : A\mathbf{x} \leq \mathbf{b}\right\}$, $X = P \cap \mathbb{Z}^n \neq \emptyset$*, and* $\mathbf{c} \in \mathbb{R}^n$*, we consider two optimization problems:*

$$\begin{aligned} &\textit{(IP)} && z^{\text{OPT}}_{\text{IP}} = \max\left\{\mathbf{cx} : \mathbf{x} \in X\right\}. \\ &\textit{(LP)} && z^{\text{OPT}}_{\text{LP}} = \max\left\{\mathbf{cx} : \mathbf{x} \in \text{conv}(X)\right\}. \end{aligned}$$

  Then

  1. $z^{\text{OPT}}_{\text{IP}}$ *is of finite value if and only if* $z^{\text{OPT}}_{\text{LP}}$ *is.*
  2. *If* $z^{\text{OPT}}_{\text{LP}}$ *is finite then there exists an optimal solution* $\mathbf{x}^{\text{OPT}}$ *to (LP) which is optimal for (IP), i.e,* $\mathbf{x}^{\text{OPT}} \in X$*.*
  3. *If* $\mathbf{x}^{\text{OPT}}$ *is optimal for (IP) then it is optimal for (LP).*

With theorem 2.15 we have achieved our goal: In theory, it is possible to solve an IP as an LP. In practice (and that is where the ideal formulation comes in), an IP can be solved as an LP only if we know a linear description of $\text{conv}(X)$, i.e., if we know

$$\text{conv}(X) = \left\{\mathbf{x} \in \mathbb{R}^n : \tilde{A}\mathbf{x} \leq \tilde{\mathbf{b}}\right\}.$$

Obtaining a linear description of $\text{conv}(X)$ is, unfortunately, possible in only very few special cases, and that is why integer programs are difficult to solve. As seen in example 2.3, writing down the system of linear inequalities $P = P(A, \mathbf{b})$ explicitly is impossible for large n.

But I was not chasing this wild goose for nothing. There are IPs where $\text{conv}(X)$ is attainable, and I discuss these problems in section 2.2.

2.2 Integral Polyhedra

In this section we consider special IPs—those that can be easily solved. The reason these IPs are special is because the polyhedral descriptions $P(A, \mathbf{b})$ of their convex hulls are easy to find.

Throughout this chapter we assume that all elements of A and $\mathbf{b}$ are integer numbers.

Corollary 2.16 (Integral polyhedron) [8] *Polyhedron* $P \subseteq \mathbb{R}^n_+$ *is integral (or integer) if and only if all extreme points of* P *are integral.*

[8] The formal definition of an integral polyhedron is missing from these Lecture Notes due to the fact that I had to omit significant parts of polyhedral theory. Therefore, in these Lecture Notes this corollary serves as the definition of an integral polyhedron.

Lemma 2.17 *Let* $P = P(A, \mathbf{b}) \subseteq \mathbb{R}^n_+$ *be a formulation of* X. P *is the ideal formulation if and only if* P *is an integral polyhedron.*

Hence, if we could show that all extreme points of polyhedron $P = P(A, \mathbf{b})$ are integer points, then by theorem 2.15 we can solve the corresponding IP as an LP. However, proving something like that is not straightforward. An easier way to do this is via total unimodularity which I introduce next.

Total Unimodularity

The idea of total unimodularity (TU) is an important one because of the main result in theorem 2.23 which states that integral polyhedra and TU go hand-in-hand. The theorems and corollaries in this section provide tools to determine when an IP has an easily attainable integral polyhedron.

Definition 2.18 $m \times n$ *matrix A is* ***totally unimodular (TU)*** *() if* $\det(A') \in \{0, +1, -1\}$ *for all* $k \times k$ *submatrices of* A, $k = 1, 2, \ldots, \min\{m, n\}$.

Example 2.19

1. Matrix
$$A = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 1 \end{pmatrix}$$
is not TU since rows 2 and 4 and columns 2 and 4 define
$$A' = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$$
with $\det(A') = 2$.
2.
$$A = \begin{pmatrix} 0 & 1 \\ 3 & -1 \end{pmatrix}$$
is not TU since row 2 and column 1 define $A' = (3)$ with $\det(A') = 3$.

△

Lemma 2.20 A *is TU implies* $a_{ij} \in \{0, \pm 1\}$.

Is there a better way than checking all $\det(A')$ for all square submatrices A' of A?

Theorem 2.21 *If* A *is TU and* $\mathbf{b} \in \mathbb{Z}^m$ *such that* $P = \{\mathbf{x} \in \mathbb{R}^n_+ : A\mathbf{x} \leq \mathbf{b}\} \neq \emptyset$ *then* P *is integral.*

Corollary 2.22 *1.* A *is TU,* $\mathbf{b} \in \mathbb{Z}^m$ *and* $P = \{\mathbf{x} \in \mathbb{R}^m_+ : A\mathbf{x} = \mathbf{b}\} \neq \emptyset$*, then* P *is integral.*

2. A *is TU,* $\mathbf{b}, \mathbf{b}', \mathbf{d}, \mathbf{d}'$ *are integer vectors such that* $P = P(A, \mathbf{b}, \mathbf{b}', \mathbf{d}, \mathbf{d}') := \{\mathbf{x} \in \mathbb{R}^n : \mathbf{b}' \leq A\mathbf{x} \leq \mathbf{b},\ \mathbf{d}' \leq \mathbf{x} \leq \mathbf{d}\} \neq \emptyset$*, then* P *is integral polyhedron.*
3. A *is TU,* $\mathbf{c} \in \mathbb{Z}^n$*,* $Q(\mathbf{c}) := \{\boldsymbol{\mu} \in \mathbb{R}^m_+ : \boldsymbol{\mu}A \geq \mathbf{c}\} \neq \emptyset$*, then* $Q(\mathbf{c})$ *is integral polyhedron.*

Theorem 2.23 (Hoffmann and Kruskal, 1956) $P = \{\mathbf{x} \in \mathbb{R}^m_+ : A\mathbf{x} \leq \mathbf{b}\} \neq \emptyset$ *is integer for all* $\mathbf{b} \in \mathbb{Z}^m$ *if and only if* A *is TU.*

Corollary 2.24 A *is TU if and only if*

$$\forall\, Q \subseteq M = \{1, \ldots, m\} \quad \exists\, Q_1 \dot{\cup} Q_2 = Q \text{ such that}$$
$$\left| \sum_{i \in Q_1} a_{ij} - \sum_{i \in Q_2} a_{ij} \right| \leq 1 \quad \forall\, j = 1, \ldots, n$$

Corollary 2.25 *Let* A *be a* $(0, \pm 1)$*-matrix with the following properties:*

1. A *has at most two entries* $a_{ij} \neq 0$ *in each columns* $A_{\cdot j}$*.*
2. *If* $A_{\cdot j}$ *contains two entries* $a_{ij} \neq 0$ *, then* $\sum_{i=1}^m a_{ij} = 0$*.*

Then A *is TU.*

Lemma 2.26 *The following statements are equivalent:*

1. A *is TU.*
2. A^T *is TU.*
3. (A, I) *is TU.*[9]
4. $\tilde{A}$ *is obtained from* A *by multiplying a row (column) of* A *with* (-1) *is TU.*
5. $\tilde{A}$ *is obtained from* A *by interchanging two rows (columns) of* A *is TU.*
6. $\tilde{A}$ *is obtained from* A *by duplicating a row (column) of* A *is TU.*

Minimum Cost Network Flows

In this section I briefly introduce the minimal cost network flow problem and the basics of graph theory needed for this section as well as the examples and exercises which follow. (The material of this section is collated from (Bazaraa et al. 1990, chapter 9).)

Consider a **directed network** or a **digraph** G consisting of a finite set of **vertices** $V = \{v_1, \ldots, v_m\}$ (a.k.a. **nodes** and **points**) and **edges** $E = \{e_{ij}\}_{\forall\, i,j \in V}$ (a.k.a. **arcs** and **links**).

Throughout we assume that $|E| \geq 1$ and $|V| \geq 2$. Two nodes in G are **adjacent** if they are directly connected by an edge. A graph with edges that permit flow in both directions is called an **undirected** graph. Any undirected graph can be represented by a directed graph. (How?)

[9] Matrix notation (A, I) means that matrix A is augmented by an identity matrix I.

Because this section is concerned with minimizing cost of flow, another reasonable assumption is that $c_{ij} \geq 0$. (Why is it reasonable?)

Here are a few important concepts in network flow theory:

Path from node v_i to v_p is a sequence of edges $(v_i, v_{i+1}), (v_{i+1}, v_{i+2}), \ldots, (v_{i+k}, v_p)$ in which the initial node of each arc is the same as the end-node of the preceding arc, and $v_i, v_{i+1}, \ldots, v_p$ are all distinct nodes; moreover, each arc in the path is directed toward v_p in the case of a digraph.

Cycle is a closed path, i.e., a cycle is a sequence $(v_i, v_{i+1}), (v_{i+1}, v_{i+2}), \ldots, (v_{i+k}, v_p), (v_p, v_i)$; clearly, in a digraph edge (v_p, v_i) is directed toward v_i.

Tree is a connected graph with no cycles.

Consider a typical minimum cost network flow problem in example 2.27.

Example 2.27 ((Integral) Minimum Cost Network Flow (I-MINCOSTFLOW)) Given a digraph $G = (V, E)$ with cost c_{ij}, capacity $u_{ij} \in \mathbb{Z}$ for all $(i, j) \in E$, and supplies/demands $b_i \in \mathbb{Z}$ for all $i \in V$ (with $\sum_{i \in V} b_i = 0$), find a minimum cost flow $\mathbf{x} = (x_{ij})_{(i,j) \in E}$ with integral x_{ij} for all $(i, j) \in E$.

Let x_{ij} be the amount of flow on arc $(i, j) \in E$. Then the formulation is as follows:

$$\begin{aligned}
\min \quad & \mathbf{cx} && && (2.10a)\\
\text{s.t.} \quad & \sum_{(i,j) \in E} x_{ij} - \sum_{(j,i) \in E} x_{ji} = b_i && \forall i \in V && (2.10b)\\
& \mathbf{x} \leq \mathbf{u} && && (2.10c)\\
& \mathbf{x} \geq \boldsymbol{\lambda} && && (2.10d)\\
& x_{ij} \in \mathbb{Z}_+ && \forall (i, j) \in E, && (2.10e)
\end{aligned}$$

where $b_i > 0$ if i is a supply node, $b_j < 0$ for demand nodes, and $b_i = 0$ if i is a transshipment node; $\boldsymbol{\lambda} \geq \mathbf{0}$. △

Constraints (2.10b) are usually written as $A\mathbf{x} = \mathbf{b}$, where A is the **node-arc incidence matrix** of G. The structure of A is crucial: A has one row for each node of the network and one column for each arc. Moreover, each column of A contains exactly two non-zero elements: $+1$ in in the row corresponding to the *from-node* of an arc under consideration and -1 in the row of the *to-node* of the arc. Matrix A for graph 2.2 is then the following:

	(a,b)	(a,c)	(b,d)	(b,e)	(c,d)	(c,e)	(d,f)	(e,f)
a	1	1	0	0	0	0	0	0
b	-1	0	1	1	0	0	0	0
c	0	-1	0	0	1	1	0	0
d	0	0	-1	0	-1	0	1	0
e	0	0	0	-1	0	-1	0	1
f	0	0	0	0	0	0	-1	-1

and

$$A \begin{pmatrix} x_{ab} \\ x_{ac} \\ x_{bd} \\ x_{be} \\ x_{cd} \\ x_{ce} \\ x_{df} \\ x_{ef} \end{pmatrix} = \begin{pmatrix} 9 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -9 \end{pmatrix}. \tag{2.11}$$

Matrix A has the special structure that according to corollary 2.25 allows us to conclude that A is TU. Then by theorem 2.23 we can conclude that the formulation polyhedron of MINCOSTFLOW is integral. Therefore, by theorem 2.15 MINCOSTFLOW can be solved as an LP.

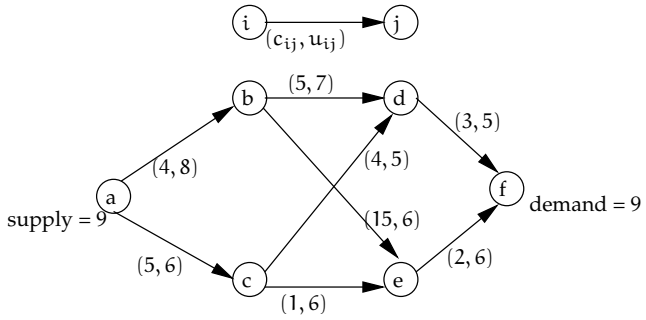


Figure 2.5: A graph for a network flow problem. Each arc/edge has cost c_{ij} and capacity u_{ij}. Node a is the supply node with 9 units and node f is the demand node.

There are a few types of IPs that can be formulated as MINCOSTFLOW problems, for example, the famous **assignment problem** presented next.

Example 2.28 (Assignment Problem (AP))
Given n jobs, n workers and cost c_{ij} of assigning a job j to a worker i, find an assignment of jobs to workers such that: every individual does exactly one job and the overall cost is minimized.

The decision variable is:

$$x_{ij} = \begin{cases} 1 & \text{if worker } i \text{ is assigned to job } j \\ 0 & \text{else.} \end{cases}$$

$$\min \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.12a}$$

$$\text{s.t.} \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i = 1, \ldots, n \tag{2.12b}$$

$$\sum_{j=1}^{n} x_{ji} = 1 \qquad \forall i = 1, \ldots, n \tag{2.12c}$$

$$x_{ij} \in \mathbb{B} \qquad \forall i, j = 1, \ldots, n \tag{2.12d}$$

Constraints (2.12b) ensure that every worker does a single job, whereas constraints (2.12c) guarantee that every job is done by exactly one worker.

△

The formulation polytope of the AP in example 2.28 is integral. Therefore, even though AP is an IP it can be solved as an LP with Simplex method, and the solution is guaranteed to be integer.

2.3 A few Words about Computational Complexity Theory

In section 2.2 I discussed IPs which are easy, i.e., those that can be solved as LPs. Unfortunately, most IPs one encounters in the real world are not easy. In fact, the difficulty of these IPs has so exasperated researchers in many fields of mathematics, computer science and even physics, that a whole new subfield has been constructed—the theory of computational complexity.

Complexity theory has been around since the 1920s or so when scientists started thinking about computers (at first, just in theory).

A problem is considered to be *easy* if it can be solved in polynomial time with respect to the number of its variables and constraints. A set of such easy problems is denoted by P. After it was shown that LPs can be solved in polynomial time (see section 1.3) and, thus, are easy, scientists turned their attention to IPs.

A problem is considered to be in class **NP-complete** if it is a decision problem with a property that any given solution can be verified in polynomial time, but there is no known polynomial-time algorithm to find a solution.[10]

An example of a *decision* problem for the 0-1 KNAPSACK is as follows:

Given: $n \in \mathbb{N}$ (where $\mathbb{N}$ denotes the set of non-negative integers) items with weights $a_1, \ldots a_n \in \mathbb{N}$, costs $c_1, \ldots c_n \in \mathbb{N}$, weight $W \in \mathbb{N}$ and $\gamma \in \mathbb{N}$

Question: Does there exist $K \subseteq \{1, \ldots, n\}$ such that $\sum_{k \in K} c_k \geq \gamma$ and $\sum_{k \in K} a_k \leq W$?

[10] This is a very informal definition of NP-completeness, but I deem it sufficient. If you want to learn more about NP-completeness, check out *Wikipedia*. If you are really hard core, then see Garey and Johnson (1979).

The optimization-problem counterpart of the above decision problem is the 0-1 KNAPSACK *optimization* problem:

Given: $n \in \mathbb{N}$ items with weights $a_1, \ldots a_n \in \mathbb{N}$, costs $c_1, \ldots c_n \in \mathbb{N}$ and weight $W \in \mathbb{N}$

Find: $K \subseteq \{1, \ldots, n\}$ such that $\sum_{k \in K} c_k$ is maximal and $\sum_{k \in K} a_k \leq W$.

Definition 2.29 *A partition of a set* $\{1, \ldots, n\}$ *into sets* $I, J \subseteq \{1, \ldots, n\}$ *is called a* ***disjoint partition*** *if* $I \cup J = \{1, \ldots, n\}$ *and* $I \cap J = \emptyset$.

Examples of well-known decision problems:

- PARTITION
  Given: Integers $a_1, \ldots, a_n \in \mathbb{N}$
  Question: Does there exist a disjoint partition I and J of set $\{1, \ldots, n\}$ so that $\sum_{i \in I} a_i = \sum_{j \in J} a_j$?

- SUBSETSUM
  Given: Integers $b_1, \ldots, b_n \in \mathbb{N}$ and $B \in \mathbb{N}$
  Question: Does there exist $K \subseteq \{1, \ldots, n\}$ so that $\sum_{k \in K} b_k = B$?

- BINPACKING
  Given: Rational numbers $s_1, \ldots, s_n \in \mathbb{Q}$, with $0 \leq s_i \leq 1 \; \forall i = 1, \ldots n$, and $L \in \mathbb{N}$
  Question: Does there exist a disjoint partition $K_1, \ldots, K_L$ of $\{1, \ldots, n\}$ so that $\sum_{i \in K_j} s_i \leq 1 \; \forall j = 1, \ldots, L$?

- 3SAT (3-Satisfiability)
  Given: Collection $C = \{c_1, c_2, \ldots, c_m\}$ of clauses on a finite set U of variables such that $|c_j| = 3$ with $1 \leq j \leq m$
  Question: Is there a truth assignment for U that satisfies all the clauses in C?

- VERTEXCOVER (also known as Node Cover)
  Given: Graph $G = (V, E)$ and $\gamma \in \mathbb{N}$ such that $\gamma \leq |V|$
  Question: Does there exist $V' \subseteq V$ such that each edge in E has at least one of its endpoints in V' and $|V'| \leq \gamma$?

You have probably heard about the class of **NP-hard** (**N**on-deterministic **P**olynomial-time hard) problems. There is a subtle difference between NP-completeness and NP-hardness: An optimization problem is in class NP-hard if its corresponding decision problem is in class NP-complete. Hence, if a decision problem could be solved in polynomial time, then the corresponding optimization problem would also be polynomially solvable. (Why?)

The main results of computational complexity theory with respect to IP are the following:

1. IPs can be divided into two main groups: those that can be solved in polynomial time and are in class P, e.g., I-MINCOSTFLOW, and those which are in set NP-hard.
2. IPs that are NP-hard can be transformed into one another. The reason for that is that their underlying decision problems are in essence the same,

e.g., it can be shown that PARTITION is a special case of the KNAPSACK decision problem. (See exercise 2.12.)

Therefore, if one can solve one of the problems in class NP-hard in polynomial time, then all NP-hard problems are easy to solve.

One of the million-dollar questions is whether P equals NP (set NP contains set NP-hard). Since August 6, 2010, the scientific community is digesting a proof by Vinay Deolalikar, a mathematician at Hewlett-Packard Labs in Palo Alto, California, which claims that $\mathsf{P} \neq \mathsf{NP}$. See

`http://www.newscientist.com/article/dn19287-p--np-its-bad-news-for-the-power-of-computing.htm`

If the proof is correct, we can abandon the idea of solving many interesting IPs in polynomial time until the coming of quantum computers.[11] But in the meantime, we must try to find ways to deal with IPs which are NP-hard.

Believe it or not, even though the KNAPSACK optimization problem looks very innocent, it is NP-hard.

Example 2.30 (Generalized Assignment Problem (GAP))
Given n jobs, m workers and cost c_{wj} of assigning a job j to a worker w, find the cheapest assignment of jobs to workers if a worker can be assigned an arbitrary number of jobs.

However, there is a wrinkle: there is a limited number of hours in a time period (e.g., day, week). Hence, just because worker w is best suitable to perform job j, he/she may not be able to due to his/her availability. Let h_j be the number of hours consumed by job j, and let Ω_w denote the number of hours that worker w is available.

$$x_{wj} = \begin{cases} 1 & \text{if worker } w \text{ is assigned to job } j \\ 0 & \text{else.} \end{cases}$$

$$\min \quad \sum_{j=1}^{n} \sum_{w=1}^{m} c_{wj} x_{wj} \tag{2.13a}$$

$$\text{s.t.} \sum_{w=1}^{m} x_{wj} \geq 1 \qquad \forall j = 1, \ldots, n \tag{2.13b}$$

$$\sum_{j=1}^{n} h_j x_{wj} \leq \Omega_w \qquad \forall w = 1, \ldots, m \tag{2.13c}$$

$$x_{wj} \in \mathbb{B} \qquad \forall w = 1, \ldots, m, \ \forall j = 1, \ldots, n \tag{2.13d}$$

Constraints (2.13b) ensure that every job gets done. Constraints (2.13c) make sure that each worker's capacity is respected. Other types of restrictions can be added as well. △

[11] That's good news for people like myself because I will always have a job solving those difficult IPs.

I have already discussed why AP is in set P. Unfortunately GAP is NP-hard. (It is not a formal proof, but an indication of NP-hardness of GAP is its resemblance of the KNAPSACK.)

The brief introduction to computational complexity theory I have given herein does not even begin to scratch the surface of it. An excellent (but not easy) read about complexity is Garey and Johnson (1979), where you also discover that even among the problems which are NP-hard there are differences in complexity.

2.4 Exercises

Exercise 2.1 (Uncapacitated Facility Location Problem (UFL)) *Which depots from the set $\mathcal{N} = \{1, \ldots, n\}$ should be opened (each at a fixed price f_j for $j \in \mathcal{N}$) to satisfy the demand of customers in $\mathcal{M} = \{1, \ldots, m\}$ (at cost c_{ij} if the* complete *demand of $i \in \mathcal{M}$ is satisfied from $j \in \mathcal{N}$). The goal is to minimize the overall cost. Depots have unlimited capacity.*

1. *Formulate as an MIP. How many variables and constraints does your formulation contain?*
2. *Define the resulting formulation polyhedron.*
3. *Could you find another formulation for* UFL*? Are the two formulations comparable? If so, which one is better?*

Exercise 2.2 (A transportation problem as a MINCOSTFLOW) *Given a set of origins and destinations, the cost of transporting goods from origin o_i to destination d_j as well as demands and supply at the origins and destinations, respectively are given in table 2.1*

	d_1	d_2	d_3	supply
o_1	2	1	1	3
o_2	1	4	4	5
demand	1	3	4	

Table 2.1: Transportation costs, demand, and supply for exercise 2.2.

a) *Formulate this transportation problem as a* MINCOSTFLOW *where variables do not have to be integral.*

b) *Draw a graph of the corresponding network.*

Exercise 2.3 (WAM) *Consider time horizon T consisting of weeks $t_1, t_2, \ldots, t_n$. Let $\mathcal{I}_t$ be the set of interviewers available in week t of the horizon T. During T interviewers must perform enumeration of blocks of dwellings for different surveys. Let $\mathcal{B}_t$ be the set of MPS blocks which must be enumerated in MPS week $t \in T_{MPS}$. (For the MPS enumeration week of each block is specified.) If T is longer than a month, then it can include a few MPS weeks (two predetermined weeks each month) and MPS follow-up weeks (usually 2.5 weeks every month*

starting with the second MPS week). Subset T_{MPS} *denotes the MPS weeks;* T_F *is the set of the MPS follow-up weeks.*

Let the set of Special Social Surveys (SSSs) be denoted by $\mathcal{L}$*. Denote by* T_l *the set of weeks in which social survey* $l \in \mathcal{L}$ *must take place. Let set* $\mathcal{S}_l$ *be the set of blocks for SSS* l*.*

Let W^{max} *be the maximum number of hours an interviewer can work during a week.*

Let M_m *be a set of weeks comprising month* m*, thus* $T = \dot{\cup}_{\forall m} M_m$*. By their Enterprise Agreement, interviewers are guaranteed one week of work during a month, where one week of work comprises of* W^{min} *hours.*

Denote the time that interviewer i *spends interviewing (enumerating) all dwellings in block* b *by* ω_b*, which depends on the number of dwellings in a block. However, to allow for differentiating between interviewers' efficiencies,* ω_{ib} *is considered instead of* ω_b*.*

The cost of interviewer i *doing block* b *has three components: distance* $dist_{ib}$ *which is travelled by interviewer* i *to get to block* b*, time* ω_{ib} *to interview dwellings in block* b*, and time* τ_{ib} *to get to block* b*. Coefficient* p_{ib} *is a function of* $dist_{ib}$*,* ω_{ib} *and* τ_{ib} *converted to dollar value.*

Find the cheapest allocation of interviewers to blocks to be enumerated for MPS and all SSSs, taking into account the following:

1. *Each MPS block must be enumerated in its prescribed enumeration week.*
2. *Blocks for each SSS* l *must be enumerated during the time period* T_l*.*
3. *An interviewer can only work during the prime hours, hence the number of total hours worked is at most* W^{max}*.*
4. *Each month there is a set of blocks that are in their first month in MPS. An interviewer cannot be assigned more than* N *of dwellings that are in those special blocks.*

What changes must be made to the above formulation if:

1. *There is a difference in interviewer efficiencies, i.e., the time to enumerate a block differs between interviewers.*
2. *The time an interviewer is available to work each week changes from week to week and from interviewer to interviewer.*
3. *Due to insufficient data, the only thing we have available is the distance between a block and interviewer.*
4. *Each interviewer must be given at least* W^{min} *hours of work per month.*
5. *Under the new EA interviewers are requesting that they will be given at least* Ω *hours with at least* Γ *hours for two weeks (with* $\Omega > 2\Gamma$*).*

Exercise 2.4 (UNCAPACITATEDLOTSIZING problem (from Pochet and Wolsey (2000))) *A manufacturer produces a wide variety of gadgets. We are interested in the production plan of a high-tech model gadget-X whose production requires special materials and equipment. At most one batch of gadget-X is produced per*

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
400	400	800	800	1200	1200	1200	1200	800	800	400	400

Table 2.2: Demand for gadget-X. (Data for exercise 2.4.)

month as the set-up cost q *is very high. The cost of production is* c *monetary units per gadget-X. The demand for gadget-X for the next year is given in table 2.2. There will be* σ *gadgets-X in stock at the end of the current year. To hold a gadget-X in inventory during one month costs on average* h *monetary units. As the name of the model suggests, we assume that the production and storage are uncapacitated. The manufacturer wants to plan the production and inventory levels of gadget-X in order to satisfy demand and maximize profit.*

Assume that $q = 5000$, $c = 100$, $h = 5$, $\sigma = 200$, *revenue* p *per gadget-X is 150, and at the end of the year there should be no units of gadget-X remaining in the inventory.*

1. *Calculate the cost of an option where the entire demand is produced at the beginning of the year.*
2. *Calculate the cost of an option when the demand of each month is produced in that month.*
3. *Formulate the problem as an IP. Solve.*

Exercise 2.5 (UNCAPACITATEDLOTSIZING with pricing for production and delivery) *Consider* UNCAPACITATEDLOTSIZING *problem from exercise 2.4. Suppose that the manufacturer offers a price discount* $\alpha \in [0, 1]$ *for the flexibility to deliver a fraction of the demand* d_t *either in period* t *itself or in any of the periods* $\{t + 1, \ldots, t + \Phi\}$ *where* $\Phi \geq 1$. *Denote the flexible demand in each period by* d_t^f *with* $d_t^f \leq \Delta_t$, *where* Δ_t *is determined by the client of the manufacturer. Discount factor* α *is set as the result of negotiations between the manufacturer and the client. How should the manufacturer plan the production and inventory levels of gadget-X in order to maximize profit?*

Exercise 2.6 (from Bazaraa et al. (1990)) *Formulate and solve using Gurobi exercise 1.5 p. 27 of Bazaraa et al. (1990). (copy and distribute)*

Exercise 2.7 (A simplified GAP) *Given* n *jobs,* m *workers and cost* c_{wj} *of assigning a job* j *to a worker* w, *find the cheapest assignment of jobs to workers. A worker can be assigned an arbitrary number of jobs.*

a) Formulated this problem as an IP.

b) Show that the formulation polytope is integral.

Exercise 2.8 (AP as MINCOSTFLOW) *Show that AP in example 2.28 can be formulated as a* MINCOSTFLOW, *and thus its formulation polytope (why polytope?) is integral.*

Exercise 2.9 *Prove that a rational polyhedron (defined in 2.5) is a convex set.*

Exercise 2.10 (Bad rounding) *Find an example of IP where solving it as an LP and then rounding leads to a sub-optimal solution.*

Exercise 2.11 (NP-completeness vs. NP-hardness) *Prove that if a decision problem can be solved/answered in polynomial time, then so can be its corresponding optimization problem.*

Exercise 2.12 (SUBSETSUM vs. KNAPSACK (decision problem)) *Show that* SUBSETSUM *is a special case of* KNAPSACK *decision problem.*

CHAPTER 3

Integer Programming: Solution Techniques

In light of the discussion in section 2.3, this section is concerned with the methods for tackling difficult IPs.

Going back to the idea of solving an IP as an LP, solve a relaxed IP[1]

$$\begin{aligned} \textbf{P1:} \quad & \min & \mathbf{cx} & & (3.1a)\\ & \text{s.t.} & A\mathbf{x} = \mathbf{b} & & (3.1b)\\ & & \mathbf{x} \leq \mathbf{1} & & (3.1c)\\ & & \mathbf{x} \geq \mathbf{0} & & (3.1d) \end{aligned}$$

instead of the original IP

$$\begin{aligned} \min \quad & \mathbf{cx} & & (3.2a)\\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b} & & (3.2b)\\ & \mathbf{x} \in \mathbb{B}^n & & (3.2c) \end{aligned}$$

More than likely, an optimal $\mathbf{x}_{LP}^{OPT}$ solution to (3.1) is non-integer, hence there exists a component x_i of $\mathbf{x}_{LP}^{OPT}$ such that $0 < x_i < 1$, w.l.o.g.[2], let $i = 1$.

One can enforce integrality on variable x_1 by solving the following two problems:

$$\begin{aligned} \textbf{P2:} \quad & \min & \mathbf{cx} & & (3.3a)\\ & \text{s.t.} & A\mathbf{x} = \mathbf{b} & & (3.3b)\\ & & \mathbf{x} \leq \mathbf{1} & & (3.3c)\\ & & \mathbf{x} \geq \mathbf{0} & & (3.3d)\\ & & x_1 \leq 0 & & (3.3e) \end{aligned}$$

[1] Remember, it can be shown that an IP is equivalent to a BIP.
[2] w.l.o.g. means *without loss of generality*

$$\begin{aligned}
\textbf{P3:} \quad & \min & \mathbf{cx} & \quad (3.4a)\\
& \text{s.t.} & A\mathbf{x} = \mathbf{b} & \quad (3.4b)\\
& & \mathbf{x} \leq \mathbf{1} & \quad (3.4c)\\
& & \mathbf{x} \geq \mathbf{0} & \quad (3.4d)\\
& & x_1 \geq 1 & \quad (3.4e)
\end{aligned}$$

In the case of the relaxed IP (3.3), constraints (3.3d) and (3.3e) force x_1 to take on value 0. Analogously, in the case of the relaxed IP (3.4), constraints (3.4d) and (3.4e) force x_1 to take on value 1.

It can transpire that either (3.3) or (3.4) is infeasible. (Can both be infeasible?) Suppose (3.3) is feasible, but its optimal solution is again not integral. And so our process of enforcing integrality continues.

3.1 Branch-and-Bound

The process described above is the basis for the method called **branch-and-bound (BB)**. And a common way to picture this method is through a binary **enumeration tree**, sometimes called the **branch-and-bound tree**.

Following up on the discussion above, suppose both relaxed IPs (3.3) and (3.4) are feasible, but their optimal solutions are non-integral. W.l.o.g., $0 < x_2 < 1$ for problem (3.3), and $0 < x_3 < 1$ for problem (3.4). Then we must solve the following four relaxed IPs:

$$P4 = P2 \cup \{x_2 \leq 0\} \quad (3.5)$$

$$P5 = P2 \cup \{x_2 \geq 1\} \quad (3.6)$$

$$P6 = P3 \cup \{x_3 \leq 0\} \quad (3.7)$$

$$P7 = P3 \cup \{x_3 \geq 1\} \quad (3.8)$$

The BB tree for this example is depicted in figure 3.1.

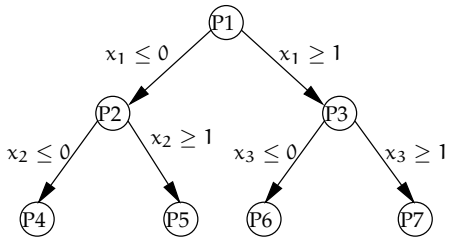


Figure 3.1: The branch-and-bound tree.

Clearly, as a BB tree continues to grow and the further we move down the branches the more variables are forced to be integers. Finally, at the end-nodes of a BB tree—the **leaves**—all variables are forced to be integers. The crucial thing to note here is that at each node of a BB tree an LP, not an IP, is solved. Obviously, as we go deeper down a BB tree, the number of constraints increases, so LPs are becoming larger and more difficult (i.e., more time- and computer memory-consuming) to solve. Usually a BB method combines searches in both directions, depth and breadth, along a BB tree, hence a couple of LPs have to be stored in computer memory simultaneously.

Fortunately, some branches of the tree can be pruned and, thus, not needed to be explored all the way to the leaves. One obvious reason for pruning is infeasibility. There are two other types of pruning—by bound and by optimality.

Researches have explored different rules for branching and tried answering the following questions:

- Which variable x_i to branch on first when more than one is non-integral?
- What type of branching should be used: depth-first or breadth-first?
- What other techniques can be used to prune a BB tree more efficiently?

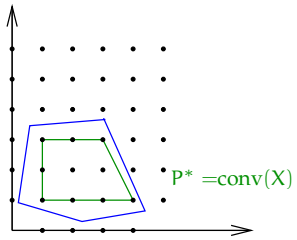
In my opinion, there are no hard rules. The answers to the above questions are problem-dependent, and that is what makes IPs so challenging.[3]

As for the techniques to improve solving efficiency, researchers have considered adding different types of inequalities—**cuts**—in order to accelerate movement through a BB tree. Cuts do what their name suggests, they cut off pieces of IP's formulation polyhedron that do not contain feasible (which, in the case of IPs, means integer) solutions. Figures 3.2(a)–3.2(e) display a progression of cuts and how formulation polytopes decrease in size with addition of each cut.

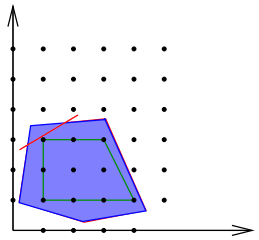
Some cuts have very little OR theory behind them: they are very simple logical constraints. (To get a taste of logical cuts, do exercise 3.1.) Other cuts are much more complicated. Because cuts can be very useful, Nemhauser and Wolsey (1999) and Wolsey (1998) devote a fair amount of attention to different types of cuts. Because many cuts apply only to certain types of IPs, I choose to say very little about them herein. However, I would like to highlight three important points:

- Quoting directly from Wolsey (1998), "A **branch-and-cut algorithm** is a branch-and-bound algorithm in which cutting planes are generated throughout the branch-and-bound tree. Though this may seem to be a minor difference, in practice there is a change of philosophy. Rather than re-optimizing at each node [of a BB tree], the new philosophy is to do as much work as is necessary to get a tight dual bound at the node." However, adding cuts has the obvious disadvantage of increasing the number of constraints; hence, the process of solving an LP at each node of a tree becomes slower.

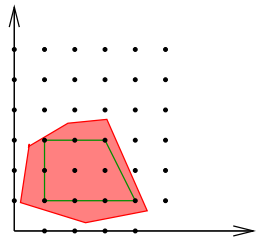
[3] Plus it gives academics plenty of publishing material even though sometimes not very worthy of publishing.



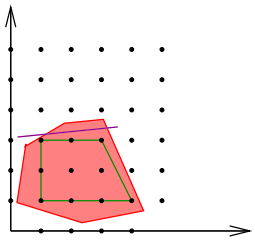
(a) Starting formulation polytope in blue, and conv(X) in green.



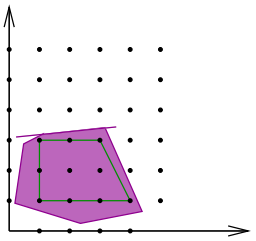
(b) A first cut is introduced. It cuts off a piece of the blue formulation polytope.



(c) The new formulation polytope in red.



(d) Another cut (in purple) is introduced. It cuts off a piece of the red formulation polytope.



(e) The new formulation polytope in purple.

Figure 3.2: Cuts are introduced one by one. The cuts tighten the formulation polytope by cutting off pieces of the original formulation polytope. The idea is that as cuts are being added the sequence of formulation polytopes converges to the conv(X) around the optimal point.

- When you encounter a difficult IP (i.e., your solver is falling over due to a large size of the resulting BB tree) and you are keen to get your hands dirty with proofs, look into the topic of cuts.
- Both algorithms,—the branch-and-bound and branch-and-cut,—are complete enumeration methods. In some cases many branches can be cut and pruned. But in general, there are not many shortcuts, and almost-complete enumeration is necessary—and therein lies the beauty[4] of IP.

When you watch the log output as Gurobi is solving an IP, you can obtain some[5] information on how the search through a BB tree is progressing.

After a problem is solved, Gurobi reports on some well-known cuts it utilized during the search through a BB tree, e.g., *clique* inequalities.

Another valuable piece of information a Gurobi log provides is the best **incumbent solution**—an integral solution with the best objective function value at

[4] But it depends on your definition of beauty, I suppose.

[5] Gurobi is a fairly expensive software which contains tricks of the trade. Obviously, we do not know the exact techniques Gurobi optimizer is employing, but we have an idea.

any given time during a BB tree search. Obviously, it might take time for a solver to obtain such an incumbent solution.

What the solver can do from the start is to obtain the **bound**—the value of the objective function after solving a relaxed IP at the root node of a BB tree, provided the solver can handle the LP. The bound improves (i.e., gets larger for minimization problems and smaller for maximization problems) as more nodes are examined. (Why?) Simultaneously, the objective function value of the best incumbent solution also improves (but in this case, OFV decreases for a minimization problem and increases for a maximization problem). Hence the gap between the best bound and the OFV of the best incumbent decreases. That gap is also reported in the Gurobi log. The first time the gap appears is when the first incumbent solution is available.

In summary, the two exact methods for solving IPs are composed of solving numerous LPs.

3.2 Lagrangean Relaxation

In section 3.1 I described exact methods for solving IPs. In this section I attempt to explain one of the most well-known non-exact methods—the **Lagrangean relaxation**.[6]

Consider the following IP:

$$
\begin{aligned}
\Pi: \quad & \\
& z_{\Pi}^{\text{OPT}} = \max && \mathbf{cx} && \text{(3.9a)}\\
& \text{s.t.} && A\mathbf{x} \leq \mathbf{b} && \text{(3.9b)}\\
& && D\mathbf{x} \leq \mathbf{d} && \text{(3.9c)}\\
& && \mathbf{x} \in \mathbb{Z}_{+}^{n}. && \text{(3.9d)}
\end{aligned}
$$

Suppose constraints $A\mathbf{x} \leq \mathbf{b}$ are "easy," e.g., A is a TU matrix. That is, suppose that

$$X = \{A\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \in \mathbb{Z}_{+}^{n}\}$$

and

$$\text{conv}(X) = \{A\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\}.$$

Then we know that solving IP

$$
\begin{aligned}
\max \quad & \mathbf{cx}\\
\text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b}\\
& \mathbf{x} \in \mathbb{Z}_{+}^{n}
\end{aligned}
$$

[6] Just because Lagrangean relaxation is well-known, it does not mean it is the most effective non-exact method. I discuss my reservations and the reason for the hype around this method at the end of the section.

is the same as solving LP

$$\begin{aligned} \max \quad & \mathbf{cx} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}. \end{aligned}$$

Define set $Y = X \cap \{D\mathbf{x} \leq \mathbf{d}\}$. Assume that adding constraints $D\mathbf{x} \leq \mathbf{d}$ into X destroys the nice integrality property, i.e.,

$$\text{conv}(Y) \neq \{A\mathbf{x} \leq \mathbf{b},\ D\mathbf{x} \leq \mathbf{d},\ \mathbf{x} \geq \mathbf{0}\}.$$

Hence, $D\mathbf{x} \leq \mathbf{d}$ are often referred to as *complicating* constraints. Thus, it would be preferable to ignore them; however, it is not an option if we want to solve the original problem. Instead, these constraints are enforced by **dualization** in the following way: the constraints are included into the original objective function with a penalty as in equation (3.12a).

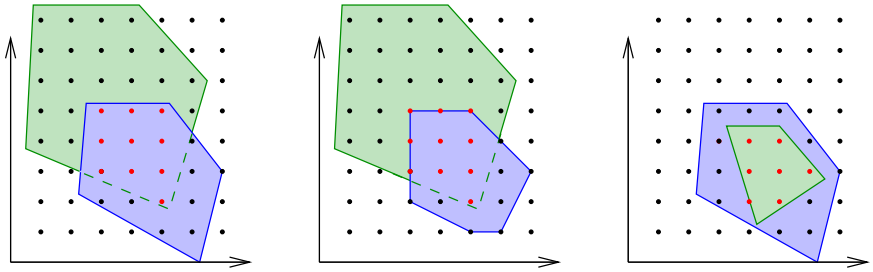
$\Pi(\boldsymbol{\lambda})$

$$z(\boldsymbol{\lambda}) = \max \quad \mathbf{cx} + \boldsymbol{\lambda}(\mathbf{d} - D\mathbf{x}) \tag{3.12a}$$

$$\text{s.t.} \quad \mathbf{x} \in X \tag{3.12b}$$

where $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_m) \geq 0$ assuming that D is an $m \times n$ matrix.

Scalars λ_i's are called **Lagrangean multipliers**, and $\Pi(\boldsymbol{\lambda})$ is called a **Lagrangean relaxation** of Π with parameter $\boldsymbol{\lambda}$.



(a) (b) A situation when $\text{conv}(X) = \{A\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\}$. (c)

Figure 3.3: Constraints $D\mathbf{x} \leq \mathbf{d}$ form the green polytope while constraints $A\mathbf{x} \leq \mathbf{b}$ form the blue one. Red points are the feasible solutions to Π.

It is important to convince ourselves of the following:

Proposition 3.1 *$\Pi(\boldsymbol{\lambda})$ is a relaxation of the original problem Π.*

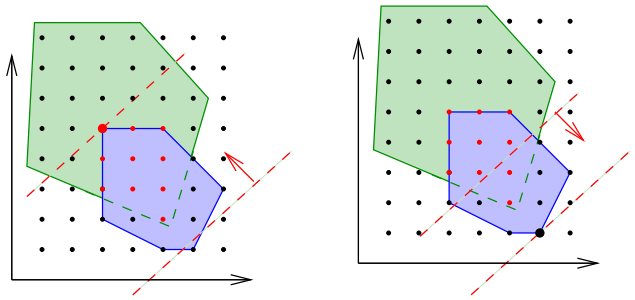
Proof: First, it is clear[7] that the feasible region of $\Pi(\boldsymbol{\lambda})$ is larger than the feasible region of Π. Second, for any feasible solution $\mathbf{x}$ to problem Π, the objective function value of $\Pi(\boldsymbol{\lambda})$ based on $\mathbf{x}$ is greater or equal to the objective function value of Π. (Why?) ■

[7] If it is not clear, think back to exercise 1.7.

Figures 3.3(a)–3.3(c) show different situations of what happens to the feasible region during Lagrangean relaxation.

Because $\Pi(\boldsymbol{\lambda})$ is a relaxation of Π, $z(\boldsymbol{\lambda}) \geq z_{\Pi}^{\text{OPT}}$, hence $z(\boldsymbol{\lambda})$ is an upper bound on the optimal objective value of Π. By the duality principle, we try to find such $\boldsymbol{\lambda}$ so that $z(\boldsymbol{\lambda})$ is as small as possible and, hence, provides the best bound on z_{Π}^{OPT}. Therefore the **Lagrangean Dual (LD)** of integer program Π is defined as

$$w_{\text{LD}} = \min \{z(\boldsymbol{\lambda}) : \boldsymbol{\lambda} \geq \mathbf{0}\}.$$



(a) A lucky situation when $\widetilde{\mathbf{c}}$ delivers a solution within X which is also feasible for Π. As the result, the big red point is an optimal solution to Π.

(b) Picture of a situation when $\widetilde{\mathbf{c}}$ delivers a solution to $\Pi(\boldsymbol{\lambda})$ which is in X but not in the feasible region of Π.

Figure 3.4: When solving $\Pi(\boldsymbol{\lambda})$ for a given $\boldsymbol{\lambda}$, we effectively solve $\max \{\widetilde{\mathbf{c}}\mathbf{x} - \boldsymbol{\lambda}\mathbf{d} : \mathbf{x} \in X\}$, where $\widetilde{\mathbf{c}} = \mathbf{c} - \boldsymbol{\lambda}D$. Hence, we optimize over points in the blue polytope. The dashed red line is the projection of $\tilde{\mathbf{c}}$ with the red arrow being its gradient.

Solving Lagrangean relaxation $\Pi(\boldsymbol{\lambda})$ may *sometimes* produce a solution $\mathbf{x}(\boldsymbol{\lambda})$ which is optimal for the original problem Π. The necessary conditions for that are stated in the following proposition:

Proposition 3.2 *If* $\boldsymbol{\lambda} \geq 0$ *and*

1. $\mathbf{x}(\boldsymbol{\lambda})$ *is an optimal solution of* $\Pi(\boldsymbol{\lambda})$*, and*
2. $D\mathbf{x} \leq \mathbf{d}$*, and*
3. $\big(D\mathbf{x}(\boldsymbol{\lambda})\big)_i = d_i$ *whenever* $\lambda_i > 0$*,*

then $\mathbf{x}(\boldsymbol{\lambda})$ *is an optimal solution to* Π*.*

Condition (3) of the proposition 3.2 just ensures that complementary slackness conditions are satisfied (see theorem 1.3).

The main result of Lagrangean Duality is stated in the following theorem:

Theorem 3.3 (Lagrangean Dual Bound)

$$w_{\text{LD}} = \max \{\mathbf{c}\mathbf{x} : \ D\mathbf{x} \leq \mathbf{d},\ \mathbf{x} \in \text{conv}(X)\}.$$

Theorem 3.3 shows the strength of the bound one can obtain from dualization.

The tricky part of the LD is finding good λ. A generic algorithm is presented in 1.

Algorithm 1 Lagrangean Dual Algorithm

LAGRANGEAN DUAL ALGORITHM

Input: Initial value λ_0, problem Π

1 **for** $i = 0, 1, \ldots$ **do**
2 solve $\Pi(\lambda_i)$ and obtain solution $\mathbf{x}(\lambda_i)$
3 **if** $\mathbf{x}(\lambda_i)$ satisfies the conditions of proposition 3.2 **then**
4 stop
5 **else**
6 $\lambda_{i+1} = \max\{\lambda_i - \mu_i(\mathbf{d} - D\mathbf{x}(\lambda_i)), 0\}$
7 **end if**
8 **end for**

Output: either an optimal solution to Π, or the best bound w_{LD}

Though algorithm 1 seems simple and straightforward, it omits one of the most important issues—how to choose the step length $\mu_i \ \forall i = 0, \ldots$. Theoretically, as per theorem 3.4, algorithm 1 converges.

Theorem 3.4 (Convergence of the Lagrangean Dual Algorithm)

1. *If* $\sum_i \mu_i \to \infty$ *and* $\mu_i \to 0$ *as* $i \to \infty$, *then* $z(\lambda_i) \to w_{LD}$.
2. *If* $\mu_i = \mu_0 \rho^i$ *for some parameter* $\rho < 1$, *then* $z(\lambda_i) \to w_{LD}$ *if* μ_0 *and* ρ *are sufficiently large*[8]
3. *If* $\bar{w} \geq w_{LD}$ *and* $\mu_i = \epsilon_i \frac{z(\lambda_i) - \bar{w}}{||\mathbf{d} - D\mathbf{x}(\lambda_i)||}$ *with* $\epsilon_i \in (0, 1)$, *then* $z(\lambda_i) \to \bar{w}$, *or algorithm 1 finds* λ_i *with* $\bar{w} \geq z(\lambda_i) \geq w_{LD}$ *for some finite* i.

Statements 2 and 3 of theorem 3.4 give an indication on how to speed-up a convergence.

I have the following reservations about the usefulness of the Lagrangean Duality as a method for solving IPs[9]:

- It is important to note that if

  $$\text{conv}(X) = \{A\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\},$$

  then the Lagrangean Dual does not provide any stronger of a bound than what one gets when solving

  $$\max\{\mathbf{cx} :\ A\mathbf{x} \leq \mathbf{b},\ D\mathbf{x} \leq \mathbf{d},\ \mathbf{x} \geq \mathbf{0}\}.$$

  So what is the use of the LD in such a case?

[8] What does it mean for ρ to be "sufficiently large"? Good question. Your guess is almost as good as mine, hence my reservations about the LD method.

[9] I have the same reservations about *all* non-exact methods for handling IPs, and in section 3.3 you discover why.

- If
$$\text{conv}(X) \neq \{A\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\},$$
then the value of solving LD is obvious, but then there is another complication: solving of Lagrangean relaxation $\Pi(\boldsymbol{\lambda})$ may not be a simple task because the formulation polyhedra is not integral. In this case the natural question is: is it worth the effort?
- Solving LD does not guarantee obtaining an optimal solution to Π. Moreover, even though theoretically theorem 3.4 guarantees convergence, finding $\boldsymbol{\lambda}_i$ in practice is a different story.

However, having stated my reservations, I have encountered one real world problem where a procedure which somewhat resembles algorithm 1 performed better than CPLEX—one of the best optimization solvers on the market (see Ernst et al. (2011)).

Based on my experience and common sense, I believe that the step length μ_i should not be a scalar that is the same for each element of vector $(\mathbf{d} - D\mathbf{x}(\boldsymbol{\lambda}_i))$. Take this point into consideration while doing exercise 3.3.

The straight Lagrangean Dual method is rarely effective. Usually it is combined with a BB procedure to obtain good bounds for pruning a BB tree.

3.3 Other Decomposition Techniques

Another famous technique for solving IPs is **column generation**. This technique is sometimes applied to IPs where the number of variables is very large and the structure of a problem is decomposable, for example if constraints of an IP resemble the following:

$$\begin{array}{llllll} A_1\mathbf{x}_1 & +A_2\mathbf{x}_2 & +\ldots & +A_K\mathbf{x}_K & = & \mathbf{b} \\ D_1\mathbf{x}_1 & & & & \leq & d_1 \\ & D_2\mathbf{x}_2 & & & \leq & d_2 \\ & & \cdots & & \leq & . \\ & & & D_K\mathbf{x}_K & \leq & d_K \\ \mathbf{x}_1 \in \mathbb{B}_+^{n_1}, & \mathbf{x}_2 \in \mathbb{B}_+^{n_2}, & \ldots, & \mathbf{x}_K \in \mathbb{B}_+^{n_K} & & \end{array}$$

Define
$$X_k = \{\mathbf{x}_k \in \mathbb{Z}_+^{n_k} : D_k\mathbf{x}_k \leq d_k\}. \tag{3.13}$$

Then for $k = 1, \ldots, K$, sets X_k are independent and are joined only by constraints
$$\sum_{k=1}^{K} A_k\mathbf{x}_k = \mathbf{b}. \tag{3.14}$$

Let the objective function be
$$\max \sum_{k=1}^{K} \mathbf{c}_k\mathbf{x}_k. \tag{3.15}$$

Hence the problem is formulated as

$$z = \max\{\sum_{k=1}^{K} \mathbf{c}_k\mathbf{x}_k : \sum_{k=1}^{K} A_k\mathbf{x}_k = \mathbf{b},\ \mathbf{x}_k \in X_k \text{ for } k = 1,\ldots,K\}, \tag{3.16}$$

where X_k is as defined in (3.13).

Assuming that each set X_k contains a large but finite number of points $\{\mathbf{x}_{k,t}\}_{t=1}^{T_k}$, any point $\mathbf{x}$ in X_k then can be written as

$$\mathbf{x} = \sum_{t=1}^{T_k} \lambda_{k,t}\mathbf{x}_{k,t},\ \sum_{t=1}^{T_k} \lambda_{k,t} = 1,\ \lambda_{k,t} \in \{0,1\} \text{ for } t = 1,\ldots,T_K. \tag{3.17}$$

Substituting for $\mathbf{x}_k$ into (3.16) leads to the following **integer Master Problem**:

IMP:

$$z^{IMP} = \max \quad \sum_{k=1}^{K}\sum_{t=1}^{T_k} (\mathbf{c}_k\mathbf{x}_{k,t})\lambda_{k,t} \tag{3.18a}$$

$$\text{s.t.} \quad \sum_{k=1}^{K}\sum_{t=1}^{T_k} (A^k\mathbf{x}_{k,t})\lambda_{k,t} = \mathbf{b} \tag{3.18b}$$

$$\sum_{t=1}^{T_k} \lambda_{k,t} = 1 \text{ for } k = 1,\ldots,K \tag{3.18c}$$

$$\lambda_{k,t} \in \{0,1\} \text{ for } t = 1,\ldots,T_K, \text{ and for } k = 1,\ldots,K. \tag{3.18d}$$

However, in practice, one does not solve the IMP as it is a huge IP, but the following relaxed version of it:

RMP:

$$z^{RMP} = \max \quad \sum_{k=1}^{K}\sum_{t=1}^{T_k} (\mathbf{c}_k\mathbf{x}_{k,t})\lambda_{k,t} \tag{3.19a}$$

$$\text{s.t.} \quad \sum_{k=1}^{K}\sum_{t=1}^{T_k} (A^k\mathbf{x}_{k,t})\lambda_{k,t} = \mathbf{b} \tag{3.19b}$$

$$\sum_{t=1}^{T_k} \lambda_{k,t} = 1 \text{ for } k = 1,\ldots,K \tag{3.19c}$$

$$\lambda_{k,t} \geq 0 \text{ for } t = 1,\ldots,T_K, \text{ and for } k = 1,\ldots,K \tag{3.19d}$$

where there is column $\begin{pmatrix} \mathbf{c}_k\mathbf{x} \\ A^k\mathbf{x} \\ \mathbf{e}_k \end{pmatrix}$ for each $\mathbf{x}$ in X_k, where $\mathbf{e}_k$ is a unit vector with 1 in the k^{th} position and zeros everywhere else. In essence, the column generation procedure *generates* one $\mathbf{x}$ in each set X_k in such a way so that constraints

(3.19b) are satisfied. (Note that choosing one $\mathbf{x}$ in each set X_k is equivalent to introducing another column $\begin{pmatrix} \mathbf{c}_k\mathbf{x} \\ A^k\mathbf{x} \\ \mathbf{e}_k \end{pmatrix}$, hence the name of the algorithm.) Which column to generate depends on the price of the corresponding variable $\mathbf{x}$. Due to the large number of variables (i.e., columns) eligible for generation, optimization is needed even to pick the best ones. Hence, optimization is needed within this optimization.

Obviously, the size of an RMP increases with each added column. Hence, at the end of the column generation procedure a huge LP must be solved. And at the very end of the procedure one must solve *exactly* an IP with the generated columns.

To keep with the spirit of these Lecture Notes, I do not cover the details of the column generation procedure. However, it is important to consider the strength of this procedure, i.e., what is the bound one can obtain from solving RMP? Proposition 3.5 answers this question.

Proposition 3.5 (Strength of RMP)

$$z^{RMP} = \max\{\sum_{k=1}^{K} \mathbf{c}_k\mathbf{x}_k : \sum_{k=1}^{K} A^k\mathbf{x}_k = \mathbf{b},\ \mathbf{x}_k \in \text{conv}(X_k)\ \textit{for}\ k = 1,\dots,K\}.$$

The following theorem then needs no proof.

Theorem 3.6 (Column generation vs. Lagrangean Dual)

$$z^{RMP} = w_{LD}.$$

In fact, column generation can be viewed as an algorithm for solving the Lagrangean Dual.[10]

Formulation IMP is also known as the **Dantzig-Wolfe** reformulation. Therefore, the column generation procedure is associated with *Dantzig-Wolfe decomposition* or *Benders' decomposition*.[11]

It comes as no surprise, thanks to duality theory, that if there is a column generation procedure, there must be a constraint generation one too. (See exercise 3.7.)

[10] This fact becomes clear when you plow through the proof of theorem 3.3. But because herein I omit most proof details, do not be surprised if this fact is not *clear* to you. Moreover, in mathematics the word "clear" in the case of proofs has a different meaning: you think about and stare at it for a very-very long time, and all of a sudden it just hits you.

[11] I introduce the names of the two decompositions for general information only (these names get thrown around frequently among researchers in OR, so you might want to note them). However, the two decompositions become important when proving theorems 3.3 and 3.6.

3.4 Heuristics

So far I have reported on some exact and not-so-exact techniques for solving IPs. As the name of the section suggests, I now discuss heuristical methods for "solving" IPs.

A heuristic (or *heuristic algorithm*) is an algorithm designed to find feasible solutions in a relatively short amount of time (a.k.a. *runtime*).

The goodness of a solution obtained by a heuristic is determined by how close it is to an optimal solution. Usually, the goodness of a solution found by a heuristic cannot be determine a priori.

Consider a maximization problem consisting of instances

$$z(I) = \max \{\mathbf{c}_I(\mathbf{x}) : \mathbf{x} \in S_I \neq \emptyset \text{ for } I \in \mathcal{I}\}, \tag{3.20}$$

where S_I is a set of feasible solutions for instance I, and $\mathcal{I}$ is the set of all instances. (For example, $\mathcal{I}$ is the set of all instances of a KNAPSACK problem, then I is one specific instance of a KNAPSACK, that is, $\mathbf{c}$, $\mathbf{a}$, and b are given.)

Let $z_H(I)$ be the value of a solution obtained by a heuristic for instance I of a problem. Can the (maximum) deviation between $z(I)$ and $z_H(I)$ be determined prior to executing a heuristic?

W.l.o.g., assume[12] that $\mathbf{c}_I(\mathbf{x}) \geq 0$ for all $\mathbf{x} \in S_I$ and all $I \in \mathcal{I}$.

Definition 3.7 (Worst-case performance of a heuristic) *A heuristic* H *has a* ***worst-case relative error performance*** *of* ϵ_H *if*

$$\epsilon_H = \inf \{\epsilon : \epsilon \geq \frac{|z(I) - z_H(I)|}{z(I)} \ \forall I \in \mathcal{I}\}. \tag{3.21}$$

Clearly, $0 \leq \epsilon_H \leq 1$ and H guarantees to find a solution with value $z_H(I) \geq (1 - \epsilon_H) \times 100\% \times z(I)$ for each instance I.

Now arises a question: Given an optimization problem of class NP-hard and $\epsilon \in (0, 1)$, does there exist a *polynomial*-time heuristic H that gives relative error $\epsilon_H \leq \epsilon$?

The answer to this question depends on the optimization problem under consideration. That is, the answer is "yes" for some NP-hard problems (e.g., KNAPSACK), "no" for many NP-hard problems, and "depende" to yet another sub-class of problems in the NP-hard category. There are such difficult optimization problems that heuristics guaranteeing $\epsilon_H \leq \epsilon$ are themselves NP-hard.[13] In such difficult cases, one just resigns to finding heuristics which perform well based on probabilistic or statistical analysis.[14]

Next, I briefly introduce some of the most popular heuristics types.

[12] We do not have to make that assumption, it's just convenient do so for a concise definition of what follows.

[13] This should not be surprising, especially if you have done exercise 2.11 and understood its consequence.

[14] Because these Lecture Notes are written for a statistically-inclined audience, I do not say any more about these types of analysis.

3.4.1 Greedy Heuristics

One of natural ways to solve a problem is to build a solution one variable at a time. This building of a solution can take on many variants. For example, a variable that gives the most improvement to the objective becomes a part of a solution—the **greedy** approach.

Example 3.8 (A greedy heuristic for KNAPSACK)
Consider KNAPSACK in example 2.1. Assume that $a_j, c_j, b \in \mathbb{Z}_+$ for $j = 1, \ldots, n$. A greedy heuristic for KNAPSACK could be formulated as in algorithm 2.

Algorithm 2 Greedy Heuristic for KNAPSACK

GREEDY HEURISTIC FOR KNAPSACK
Input: Instance I of a KNAPSACK, and starting solution $\mathbf{x}^{GH} = \mathbf{0}$.

1 define $\tilde{J} = \{j : x_j^{GH} = 0\}$
2 **for** all $j \in \tilde{J}$ **do**
3 set $v_j = 0$
4 **if** $j \in \tilde{J}$ and $\sum_{i \notin \tilde{J}} a_i x_i + a_j \leq b$ **then**
5 re-set $v_j = \frac{(\sum_{i \notin \tilde{J}} c_i x_i + c_j) - \sum_{i \notin \tilde{J}} c_i x_i}{(\sum_{i \notin \tilde{J}} a_i x_i + a_j) - \sum_{i \notin \tilde{J}} a_i x_i}$
6 **end if**
7 **end for**
8 let $\tilde{j} = \arg\max\{v_j\}$
9 **if** $v_{\tilde{j}} > 0$ **then**
10 set $x_{\tilde{j}}^{GH} = 1$
11 go to step 1
12 **else**
13 stop
14 **end if**
Output: solution $\mathbf{x}^{GH}$ to KNAPSACK

△

3.4.2 Local Search Heuristics

A greedy heuristic is usually a quick method for finding a feasible solution. But more often than not solutions delivered with greedy heuristics are far from optimal. One of the approaches to improve a solution obtained with a greedy (or any other) heuristic is a **local search**.

The idea of a local search heuristic is as follows: given a feasible solution, search a neighborhood of the solution to find a better one.

Obviously, a local search heuristic requires a definition of a *neighborhood* which can vary based on a problem structure. In term of the KNAPSACK, an example of an neighborhood is that in which one element (item) is removed or added to the "knapsack."

Example 3.9 (A local search heuristic for KNAPSACK)
Consider KNAPSACK in example 2.1. Assume that $\mathbf{x}_H$ is the solution obtained with the greedy heuristic 2. A possible local search heuristic is a procedure described in algorithm 3. △

Algorithm 3 Local Search Heuristic for KNAPSACK

LOCAL SEARCH HEURISTIC FOR KNAPSACK

Input: Instance I of a KNAPSACK, and starting solution $\mathbf{x}^H$.

1 **for** all $i = 1, \ldots, n$ such that $x_i^H = 1$ **do**
2 fix $x_i = 1 - x_i^H$, and for $j \neq i$ let $x_j = x_j^H$
3 define neighborhood of $N(i)$ as $\{j : j \neq i \text{ and } x_j^H = 0\}$
4 **for** all $j \in N(i)$ **do**
5 set $v_j = 0$
6 if $\mathbf{ax} \leq b$ re-set $v_j = \mathbf{cx}^H - \mathbf{cx}$
7 **end for**
8 let $\tilde{j} = \arg\max\{v_j\}$
9 **if** $v_{\tilde{j}} > 0$ **then**
10 set $x_i^H = 0$ and $x_{\tilde{j}}^H = 1$
11 **end if**
12 **end for**

Output: solution $\mathbf{x}_{LSH}$ to KNAPSACK

Note that steps 4–11 of heuristic algorithm 3 could be done differently. I use a greedy approach to determine an element from the neighborhood that is being added to the "knapsack." Other approaches can be used. (Which ones?) Also, the way I defined a neighborhood in step 3 is arbitrary. Hence, with heuristics one has plenty of degrees of freedom, and that is why there can be such an abundance of heuristics even for one class of problems.

Another point to note in algorithm 3 is the decision step 9. I allow for variable j to be added only if swapping between j and i increases the OFV. But that is not always the best strategy. Very often heuristics get trapped in a **local optimal** solution. But in order to get to a better solution, one must allow the OFV to worsen before it improves again, hence leading to another type of a heuristic.

3.4.3 Other Types of Heuristics

The idea that one should sometimes accept a worse solution in order to be able to get to a better one later in a solution process is natural to a **simulated annealing** heuristic: a solution with a worse OFV is accepted with some probability in the interval $(0, 1)$. How to determine this probability usually depends on the OFV.

A **tabu search** heuristic is designed to prevent cycling between the same solutions by keeping a *tabu* list of forbidden solutions, i.e., solutions that have already been found and evaluated in previous iterations.

Then there are **genetic algorithms** which suit binary variables very well. For example, $\tilde{\mathbf{x}}$ and $\hat{\mathbf{x}}$ can be two good solutions one has found for an instance of KNAPSACK. A natural assumption that a combination—a *crossover*—of these two parent solutions should, according to the evolution theory, produce a good offspring, i.e., another good solution. The problem with a genetic heuristic is ensuring that an offspring of two good feasible solutions is, too, a feasible solution.

3.5 Summary

I already stated some of my reservations about different methods through out this chapter. In table 3.1 I summarize disadvantages and advantages of different methods.

Method	Advantages	Disadvantages
BB	obtains an optimal solution	memory and CPU time consuming
BC	obtains an optimal solution	needs theoretical work to determine good cuts; can still be memory and CPU time consuming
LD	can provide good bounds	not an exact method; delivered solutions might be far from optimal
CG	can obtain a near-optimal solution; can be used to handle nonlinear constraints	can be memory and CPU time consuming
Heuristics	usually fast	delivered solutions can be of arbitrarily poor quality

Table 3.1: Advantages and disadvantages of various solution methods for IPs.

As you see, there is no silver bullet. For each class of problems there is usually a combination of methods that give good results. At the same time, because there are so many possible combinations of methods (e.g., even a definition of a neighborhood can make a difference and, hence, result in a different local search heuristic), the number of things one can try is almost limitless.[15]

[15]Good news for academics who are always in need of publishing material. Also good news for OR practitioners like me, who will always have a job.

3.6 Exercises

Exercise 3.1 (Simple logical inequalities from Wolsey (1998)) *Consider the following set of constraints:*

$$\begin{aligned} 7x_1 + 3x_2 - 4x_3 - 2x_4 &\leq 1 && \text{(3.22a)}\\ -2x_1 + 7x_2 - 3x_3 - x_4 &\leq 6 && \text{(3.22b)}\\ -2x_2 - 3x_3 - 6x_4 &\leq -5 && \text{(3.22c)}\\ 3x_1 - 2x_3 &\geq -1 && \text{(3.22d)}\\ \mathbf{x} &\in \mathbb{B}^4 \end{aligned}$$

1. *Examine constraint* (3.22a). *Note that if* $x_1 = 1$, *then necessarily* $x_3 = 1$. *How can this relationship be expressed with an additional constraint? (Remark: the additional constraint strengthens the formulation.)*
2. *Other similar observations?*

Exercise 3.2 (BB tree pruning from Wolsey (1998)) *Consider the enumeration tree (minimization problem) in figure 3.2. As discussed in the lecture, the number under a node is the optimal OFV of a relaxation problem solved at that node. The number above a node (if it exists) is the OFV of a feasible solution found at that node (often due to a heuristic applied at that node).*

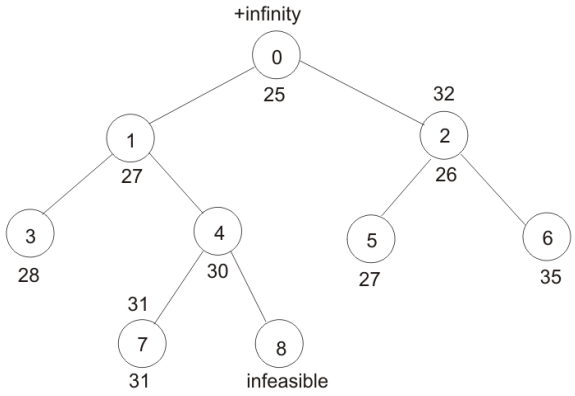


Figure 3.5: BB tree for exercise 3.2.

a) *Give the tightest possible lower and upper bounds on the optimal OFV.*
b) *Which nodes can be pruned (why?) and which must be explored further?*

Exercise 3.3 *Consider GAP from example 2.30.*

- *Formulate a Lagrangean relaxation (i.e., problem* $\Pi(\boldsymbol{\lambda})$*) to GAP, and show that it is a relaxation (as in proposition 3.1).*
- *Formulate the LD to GAP.*

- *What type of bound do you expect from solving the LD?*
- *Design a Lagrangean Dual algorithm for solving GAP.*

Exercise 3.4 (Lagrangean Dual for a capacitated minimum network flow problem) *Consider the constrained shortest path problem in the figure below. The cost of using arc (i,j) is given by the value c_{ij}. The travel time on arc (i,j) is given by the value t_{ij}. The objective is to find a path from node s to node f so that the total cost is minimized and the total time does not exceed 42 units.*

1. *Formulate this problem as IP. (Note, the problem is* NP*-complete.)*
2. *Enumerate all the paths from s to f and give the cost and time for each path.*
3. *Form the Lagrangian problem by dualizing the time constraint.*
4. *For each path from s to f plot the Lagrangian objective function as a function of the Lagrangian variable λ.*
5. *Using the plot in the previous part, find the value of λ that maximizes the Lagrangian dual. What is the strength of this LD?*
6. *Now write the Lagrangian relaxation where the network constraints are relaxed (dualized). What is the strength of such LD as compared to the one we considered above? (I.e. which LD could possibly give a better bound on z_{IP}?)*

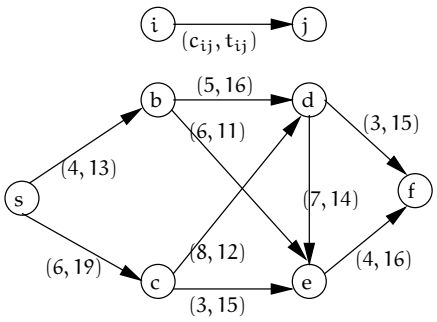


Figure 3.6: Network for exercise 3.4.

Exercise 3.5 (Greedy Heuristic for INTEGERKNAPSACK) *Consider* INTEGERKNAPSACK

$$z(b) = \max\{\mathbf{cx} : \mathbf{ax} \leq b,\ \mathbf{x} \in \mathbb{Z}^n_+\}, \tag{3.23}$$

where $\mathbf{c}, \mathbf{a} \in \mathbb{Z}^n_+$, $b \in \mathbb{Z}_+$, and $a_i \leq b\ \forall i = 1, \ldots, n$. W.l.o.g., assume that $\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \ldots \geq \frac{c_n}{a_n}$. Develop a heuristic for INTEGERKNAPSACK *with the worst-case performance guarantee $\epsilon_H \leq 0.5$.*

Exercise 3.6 (A heuristic for a simplified GAP) *Given n jobs, m workers and cost c_{wj} of assigning a job j to a worker w, find the cheapest assignment of jobs to workers. A worker can be assigned an arbitrary number of jobs. The problem formulation is as follows:*

$$x_{wj} = \begin{cases} 1 & \textit{if worker } w \textit{ is assigned to job } j \\ 0 & \textit{else.} \end{cases}$$

$$\min \quad \sum_{j=1}^{n} \sum_{w=1}^{m} c_{wj} x_{wj} \tag{3.24a}$$

$$s.t. \sum_{w=1}^{m} x_{wj} = 1 \qquad \forall j = 1, \ldots, n \tag{3.24b}$$

$$x_{wj} \in \mathbb{B} \qquad \forall w = 1, \ldots, m, \ \forall j = 1, \ldots, n \tag{3.24c}$$

Design a greedy heuristic for the problem type. What is the worst-case performance guarantee of your heuristic?

Exercise 3.7 (Constraint Generation Algorithm) *Design a constraint generation algorithm for a generic IP which delivers a feasible solution. Determine the worst-case performance of your algorithm.*

Exercise 3.8 (Greedy Heuristic for GAP)

1. *Formulate and program a greedy heuristic for GAP (see example 2.30). How good are the solutions obtained by your heuristic?*
2. *Enhance your greedy heuristic from part 1 to include a local search component.*

Exercise 3.9 (Greedy Heuristic for AP) *Design a greedy heuristic for an AP like in example 2.28 What is the worst-case performance guarantee of your heuristic?*

CHAPTER 4

Special Topics

In this section I highlight some of the most well-known optimization topics.

4.1 Network Optimization

The main references for section 4.1 are Hamacher and Klamroth (2000) and Bazaraa et al. (1990). So far I have touched on network optimization with the MINCOSTFLOW in section 2.2. There are a few other types of network flow problems, e.g., the shortest path problem[1], and the maximum flow problem.

Throughout this section we assume that graphs do not contain negative cost cycles.

The **shortest path problem** is concerned with finding a shortest path P_{ab} between two nodes a and b in V. Usually, a graph $G = (V, E)$ with costs on the arcs is given. There are no supply and demand nodes. There are no capacities on the arcs.

The shortest path problem easily converts into MINCOSTFLOW by creating a supply and a demand of 1 at the start- and end-node of the path, respectively. Even if some arc costs are negative (as long as no negative cycles exist), there is a polynomial-time algorithm for finding a shortest path between any two nodes. The runtime of the algorithm is $\mathcal{O}(|V| \cdot |E|)$. Moreover, for any two nodes i and j on the path P_{ab}, the shortest path between them is a sub-path of P_{ab}. Clearly, the problem of finding shortest paths between all pairs of vertices $i, j \in V$ of $G = (V, E)$ can also be solved in polynomial time.

Because of the structure of the node-arc adjacent matrices in network flow problems, there is an interesting occurrence which is often very handy: at optimality $c_{ij} - v_i + v_j = 0$ for all arcs $(i, j) \in E$ which have non-zero flow in the

[1] If you have done exercise 3.4, you are already familiar with this problem.

optimal solution, with v_i and v_j being duals to the corresponding constraints (a.k.a. node potentials).

The **maximum flow problem** (MAXFLOW) poses the question of how many units of flow can be transported between two nodes in a network. For such a problem graph $G = (V, E)$ usually has no cost but has capacities on the amount of flow that can be pushed through each arc and/or node. W.l.o.g. assume that the flow from node 1 to n (where $n = |V|$) is to be maximized. Let f represent the amount of flow. Then the MAXFLOW problem can be formulated as follows:

$$\max \quad f \tag{4.1a}$$

$$\text{s.t.} \sum_{(i,j)\in E} x_{ij} - \sum_{(j,i)\in E} x_{ji} = \begin{cases} f & \text{if } i = 1 \\ 0 & \text{if } i \neq 1 \text{ or } |V| \\ -f & \text{if } i = |V| \end{cases} \qquad \forall i \in V \tag{4.1b}$$

$$x_{ij} \leq u_{ij} \qquad \forall i, j \in V \tag{4.1c}$$

$$x_{ij} \geq 0 \qquad \forall i, j \in V \tag{4.1d}$$

The above can be re-written as

$$\max \quad f \tag{4.2a}$$

$$\text{s.t.} \quad (\mathbf{e}_n - \mathbf{e}_1)f + A\mathbf{x} = \mathbf{0} \tag{4.2b}$$

$$\mathbf{x} \leq \mathbf{u} \tag{4.2c}$$

$$\mathbf{x} \geq \mathbf{0} \tag{4.2d}$$

where the left-hand side of constraints (4.2b) is a TU matrix. (Note, $\mathbf{e}_i$ is a unit vector with 1 in the i^{th} position and zeroes everywhere else.) Hence MAXFLOW problem can be easily solved even if an integer solution is required.

Another well-known problem from network optimization is the **minimal spanning tree** (MINSPANTREE). Given a graph $G = (V, E)$ with costs/weights c_{ij} $\forall (i, j) \in E$, find a spanning tree of minimal cost/weight. An optimal solution to MINSPANTREE can be found with a greedy heuristic. (What would it look like?)

With the enormous growth of broadband, telecommunications needs optimization: How should one route different traffic demands in a telecommunication network? Moreover, telecommunication networks are complicated because packets of data originate in different nodes and must be routed to their specific destinations. Because packets of data are not interchangeable, they are looked at as different commodities which must be sent through a network to satisfy respective demands with minimal cost. This is an example of a **multicommodity minimum cost flow** problem. Arcs have commodity-dependent costs $c_{k,(ij)}$ and capacities $u_{k,(ij)}$ for commodity $k \in K$. Moreover, there are also compound arc capacities u_{ij} $\forall (i, j) \in E$. The node-arc incidence matrix is the same for all commodities; however, vector $\mathbf{b}_k$ of supply-demand varies per commodity.

Components of the decision variable vector $\mathbf{x}_k$ give the amount of flow of commodity $k \in K$ on each arc $x_{k,(ij)}$ $\forall (i, j) \in E$. The formulation of the multicom-

modity minimum cost flow problem is as follows:

$$\begin{aligned}
\min \quad & \sum_{k \in K} \mathbf{c}_k \mathbf{x}_k && && (4.3a)\\
\text{s.t.} \quad & A\mathbf{x}_k = \mathbf{b}_k && \forall\, k \in K && (4.3b)\\
& \mathbf{x}_k \leq \mathbf{u}_k && \forall\, k \in K && (4.3c)\\
& \sum_{k \in K} x_{k,(ij)} \leq u_{ij} && \forall\, (i,j) \in E && (4.3d)\\
& \mathbf{x}_k \geq \mathbf{0} && \forall\, k \in K. && (4.3e)
\end{aligned}$$

If flows are restricted to be integers, i.e., $\mathbf{x}_k \in \mathbb{Z}_+^{|E|}\ \forall\, k \in K$, then the multicommodity minimum cost flow problem is NP-hard even for two commodities.

Clearly, the multicommodity minimum cost flow problem can be applied to traffic routing. Then with additional tweaks, the formulation above can be used to minimize traffic congestion.

Applications for network flows are ubiquitous: finding the shortest route to get from point A to point B, finding the best schedule, producing the best evacuation plan, designing an infrastructure for a new region/city, etc. And due to their structure (the possession of a node-arc incidence matrix), network flow problems are very interesting[2] and provide plenty of research material.

4.2 Location Theory

The main reference for this section is Drezner and Hamacher (2004). One of the oldest problems in location theory is known as the **Weber problem**.

The Weber problem is defined as follows: Given a set of points $\{p_1, \ldots, p_n\}$ with coordinates $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ in $\mathbb{R}^2$, determine a location (x^*, y^*) of point p^* so that the sum of the weighted Euclidean distances from p^* to all p_i is minimized, i.e.,

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n} w_i \operatorname{dist}(p_i, p^*) && (4.4a)\\
\text{s.t.} \quad & (x^*, y^*) \in \mathbb{R}^2, && (4.4b)
\end{aligned}$$

where $\operatorname{dist}(p_i, p^*) = \sqrt{(x_i - x^*)^2 + (y_i - y^*)^2}$.

A location problem arises whenever a question is posed: where to place a point? In the real world such a decision problem is regularly encountered when strategic business decisions are made, e.g., where to locate a new distribution center (or any type of facility)? The question of where-to-locate is usually accompanied by the following two questions:

1. What are the location options?

[2] I find them cute, but again, it's up to your definition of "cute".

2. How should the options be compared? (I.e., how to evaluate the goodness of an option?)

A location problem can either be continuous or discrete. In the case of continuous location problems, the location point is described by continuous variables, i.e., coordinates in a continuous n-dimensional space. A discrete location problem presents a finite set of possibilities (usually a finite set of coordinates), and a binary variable can be used to describe a decision of whether to chose a particular location. Hence, a discrete location problem turns into a usual IP. Most real world location problems are discrete.

The question about how to compare the options involves some sort of cost which is usually a function of distance. The notion of distance is generic as well. There are different types of distances that can be used, e.g., Manhattan and Euclidean, that is why a generic notation $\text{dist}(p_i, p_j)$ is used to denote the distance between points p_i and p_j. Whatever distance is used, it is reasonable to assume that it satisfies the following three properties:

symmetry : $\text{dist}(p_i, p_j) = \text{dist}(p_j, p_i)$

zero-distance : $\text{dist}(p_i, p_i) = 0$

triangle inequality : $\text{dist}(p_i, p_j) + \text{dist}(p_j, p_k) \geq \text{dist}(p_i, p_k)$

Consider two classic discrete location problems: k-median and k-center. Suppose we have a set of clients (stores) which must be served by k distribution centers (warehouses), where $k \geq 1$. The **k-median problem** (kMP) asks the following question: where to build distribution centers so that the total cost of serving the clients is minimized? Whereas the **k-center problem** (kCP) seeks to find locations for the distribution centers so that the maximum cost of serving any client is minimum.

Cost can be defined as a function of distance and time to get to the clients. The distance component can be precalculated to include the actual road distances. A client can be served from one or multiple warehouses (single or multiple allocation).

Let $\mathcal{I}$ be the set of possible locations of the distribution centers. The locations of clients are included into $\mathcal{I}$ as well. (Why?) The decision variable is

$$x_{ij} = \begin{cases} 1 & \text{if client } i \text{ is allocated to distribution center } j \\ 0 & \text{else.} \end{cases}$$

Distance between sites $i, j \in \mathcal{I}$ is denoted by $\text{dist}(i, j)$. As a result the discrete

kMP is formulated as follows:

$$\begin{aligned}
\min \quad & \sum_{i,j\in\mathcal{I}} \mathrm{dist}(i,j)\cdot x_{ij} && && (4.5a)\\
\text{s.t.} \quad & \sum_{j\in\mathcal{I}} x_{ij} \geq 1 && \forall\, i \in \mathcal{I} && (4.5b)\\
& x_{ij} \leq x_{jj} && \forall\, i,\, j \in \mathcal{I} && (4.5c)\\
& \sum_{j\in\mathcal{I}} x_{jj} \leq k && && (4.5d)\\
& x_{ij} \in \mathbb{B} && \forall\, i,\, j \in \mathcal{I} &&
\end{aligned}$$

The kCP is formulated as follows:

$$\begin{aligned}
\min \quad & \omega && && (4.6a)\\
\text{s.t.} \quad & \omega \geq \mathrm{dist}(i,j)\cdot x_{ij} && \forall\, i,\, j \,\mathrm{in}\mathcal{I} && (4.6b)\\
& \sum_{j\in\mathcal{I}} x_{ij} \geq 1 && \forall\, i \in \mathcal{I} && (4.6c)\\
& x_{ij} \leq x_{jj} && \forall\, i,\, j \in \mathcal{I} && (4.6d)\\
& \sum_{j\in\mathcal{I}} x_{jj} \leq k && && (4.6e)\\
& x_{ij} \in \mathbb{B} && \forall\, i,\, j \in \mathcal{I} &&
\end{aligned}$$

One very interesting problem with application in telecommunications and all types of logistics and transportation is the hub location problem.

4.2.1 Hub Location Problems

In real-world transportation and telecommunication situations there are often origin-destination (o-d) pairs which must exchange information, passengers, goods, parcels, etc. The most desirable scenario might be to allow direct exchanges between any two points, but the cost of establishing such a complete network is unreasonably high. As a result, *hub location models* have been developed to provide an alternative to a complete network structure for such transportation situations. Given a set of nodes which must exchange commodities, choose a set of completely interconnected hubs (from the set of the nodes), allocate the non-hub nodes (so-called *spokes*) to one (single allocation) or more (multiple allocation) hubs so that transportation cost is minimized. The resulting graph is called a *hub-and-spoke* network (see figure 4.1). The choice of hub locations and the allocation of spokes to hubs constitue the optimization problem.

No exchange of a commodity in such a hub network uses more than three edges due to the complete graph structure among the hub nodes. The exchange cost is modeled as $\mathrm{dist}(i,l) + \alpha\,\mathrm{dist}(l,m) + \mathrm{dist}(m,j)$, where l, m are hubs and $\alpha \in [0,1]$ is a discount rate on the interhub arcs representing economies

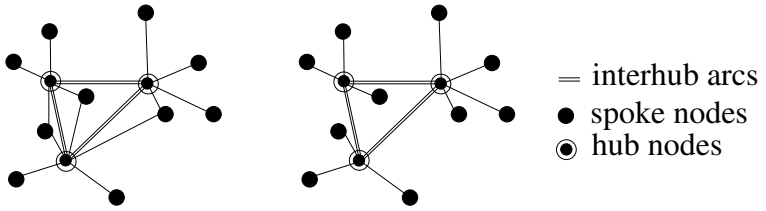


Figure 4.1: Multiple and single allocation hub-and-spoke networks

of scale. There are different types of hub location models classified according to the objective function, for example, uncapacitated single allocation hub median problem (USAHMP), uncapacitated multiple allocation hub median problem (UMAHMP), uncapacitated single allocation k-hub center problem (USAkHCP), uncapacitated multiple allocation k-hub center problem (UMAkHCP), uncapacitated single allocation k-hub median problem (USAkHMP), and uncapacitated multiple allocation k-hub median problem (UMAkHMP).

Let r_j denote the *radius* of hub j, i.e., the length of the longest spoke-edge connected to this hub. There is a decision variable y_{ij} for each node which takes on value one if and only if node i is allocated to hub at node j. Let V be the index set of the nodes.

USAkHCP: $\min\ \omega$

$$\text{s.t.} \sum_{j \in V} y_{ij} = 1 \qquad \forall i \in V \tag{4.7a}$$

$$y_{ij} \leq y_{jj} \qquad \forall i, j \in V \tag{4.7b}$$

$$\sum_{i \in V} y_{ii} = k \tag{4.7c}$$

$$r_j \geq \text{dist}(i,j) y_{ij} \qquad \forall i, j \in V \tag{4.7d}$$

$$\omega \geq r_j + r_i + \alpha\, \text{dist}(i,j) \qquad \forall i, j \in V \tag{4.7e}$$

$$y_{ij} \in \{0,1\} \qquad \forall i, j \in V. \tag{4.7f}$$

Constraints (4.7a) and (4.7f) assure single allocation of nodes to hubs. Constraints (4.7b) allow allocation to a node if and only if it is a hub. Constraint (4.7c) is the condition of having exactly k hubs. Constraints (4.7d) establish the radius of each hub. Finally, constraints (4.7e) together with the objective function minimize the maximum distance between two nodes forming an o-d pair. USAkHCP is NP-hard.

If a set of hubs is fixed, with H being its index set and $H \subset V$, finding an optimal single allocation of spokes to hubs is NP-hard.

4.3 Multicriteria Optimization

The material of this section is built on a few setions from Ehrgott (2005). Multicriteria optimization surrounds us. In fact, in life we constantly make decisions based on multiple criteria. Consider example 4.1.

Example 4.1
You are considering buying a car. You have done your research and put together the list of options: Toyota Corolla, Hyundai i30, Audi A3, and BMW 316. The following criteria are important in your decision-making process: price (Toyota and Hyundai win according to this criterion), service availability and cost of upkeep (again Toyota and Hyundai win), durability/quality (Audi and BMW are in the lead on this criterion), power (Audi and BMW give you a nice rush when you accelerate), and value depreciation (Audi and BMW are in the lead on this criterion as well). Which car should you settle for? How do you decide on an optimal choice?

Suppose that the quality measure you are using is from *Top Gear* magazine, and it is a score from 1 to 100 (100 being best). Figure 4.2 shows the position of each choice in the space of quality and price. △

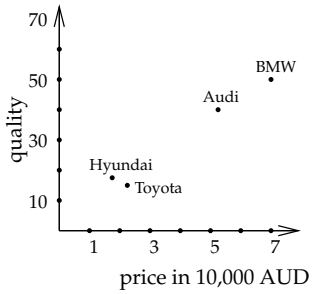


Figure 4.2: Price vs. quality for the four automobile models.

Note if price and quality were your only criteria then the choice of Toyota Corolla should be disregarded as it is worse in quality and more expensive than Hyundai. In such a case it is said that Hyundai **dominates** Toyota. However, no such comparison is possible between Hyundai, Audi and BMW because as the price increases so does the quality measure. The three alternatives are called **efficient** solutions (with respect to quality and price criteria). In other words, efficient solutions are *equally optimal*.

There are notions of **decision** (a.k.a. variable) and **criterion** (a.k.a. objective) space. The criterion space is the decision space after it has been mapped through objective functions.

Consider a multicriteria optimization problem (**MOP**):

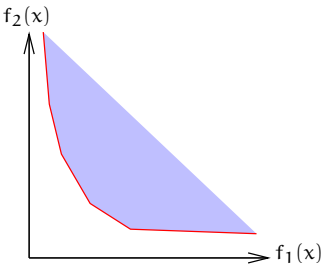
$$\min_{\mathbf{x}\in\mathcal{X}} \quad (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_K(\mathbf{x})) \tag{4.8}$$

For brevity, vector $(f_1(x), f_2(x), \ldots, f_K(x))$ is denoted by $\mathbf{f}(\mathbf{x})$.

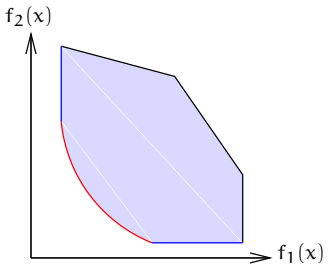
Definition 4.2 (Efficiency) *A solution $\widetilde{\mathbf{x}} \in \mathcal{X}$ is **efficient** if there is **no** $\mathbf{x} \in \mathcal{X}$ (with $\mathbf{x} \neq \widetilde{\mathbf{x}}$) such that $f_k(\mathbf{x}) \leq f_k(\widetilde{\mathbf{x}})\ \forall k = 1, \ldots, K$ with strict inequality holding in at least one criteria $f_k(\mathbf{x})$. If $\widetilde{\mathbf{x}}$ is efficient, then $\mathbf{f}(\widetilde{\mathbf{x}})$ is called a **nondominated** point.*

The set of all efficient solutions is also called the **Pareto set** or **Pareto front**. Moreover, there are notions of *weakly*, *strictly* and *properly* efficient solutions.

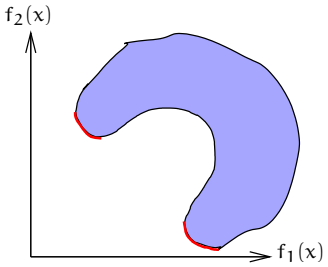
In order to cover these concepts, much more notation must be introduced. Ehrgott (2005) contains a very thorough explanation and treatment of all these ideas.



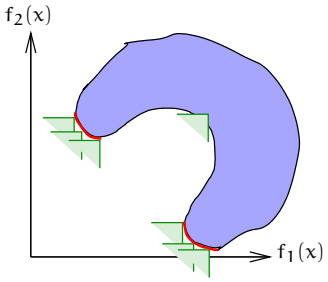
(a) Red lines represent nondominated points.



(b) Red line denotes all nondominated points. Blue lines are the set of weakly nondominated points.



(c) Objective set is nonconvex. Red curves are the nondominated points.



(d) For minimization objectives the set of nondominated points can be determined in the following way: attached a green triangle to any point in the objective space, if the intersection of the triangle and the objective space is only one point, that point is nondominated.

Figure 4.3: Objective space for MOP with objective min $(f_1(\mathbf{x}), f_2(\mathbf{x}))$.

Example 4.3
A *continuous* but less real-wordly example is to minimize

$$f_1(x) = \sqrt{x+1} \text{ and} \tag{4.9}$$
$$f_2(x) = (x-2)^2 + 1 \tag{4.10}$$

where $x \geq 0$. The individual minimization problems are easy: $f_1(x)$ is minimal at $x = 0$ and $f_2(x)$ is minimal at $x = 2$. Note that $f_1(0) = 1$, $f_1(2) = \sqrt{3}$, $f_2(0) = 5$, and $f_2(2) = 1$; thus points $(f_1(0), f_2(0))$ and $(f_1(2), f_2(2))$ are equally good and so are solutions $x = 0$ and $x = 2$. In fact, all $x \in [0, 2]$ are efficient solutions for the given set of objective functions.

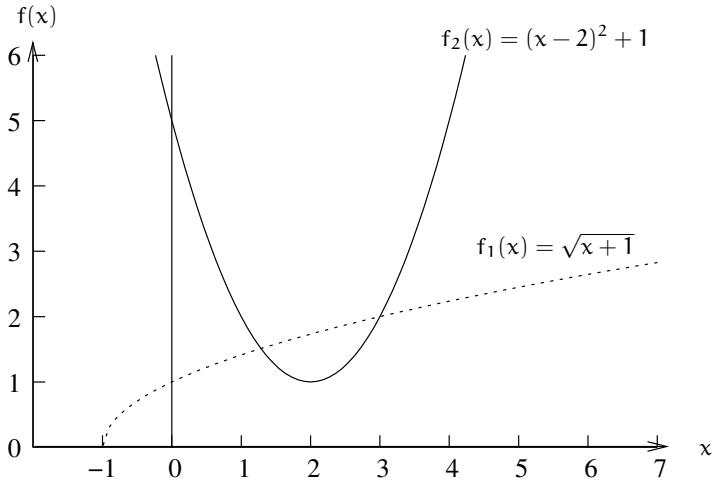


Figure 4.4: Feasible region and the two objective functions for example 4.3.

△

Solution Methods for MOP

Consider MOP:

$$\min_{x \in \mathcal{X}} \quad (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_K(\mathbf{x})). \tag{4.11}$$

Choose weight λ_k for k in K and solve the *weighted sum problem*

$$\min_{x \in \mathcal{X}} \quad \sum_{k=1}^{K} \lambda_k f_k(\mathbf{x}). \tag{4.12}$$

Optimization problem 4.12 is called a **weighted sum scalarization** of MOP 4.11.

Theorem 4.4 *Let $\widetilde{\mathbf{x}}$ be an optimal solution of the weighted sum optimization problem given in* (4.12)*.*

- *If $\boldsymbol{\lambda} > \mathbf{0}$, then $\widetilde{\mathbf{x}}$ is an efficient solution of* (4.11)*.*
- *If $\boldsymbol{\lambda} \geq \mathbf{0}$ and $\widetilde{\mathbf{x}}$ is the unique solution to* (4.12)*, then $\widetilde{\mathbf{x}}$ is an efficient solution of* (4.11)*.*

If more than one efficient solution is wanted, then (4.12) must be solved multiple times with varying $\boldsymbol{\lambda}$. Moreover, the following result holds:

Theorem 4.5 *Let $\mathcal{X}$ be a convex set and $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_K(\mathbf{x})$ be convex functions. Then for every efficient point $\widetilde{\mathbf{x}}$ in $\mathcal{X}$ there exists $\boldsymbol{\lambda} > \mathbf{0}$ such that $\widetilde{\mathbf{x}}$ is an optimal solution of* (4.12)*.*

Another approach is to minimize only one objective at a time while restricting the other objectives, i.e.,:

$$\min_{x \in \mathcal{X}} \quad f_j(\mathbf{x}) \tag{4.13a}$$

$$\text{s.t.} \quad f_k(\mathbf{x}) \leq \epsilon_k \qquad \forall\, k \in K,\ k \neq j, \tag{4.13b}$$

where $\boldsymbol{\epsilon} \in \mathbb{R}^K$. This approach is called the **ϵ-constraint method**.

Theorem 4.6
- *Let $\widetilde{\mathbf{x}} \in \mathcal{X}$ be an optimal solution of* (4.13) *for some* j. *Then $\widetilde{\mathbf{x}}$ is an efficient solution of* (4.11).
- *The feasible solution $\widetilde{\mathbf{x}} \in \mathcal{X}$ is an efficient solution of* (4.11) *if and only if there exists $\hat{\boldsymbol{\epsilon}} \in \mathbb{R}^K$ such that $\widetilde{\mathbf{x}}$ is an optimal solution of* (4.13) *for all* $j = 1, \ldots, K$.

Hence theorem 4.6 says that with appropriate $\boldsymbol{\epsilon}$ *all* efficient solutions can be found. (Note the resemblance between theorems 4.6 and 4.5.)

Theorem 4.7 (Comparing two methods)
- *Let $\widetilde{\mathbf{x}} \in \mathcal{X}$ be an optimal solution of* (4.12). *If $\lambda_j > 0$, then there exists $\boldsymbol{\epsilon} \in \mathbb{R}^K$ such that $\widetilde{\mathbf{x}}$ is an optimal solution of* (4.13).
- *Suppose $\mathcal{X}$ is a convex set and $f_k : \mathbb{R}^n \to \mathbb{R}$ are convex functions. If $\widetilde{\mathbf{x}}$ is an optimal solution of* (4.13) *for some* j, *then there exists $\boldsymbol{\lambda} \geq \mathbf{0}$ such that $\widetilde{\mathbf{x}}$ is an optimal solution of* (4.11).

The next step is obvious: combine the two methods into a hybrid approach. Let $\widetilde{\mathbf{x}}$ be an arbitrary feasible point for an MOP (4.11). Consider the following optimization problem:

$$\min_{x \in \mathcal{X}} \quad \boldsymbol{\lambda} \mathbf{f}(\mathbf{x}) \tag{4.14a}$$

$$\text{s.t.} \quad f_k(\mathbf{x}) \leq f_k(\widetilde{\mathbf{x}}) \qquad \forall\, k \in K, \tag{4.14b}$$

where $\boldsymbol{\lambda} \geq \mathbf{0}$. A non-surprising result is as follows:

Theorem 4.8 *Let $\boldsymbol{\lambda} > \mathbf{0}$. A feasible solution $\widetilde{\mathbf{x}} \in \mathcal{X}$ is an optimal solution of* (4.14) *if and only if $\widetilde{\mathbf{x}}$ is an efficient solution of* (4.11).

The ϵ-constraint method is inconvenient for two reasons:

- If any element of $\boldsymbol{\epsilon}$ is too restrictive, problem (4.13) is infeasible.
- If there is much "slack" in one of the components of $\boldsymbol{\epsilon}$, then obtained optimal solutions might not be unique (which would not allow us to conclude efficiency of the obtained solutions).

The **elastic constraint method** was designed to address the two disadvantages, and is based on solving the following optimization problem:

$$\min_{x \in \mathcal{X}} \quad f_j(\mathbf{x}) + \sum_{\substack{\forall k \in K \\ k \neq j}} \mu_k s_k \tag{4.15a}$$

$$\text{s.t.} \quad f_k(\mathbf{x}) - s_k \leq \epsilon_k \qquad \forall\, k \in K,\ k \neq j \tag{4.15b}$$

$$s_k \geq 0 \qquad \forall\, k \in K,\ k \neq j \tag{4.15c}$$

where $\mu_k \geq 0 \; \forall k \in K$ with $k \neq j$.

Theorem 4.9
- *If $\widetilde{\mathbf{x}} \in \mathcal{X}$ is a unique optimal solution of* (4.15)*, then $\widetilde{\mathbf{x}}$ is an efficient solution of* (4.11)*.*
- *If $\widetilde{\mathbf{x}} \in \mathcal{X}$ is an efficient solution of* (4.11)*, then there exist ϵ_k, $\mu_k \geq 0$, and s_k such that $(\widetilde{\mathbf{x}}, \mathbf{s})$ is an optimal solution of* (4.15) *for all $k \in K$.*

There are other methods for solving MOPs, but I guess by now you get the idea and can build other hybrid methods from the ones I mentioned here.

Finding all efficient solutions is the most common form of multicriteria optimization. Having a reasonably large set of efficient solutions allows us to make more informed decisions.

4.4 Exercises

Exercise 4.1 (Network reliability) *Consider a digraph $G = (V, E)$. Suppose you want to transmit a message from node $s \in V$ to node t. The labels on the arcs reflect the probability that a message gets through the arc without getting lost, i.e., $r_{ij} \in [0, 1] \; \forall (i, j) \in E$. Formulate a linear integer problem that finds the most reliable path (i.e., the path that has the maximum probability that a message gets through from the origin node s to the destination node t).*

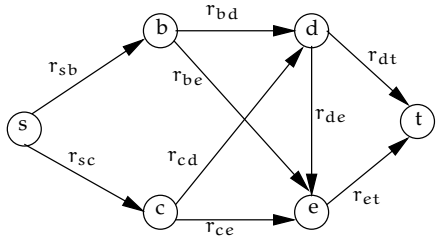


Figure 4.5: Network graph for exercise 4.1.

Exercise 4.2 (MAXFLOW with node capacities) *Consider a digraph $G = (V, E)$ with capacities $u_{ij} \; \forall (i, j) \in E$. Moreover, the nodes have capacities as well, i.e., $u_i \; \forall i \in V$. Formulate this problem for the network given in figure 4.6 as the standard* MAXFLOW *program described in section 4.1.*

Exercise 4.3 (Interviewer Location Problem) *Every five years after an Australian census, the sample for the Monthly Population Survey (MPS) is redesigned. Based on the new sample, PSO have a good idea about the areas (locations) of the new sample for the MPS. The following questions arise: How many interviewers should be hired? And where (location-wise) should they be hired? Formulate this problem as an IP. What should be your objective function(s)? What would be your solution approach?*

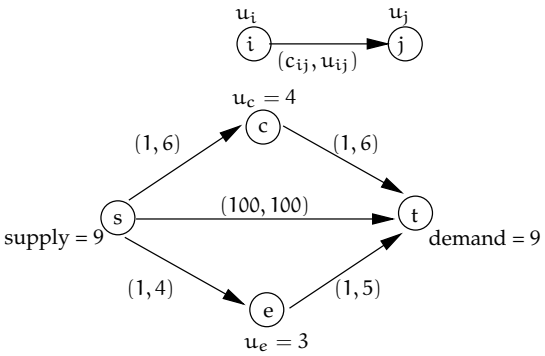


Figure 4.6: Network graph for exercise 4.2.

Exercise 4.4 (Interviewer Training Problem) *All interviewers on a panel are trained to perform MPS. However, for each special social survey (SSS) interviewers must be trained. Training can take up to 3-4 days and thus is expensive. Given the locations of interviewers, their availability, the locations of the blocks to be enumerated for a SSS, and the time frame for the enumeration, which interviewers should be trained for a given SSS? How would you go about formulating this problem as an MIP.*

Exercise 4.5 (UMAkHMP) *Based on the formulation for USAkHCP, what would be a formulation of the uncapacitated multiple allocation k-hub median problem?*

Exercise 4.6 (from Bazaraa et al. (1990)) *Exercise 12.41 p. 617 of Bazaraa et al. (1990). (copy and distribute)*

Exercise 4.7 (A transportation problem from Bazaraa et al. (1990)) *Consider a set of origins and destinations. The cost of transporting goods from origin o_i to destination d_j and the demands and supplies at the origins and destinations, respectively, are given in table 4.1*

	d_1	d_2	d_3	supply
o_1	2	1	1	3
o_2	1	4	4	5
demand	1	3	4	

Table 4.1: Transportation costs, demand, and supply for exercise 4.7.

a) *Prove that $(x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23}) = (0, 3, 0, 1, 0, 4)$ is an optimal solution.*

b) *Is there an economical interpretation of the dual variable for the solution in part (a)?*

Exercise 4.8 *Formulate a greedy algorithm for solving the* MINSPANTREE *problem on an undirected graph.*

Basic Linux Commands

To make the examples of the commands (called **utilities** in Linux) more elaborate and interesting, let us assume the following directory structure:

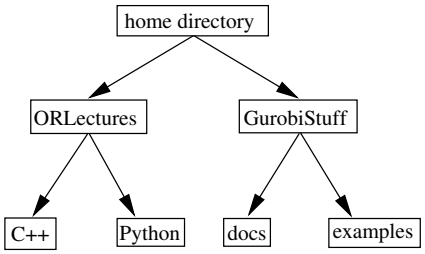


Figure 4.7: Directory structure.

The list of utilities I give here is limited. I included commands that I most often use. However, my golden rule is: Google knows everything. The following link is the one I use most often to find info about Linux utilities:

`http://www.computerhope.com/unixtop1.htm`

If you know the utility you are after but do not know how to use it, you can always look in the manual pages as follows:
elise:~> `man` [utilityname]
Even though these manual pages have exhaustive explanations of the utility you are after and its options, they do not contain examples.[3]

[3] I usually prefer to see examples of a utility's usage. Hence, I usually do not use these manual pages.

Util	Example	Explanation
mkdir	mkdir GurobiStuff	make directory called GurobiStuff
ls	ls ORLectures	lists everything in a specified directory
	ls -la ../GurobiStuff	shows hidden files as well as all the others
cd	cd /GurobiStuff	change to directory GurobiStuff
cd	cd ..	change to the parent directory
cd	cd ORLectures	change to directory ORLectures
cp	cp [dirname]/[file1] [directory]/.	copy a file from some directory into the current one
cp -r	cp -r	copy directory
diff	diff [filename1] [filename2]	to detect difference between the contents of files filename1 and filename2
top		to view the memory used up by different processes
rm	rm *.csv~	remove a file; removes all files that end with .csv~; files that end with .csv~ are usually (!) useless, they form when you are editing .csv files
	rm -rf GurobiStuff	remove directory GurobiStuff
mv	mv	moves a file to another location, also used to rename a file
grep	grep ”[phrase to find]” */*.py	to find a phrase in all python files (i.e., *.py)
ps x	ps x	lists all the processes currently running; lists their ids which then can be used to kill those processes
kill -9	kill -9 [process id]	terminates a specified process
make	make	to make a C++ program
	make 2>&1 |less	to pipe for the long output (you need it when your error list is sooooo long it does not fit in the xterm... happened to me a few times ;))
wc -l	wc -l [filename]	provides the number of lines in a file [filename]
find	find . -name ”[*.py]”	finds files in all subdirectories of the current one which have extension .py
	find . -name ”[*.py]” |xargs grep ”[csv]”	finds files in all subdirectories of the current one which have extension .py and prints on the screen every line containing ”csv” out of them
cat	cat [file1] [file2] > [file3]	concatenate file1 with file2 and put it into file3
	cat [file1] ≫ [file2]	appends file1 to the end of file2

Table 4.2: Some basic Linux utilities.

Bibliography

M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear Programming and Network Flows*. John Wiley & Sons, 2 edition, 1990.

Z. Drezner and H. W. Hamacher, editors. *Facility Location: Applications and Theory*. Springer-Verlag, 2004.

M. Ehrgott. *Multicriteria Optimization*. Springer, 2005.

A. Ernst, O.E. Gavriliouk, and L. Marquez. An efficient lagrangean heuristic for rental vehicle scheduling. *Computers and Operations Research*, 38:216–226, 2011.

M. R. Garey and D. S. Johnson. *Computers and Intractability (A guide to the theory of* NP*-completeness)*. W.H. Freeman and Company, New York, 1979.

H. W. Hamacher. Integer programming: Polyhedral analysis and algorithms. Master's thesis, University of Kaiserslautern, 2006. Lecture Notes.

H. W. Hamacher and K. Klamroth. *Linear and Network Optimization*. Vieweg & Sohn, 2000.

G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience series in discrete mathematics and optimization. John Wiley & Sons, 1999.

Y. Pochet and L. A. Wolsey. *Production Planning by Mixed Integer Programming*. Springer, 2000.

L. A. Wolsey. *Integer Programming*. Wiley-Interscience series in discrete mathematics and optimization. John Wiley & Sons, 1998.