Linear and Integer Programming
Lecture Notes

Marco Chiarandini

February 19, 2020

Contents

Chapter 1

Introduction

1.1 Operations Research

Operations Research (aka, Management Science, Analytics): is the discipline that uses a **scientific approach to decision making**. It seeks to determine how best to design and operate a system, usually under conditions requiring the allocation of scarce resources, by means of *quantitative methods*. The solution approaches lay at the intersection between the fields of *mathematics*, *computer science* and *statistics*. It encompasses a wide range of problem-solving techniques and methods applied in the pursuit of improved decision-making and efficiency: simulation, **mathematical optimization**, queuing theory and other stochastic-process models, Markov decision processes econometric methods, data envelopment analysis, neural networks, expert systems, decision analysis, and the analytic hierarchy process. This course focuses on mathematical optimization. In the modern context of data analytics, operations research contributes with *prescriptive* insights. Prescriptive insights are new quantitative indications on how to optimize the system, which although determined by the data collected are not inherent to the data. In contrast, a *descriptive* or *predictive* approach, that typically uses statistical methods or visualization techniques resumes the information present in the data.

The following are examples of areas with real-life applications that have been addressed with mathematical optimization:

- Production Planning and Inventory Control: planning the issue of orders for refilling warehouses avoiding stock out and satisfying space capacity limits.
- Budget Investment: given a budget and a number of projects, each with its own foreseen return and cost of resources, determining the projects to fund that would maximize the profit.
- Blending and Refining in the chemical industry
- Energy Planning: deciding when to activate cogeneration plants in order to meet the forecast demand of heat and electricity in the next few hours.
- Manpower Planning: scheduling the shifts of a group of nurses such that a department of an hospital is manned 24 hours a day with a given number of nurses and working agreements are respected; or in the airline and railways industries, rostering crews such that geographical locations and working agreements are satisfied.
- Packing Problems: filling containers with 3D packs without exceeding capacity and minimizing the free space.

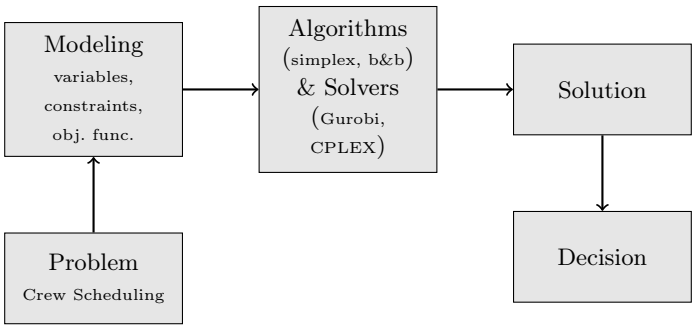


Figure 1.1:

- Cutting Problems: in textile or paper industry, cutting a paper roll in pieces to accommodate journals of different sizes while minimizing the waste.
- Vehicle Routing: in logistics, delivering products (oil, beer, food, etc.) to customers or retailers such that the total traveling distance is minimized and the capacity of the vehicle satisfied.
- Location Decisions: deciding where to open a set of warehouses having to ensure a satisfactory coverage of a number of retailers.
- Scheduling/Timetabling: in the manufacturing industry, schedule the sequences jobs in an assembly line; or in education, planning courses such that no two courses sharing students have overlap in time and a number of side constraints are satisfied. The need for timetables arises also in public transportation.

In all these contexts planning decisions must be made that relate to quantitative issues. For example, fewest number of people, shortest route, etc. On the other hand, not all plans are feasible: there are constraining rules. Moreover, there is a limited amount of available resources. Thus, it can be extremely difficult to figure out what to do.

In Figure 1.1, we depict a common scheme of the solution process in applied optimization. First, we observe the real life system and interview the persons involved to understand the problem. We then write a problem description in clear, plain English. This is useful to get back to the client and ensure that there are no misunderstandings. You should challenge your description by presenting cases that are not valid for the real life situation but that would be allowed by your description. This procedure is helpful to make the description precise and less prone to misinterpretations. Then, we are ready to introduce mathematical notation, that must make it impossible to misinterpret your model and removes all sources of disturbance. The real life objects are abstracted to sets, graphs, networks or other mathematical concepts. Then, the model made by known parameters, unknowns, objectives and constraints is formulated. Any word description is at this point removed. Finally, the model is solved on some test data and the solution interpreted and crosschecked with respect to reality. The central idea in this process is to build a mathematical model describing exactly what one wants, and what the "rules of the game" are.

1.2 Mathematical Modeling

The first step is to find out exactly what the decision maker needs to know: for example, which investment, which product mix, which task should a resource be used for? For each decision define

a **decision variable** of suitable type (continuous, integer, binary) according to the needs. For example, the decision whether to send a vehicle through an arc of a network or not can be modeled by a decision variable that can take only 0 and 1 values, that is, a binary decision variable. Clearly, in this context a value of 0.5 would not have any meaning in the physical world that we are trying to model. Next, identify the input constants for the model. These are called *parameters* and are values that are known and fixed once the specific instance of the problem is known. The next step is formulating the *objective function* to compute the benefit/cost in terms of decision variables and parameters. Finally, the **constraints** indicating the interplay between the different variables must be expressed in mathematical terms.

1.3 Resource Allocation

In the manufacturing industry, a common decision to take is which product mix to set in production. It is known as the **factory planning problem**. Suppose a factory makes two types of yogurts, a **Normal** one with a medium fermentation time and bacterial culture added and another one, **Lite**, with a long fermentation and bacterial culture added. One liter of **Normal** gives a profit of 6 Dkk while one liter of **Lite** gives 8 Dkk.

To produce the desired quantity of yogurt two processes are required. The heating of milk at 45 degrees to denature its proteins and allow fermentation and the addition of bacterial culture. For each liter of yogurt produced the amount of hours for the fermentation and the amount of bacterial cultures in decigrams are given below:

	Normal	Lite
Fermentation	5	10
Bacterial culture	4	4

The company has the following capacity for the fermentation given by the amount of hours in which the heating is possible because manned.

Heating capacity:	60 hours per week
Bacterial culture capacity:	40 decigrams per week

Question: How much of each type of yogurt, **Normal** and **Lite**, should the company produce to maximize the profit?

1.3.1 Mathematical model

Decision Variables

$$x_1 \geq 0 \text{ liters of product } \textbf{Normal}$$
$$x_2 \geq 0 \text{ liters of product } \textbf{Lite}$$

Objective Function

$$\max\ 6x_1 + 8x_2 \text{ maximize profit}$$

Constraints

$$\begin{aligned} 5x_1 + 10x_2 &\leq 60 \text{ heating capacity} \\ 4x_1 + 4x_2 &\leq 40 \text{ bacterial culture capacity} \end{aligned}$$

Calling for short the processes of heating and adding bacterial culture as A and B, and the two yogurt types as products 1 and 2, we can rewrite the model more compactly as:

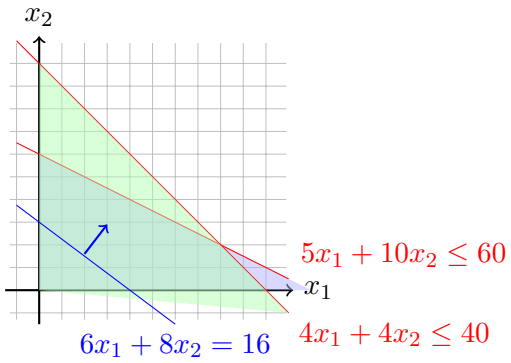


Figure 1.2: The graphical representation of the instance of factory planning problem defined on the given parameters, where 16 is an arbitrary value for the objective function. The goal is finding the value for the objective function that is largest possible.

$$\begin{array}{rrcrcl} \max & 6x_1 & + & 8x_2 & & \\ A: & 5x_1 & + & 10x_2 & \leq & 60 \\ B: & 4x_1 & + & 4x_2 & \leq & 40 \\ & & & x_1 & \geq & 0 \\ & & & x_2 & \geq & 0 \end{array}$$

In this model the variables (or unknowns) are x_1 and x_2 and the following are the given (or known) parameters, that we will call a_{ij}, b_i, c_j for $i \in \{A, B\}$ and $j \in \{1, 2\}$:

a_{ij}	1	2	b_i
A	5	10	60
B	4	4	40
c_j	6	8	

A graphical representation of the problem is given in Figure 1.2.

1.3.2 General Model

Notation Let $J = 1, 2, \ldots, n$ indexed by j be the set of products and let $I = 1, 2, \ldots, m$ indexed by i be the set of raw materials. For each raw material $i \in I$ the factory has b_i units at disposal. To produce a unit product of product $j \in J$, a_{ij} units of raw material i are needed. The values a_{ij} are called *technological coefficients*. They determine how much resources are needed to produce a given product given the current technology. The *prevailing market value* of the raw material i is ρ_i and a unit of the jth product can be sold at the market price σ_j. The return from the sell of the items produced is called *revenue*. The *profit* is the difference between the revenue and the total expenses due to the production, that is:

$$\text{profit} = \text{revenue} - \text{expenses}$$

The profit c_j derived from the sell of a unit of product j is given by $c_j = \sigma_j - \sum_{i=1}^{m} \rho_i a_{ij}$.

All the values introduced so far are given and their value for an instance of the problem is fixed. These are the *parameters* or *data* of the problem. We set out to determine the mix of products to produce. This is equivalent to decide the amount of each product to set in production. We denote

the amount of product j by x_j. Since the value of x_j, $j \in J$ is unknown initially, we call x_j the *variables* of our problem. Note that a negative value for x_j would not have a meaning, hence we know at least that $x_j \geq 0$ for all j in J.

Model We are thus ready to write a general mathematical model for the factory planning problem, which looks as follows.

$$\begin{array}{rrcrcrcrcl} \max & c_1x_1 & + & c_2x_2 & + & c_3x_3 & + \ldots + & c_nx_n & = & z \\ \text{subject to} & a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & + \ldots + & a_{1n}x_n & \leq & b_1 \\ & a_{21}x_1 & + & a_{22}x_2 & + & a_{23}x_3 & + \ldots + & a_{2n}x_n & \leq & b_2 \\ & \vdots & & & & & & & \vdots & \vdots \\ & a_{m1}x_1 & + & a_{m2}x_2 & + & a_{m3}x_3 & + \ldots + & a_{mn}x_n & \leq & b_m \\ & & & & & & & x_1, x_2, \ldots, x_n & \geq & 0 \end{array}$$

The words "subject to" are often abbreviated to "s.t.". More concisely the model can be written in scalar form as:

$$\max \sum_{j=1}^{n} c_j x_j \tag{1.1}$$

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, \ldots, m \tag{1.2}$$

$$x_j \geq 0, \quad j = 1, \ldots, n \tag{1.3}$$

Explanation Constraints (1.2) impose that the use of each resource i does not exceed the amount of resource available b_i. The objective function (1.1) calculates the total profit obtained by the sell of the products and it has to be maximized.

Further remarks The model can be rewritten in matrix form, by defining the following vectors and matrices, $\mathbf{x} \in \mathbb{R}^n, \mathbf{c} \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^m$:

$$\mathbf{c} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}, \quad A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}.$$

Our LP model can then be rewritten in matrix form as:

$$\begin{array}{rcl} \max \quad z & = & \mathbf{c}^T\mathbf{x} \\ A\mathbf{x} & \leq & \mathbf{b} \\ \mathbf{x} & \geq & 0 \end{array}$$

For our numerical example the scalar form becomes:

$$\begin{array}{rl} \max & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \ i = 1, \ldots, m \\ & x_j \geq 0, \ j = 1, \ldots, n \end{array}$$

$$\begin{array}{rrcrcl} \max & 6x_1 & + & 8x_2 & & \\ & 5x_1 & + & 10x_2 & \leq & 60 \\ & 4x_1 & + & 4x_2 & \leq & 40 \\ & & & x_1, x_2 & \geq & 0 \end{array}$$

and the matrix form:

$$\begin{aligned} \max\ & \mathbf{c}^T\mathbf{x} \\ & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

$$\mathbf{x} \in \mathbb{R}^n, \mathbf{c} \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^m$$

$$\begin{aligned} \max\ & \begin{bmatrix} 6 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ & \begin{bmatrix} 5 & 10 \\ 4 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 60 \\ 40 \end{bmatrix} \\ & x_1, x_2 \geq 0 \end{aligned}$$

It is important to check that no term in the model remains undefined. In particular, one has to check that if the model is in scalar form, then the quantifiers defining the free index of each constraint are properly stated. In the example above, these are: $i = 1, \ldots, m$ for the first set of constraints (1.2) and $j = 1, \ldots, n$ for the second set of constraints (1.3). If the model is in matrix form, then the size of the arrays involved must be expressed, as done in the last row of the left size model above.

1.3.3 Duality

Above we saw the factory planning problem from the perspective of the company owning the raw materials. We assumed that it was convenient for the company to produce and sell products. However, a plausible alternative would be to close the factory and sell the raw material to the market. What would be the price of the raw material such that this deal becomes feasible and attractive? To answer this question we have to solve a *resource valuation problem*.

Let's take the point of view of an outside company who has to make an offer for buying the raw materials. From this standpoint the unknowns that are to be determined are the values of a unit of raw material i, which we indicate by z_i, for $i = 1, 2, \ldots, m$. These values are the *variables* of the problem. The total expenses for buying the raw materials are given by $\sum_{i=1}^{m} b_i z_i$. The buying company is interested in minimizing precisely this value. However, the value z_i has to be larger than the prevailing unit market value of material i, ρ_i, otherwise the price would contradict the market and the owning company would prefer selling to someone else. Similarly, for each single product $j \in J$ the opportunity cost derived from producing a unit of product has to be larger than the unitary price σ_j of the product. If this was not true, then the owning company would not sell the raw material but rather use it to produce the product and sell that one instead.

From the perspective of the owning company the valuation problem consists in determining the minimum price this company should accept for selling all its assets instead of using them for the production. From the owning company standpoint the value $\sum_{i=1}^{m} b_i z_i$ is called the opportunity cost of owning the raw material. It is the value that could be obtained by selling all raw material and closing the factory. It is the lost opportunity with respect to producing and selling the products.

We can therefore write the model for the resource valuation problem as follows:

$$\min \sum_{i=1}^{m} b_i z_i \tag{1.4}$$

$$\sum_{i=1}^{m} z_i a_{ij} \geq \sigma_j, \quad j = 1 \ldots n \tag{1.5}$$

$$z_i \geq \rho_i, \quad i = 1 \ldots m \tag{1.6}$$

Constraints (1.5) and (1.6) ensure that we are not contradicting the market while the objective (1.4) aims at making the deal appealing for the buying company.

Let $y_i = z_i - \rho_i$ be the *markup* that the owning company would make by reselling the raw material at the price z_i with respect to the price ρ_i at which it bought it. Then we can rewrite the model above as:

$$\min \sum_{i=1}^{m} y_i b_i + \sum_i \rho_i b_i \tag{1.7}$$

$$\sum_{i=1}^{m} y_i a_{ij} \geq c_j, \quad j = 1 \dots n \tag{1.8}$$

$$y_i \geq 0, \quad i = 1 \dots m \tag{1.9}$$

where in the objective function the term $\sum_i \rho_i b_i$ is always constant and does not impact the solution. The problem we wrote is known as the *dual* of the previous resource allocation problem, which gets consequently the name of *primal*. The two models are one the dual of the other.

$$\begin{aligned} \max \ & u = \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, \dots, m \\ & x_j \geq 0, \quad j = 1, \dots, n \end{aligned} \qquad \begin{aligned} \min \ & w = \sum_{i=1}^{m} y_i b_i \\ & \sum_{i=1}^{m} y_i a_{ij} \geq c_j, \quad j = 1 \dots n \\ & y_i \geq 0, \quad i = 1 \dots m \end{aligned}$$

As we will see the optimal value of the primal problem u^* is the same as the optimal value of the dual problem w^*, ie, $u^* = w^*$.

1.4 Diet Problem

The Diet Problem belongs to the family of blending problems. We wish to select a set of foods that will satisfy a set of daily nutritional requirements at minimum cost.

min cost/weight
subject to nutrition requirements:
    eat enough but not too much of Vitamin A
    eat enough but not too much of Sodium
    eat enough but not too much of Calories
    ...

The problem was motivated in the 1930s and 1940s by the US army. It was first formulated as a **linear programming problem** by George Stigler.

Suppose there are:

- 3 foods available, corn, milk, and bread, and
- there are restrictions on the number of calories (between 2000 and 2250) and the amount of Vitamin A (between 5,000 and 50,000)

Food	Cost per serving	Vitamin A	Calories
Corn	\$0.18	107	72
2% Milk	\$0.23	500	121
Wheat Bread	\$0.05	0	65

1.5 The Mathematical Model

Parameters (given data)

F = **set** of foods
N = **set** of nutrients

a_{ij} = amount of nutrient j in food i, $\forall i \in F$, $\forall j \in N$
c_i = cost per serving of food $i, \forall i \in F$
F_{mini} = minimum number of required servings of food $i, \forall i \in F$
F_{maxi} = maximum allowable number of servings of food $i, \forall i \in F$
N_{minj} = minimum required level of nutrient $j, \forall j \in N$
N_{maxj} = maximum allowable level of nutrient $j, \forall j \in N$

Decision Variables

x_i = number of servings of food i to purchase/consume, $\forall i \in F$

Objective Function

Minimize the total cost of the food

$$\text{Minimize} \sum_{i \in F} c_i x_i$$

Constraints

Constraint Set 1: For each nutrient $j \in N$, at least meet the minimum required level

$$\sum_{i \in F} a_{ij} x_i \geq N_{minj}, \qquad \forall j \in N$$

Constraint Set 2: For each nutrient $j \in N$, do not exceed the maximum allowable level.

$$\sum_{i \in F} a_{ij} x_i \leq N_{maxj}, \qquad \forall j \in N$$

Constraint Set 3: For each food $i \in F$, select at least the minimum required number of servings

$$x_i \geq F_{mini}, \qquad \forall i \in F$$

Constraint Set 4: For each food $i \in F$, do not exceed the maximum allowable number of servings.

$$x_i \leq F_{maxi}, \qquad \forall i \in F$$

All together we obtain the following system of equalities and inequalities that gives the linear programming problem:

$$\begin{aligned}
\min \quad & \sum_{i \in F} c_i x_i \\
\sum_{i \in F} a_{ij} x_i & \geq N_{minj}, \qquad \forall j \in N \\
\sum_{i \in F} a_{ij} x_i & \leq N_{maxj}, \qquad \forall j \in N \\
x_i & \geq F_{mini}, \qquad \forall i \in F \\
x_i & \leq F_{maxi}, \qquad \forall i \in F
\end{aligned}$$

The linear programming model by Stigler consisted of 9 equations in 77 variables. He guessed an optimal solution using a heuristic method. In 1947, the National Bureau of Standards used the newly developed simplex method by Dantzig to solve Stigler's model. It took 9 clerks using hand-operated desk calculators 120 man days to solve for the optimal solution. The original instance is available at: http://www.gams.com/modlib/libhtml/diet.htm

1.5.1 Solving LP Models in Practice

There are two main approaches to pass a model to a solver. A dedicated modeling language allows to *declare* the problem in a very similar way to the mathematical model written above and then call the solver for the solution. Examples are AMPL, ZIMPL, GAMS, GNU MathProg. Alternatively, it is possible to use libraries from common programming languages. Python offers a good compromise between these two approaches, in that, even if it is an imperative, procedural language, it allows to define the model in a way very similar to those of modeling languages.

Let's first have a look at how things look in a modeling language as AMPL. A good way to proceed is to separate the model from the data in different files.

```
# diet.mod
set NUTR;
set FOOD;

param cost {FOOD} > 0;
param f_min {FOOD} >= 0;
param f_max { i in FOOD} >= f_min[i];
param n_min { NUTR } >= 0;
param n_max {j in NUTR } >= n_min[j];
param amt {NUTR,FOOD} >= 0;

var Buy { i in FOOD} >= f_min[i], <= f_max[i]

minimize total_cost: sum { i in FOOD } cost [i] * Buy[i];
subject to diet { j in NUTR }:
      n_min[j] <= sum {i in FOOD} amt[i,j] * Buy[i] <= n_max[j];
```

```
# diet.dat
data;

set NUTR := A B1 B2 C ;
set FOOD := BEEF CHK FISH HAM MCH MTL SPG TUR;

param: cost f_min f_max :=
  BEEF 3.19 0 100
  CHK 2.59 0 100
  FISH 2.29 0 100
  HAM 2.89 0 100
  MCH 1.89 0 100
  MTL 1.99 0 100
  SPG 1.99 0 100
  TUR 2.49 0 100 ;

param: n_min n_max :=
   A 700 10000
   C 700 10000
   B1 700 10000
   B2 700 10000 ;

param amt (tr):
          A C B1 B2 :=
   BEEF 60 20 10 15
   CHK 8 0 20 20
   FISH 8 10 15 10
   HAM 40 40 35 10
   MCH 15 35 15 15
   MTL 70 30 15 15
   SPG 25 50 25 15
   TUR 60 20 15 10 ;
```

Below is instead an example in Gurobi Python on a different dataset.

```
# Model diet.py
m = Model("diet")

# Create decision variables for the foods to buy
buy = {}
for f in foods:
    buy[f] = m.addVar(obj=cost[f], name=f)

# The objective is to minimize the costs
m.modelSense = GRB.MINIMIZE

# Update model to integrate new variables
m.update()

# Nutrition constraints
for c in categories:
    m.addConstr(
      quicksum(nutritionValues[f,c] * buy[f] for f in foods) <= maxNutrition[c], name
          =c+'max')
```

```
    m.addConstr(
      quicksum(nutritionValues[f,c] * buy[f] for f in foods) >= minNutrition[c], name
          =c+'min')

# Solve
m.optimize()
```

```
# data.py

from gurobipy import *

categories, minNutrition, maxNutrition = multidict({
  'calories': [1800, 2200],
  'protein': [91, GRB.INFINITY],
  'fat': [0, 65],
  'sodium': [0, 1779] })

foods, cost = multidict({
  'hamburger': 2.49,
  'chicken': 2.89,
  'hot dog': 1.50,
  'fries': 1.89,
  'macaroni': 2.09,
  'pizza': 1.99,
  'salad': 2.49,
  'milk': 0.89,
  'ice cream': 1.59 })

# Nutrition values for the foods
nutritionValues = {
  ('hamburger', 'calories'): 410,
  ('hamburger', 'protein'): 24,
  ('hamburger', 'fat'): 26,
  ('hamburger', 'sodium'): 730,
  ('chicken', 'calories'): 420,
  ('chicken', 'protein'): 32,
  ('chicken', 'fat'): 10,
  ('chicken', 'sodium'): 1190,
  ('hot dog', 'calories'): 560,
  ('hot dog', 'protein'): 20,
  ('hot dog', 'fat'): 32,
  ('hot dog', 'sodium'): 1800,
  ('fries', 'calories'): 380,
  ('fries', 'protein'): 4,
  ('fries', 'fat'): 19,
  ('fries', 'sodium'): 270,
  ('macaroni', 'calories'): 320,
  ('macaroni', 'protein'): 12,
  ('macaroni', 'fat'): 10,
  ('macaroni', 'sodium'): 930,
  ('pizza', 'calories'): 320,
  ('pizza', 'protein'): 15,
  ('pizza', 'fat'): 12,
```

```
('pizza', 'sodium'): 820,
('salad', 'calories'): 320,
('salad', 'protein'): 31,
('salad', 'fat'): 12,
('salad', 'sodium'): 1230,
('milk', 'calories'): 100,
('milk', 'protein'): 8,
('milk', 'fat'): 2.5,
('milk', 'sodium'): 125,
('ice cream', 'calories'): 330,
('ice cream', 'protein'): 8,
('ice cream', 'fat'): 10,
('ice cream', 'sodium'): 180 }
```

1.6 A Brief History of Linear Programming (LP)

Related to linear programming problems are systems of linear equations, which we study in Linear Algebra.

It is impossible to find out who knew what, who knew when, who knew first. The Egyptians and Babylonians considered about 2000 B.C. the solution of special linear equations. But, of course, they described examples and did not describe the methods in "today's style". What we call "**Gaussian elimination**" today has been explicitly described in the Chinese work "Nine Books of Arithmetic", which is a compendium written in the period 2010 B.C. to 9 A.D., but the methods were probably known long before that. Gauss, by the way, never described "Gaussian elimination". He just used it and stated that the linear equations he used can be solved "per eliminationem vulgarem"

The origins of Linear Programming date back to Newton, Leibnitz, Lagrange, etc.

- In 1827, Fourier described a variable elimination method for **systems of linear inequalities**, today often called Fourier-Motzkin elimination (Motzkin, 1937). It can be turned into an LP solver but inefficient.
- In 1932, Leontief (1905-1999) studies the Input-Output model to represent interdependencies between branches of a national economy (1976 Nobel prize).
- In 1939, Kantorovich (1912-1986) layed down the foundations of linear programming. He won the Nobel prize in economics in 1975 with Koopmans on Optimal use of scarce resources: foundation and economic interpretation of LP.
- The math subfield of **Linear Programming** was created by George Dantzig, John von Neumann (Princeton), and Leonid Kantorovich in the 1940s.
- In 1947, Dantzig (1914-2005) invented the **(primal) simplex algorithm** working for the US Air Force at the Pentagon. (program=plan)
- In 1954, Lemke describes the dual simplex algorithm. In 1954, Dantzig and Orchard Hays present the revised simplex algorithm.
- In 1970, Victor Klee and George Minty created an example that showed that the classical simplex algorithm has exponential worst-case behavior.

- In 1979, L. Khachain found a new **efficient** algorithm, the Ellipsoid method, for linear programming. It was terribly slow.
- In 1984, Karmarkar discovered yet another new **efficient** algorithm for linear programming, the interior point method. It proved to be a strong competitor for the simplex method.

Some other important marks in the history of optimization are the following:

- In 1951, Nonlinear Programming began with the Karush-Kuhn-Tucker Conditions.
- In 1952, Commercial Applications and Software began.
- In 1950s, Network Flow Theory began with the work of Ford and Fulkerson.
- In 1955, Stochastic Programming began.
- In 1958, Integer Programming began with cutting planes by R. E. Gomory.
- In 1962, Complementary Pivot Theory.

1.7 Fourier Motzkin elimination method

Suppose A is a matrix from $\mathbb{Q}^{m\times n}$ and $\mathbf{b}$ a vector from $\mathbb{Q}^n$. Does the *system of linear inequalities* $A\mathbf{x} \leq \mathbf{b}$ have a solution?

The Fourier Motzkin elimination method works with the following steps:

1. transform the system into another by eliminating some variables such that the two systems have the same solutions over the remaining variables.
2. reduce to a system of constant inequalities that can be easily decided

Let $M = \{1 \dots m\}$ be the set that indexes the constraints. For a variable $j = 1...n$ let partition the rows of the matrix A in those in which x_j appears with a positive, negative and null coefficient, respectively, that is:

$$\begin{aligned} N &= \{i \in M \mid a_{ij} < 0\} \\ Z &= \{i \in M \mid a_{ij} = 0\} \\ P &= \{i \in M \mid a_{ij} > 0\} \end{aligned}$$

Let x_r be the variable to eliminate.

$$\begin{cases} x_r \geq b'_{ir} - \sum_{k=1}^{r-1} a'_{ik}x_k, & a_{ir} < 0 \\ x_r \leq b'_{ir} - \sum_{k=1}^{r-1} a'_{ik}x_k, & a_{ir} > 0 \\ \text{all other constraints}(i \in Z) \end{cases} \qquad \begin{cases} x_r \geq A_i(x_1, \dots, x_{r-1}), & i \in N \\ x_r \leq B_i(x_1, \dots, x_{r-1}), & i \in P \\ \text{all other constraints}(i \in Z) \end{cases}$$

Hence the original system is equivalent to

$$\begin{cases} \max\{A_i(x_1, \dots, x_{r-1}), i \in N\} \leq x_r \leq \min\{B_i(x_1, \dots, x_{r-1}), i \in P\} \\ \text{all other constraints}(i \in Z) \end{cases}$$

which is equivalent to

$$\begin{cases} A_i(x_1, \dots, x_{r-1}) \leq B_j(x_1, \dots, x_{r-1}) & i \in N, j \in P \\ \text{all other constraints}(i \in Z) \end{cases}$$

we eliminated x_r but:

$$\begin{cases} |N| \cdot |P| \text{ inequalities} \\ |Z| \text{ inequalities} \end{cases}$$

After d iterations if $|P| = |N| = m/2$ exponential growth: $1/4^d(m/2)^{2^d}$

Example

$$\begin{array}{rcrcl} -7x_1 & + & 6x_2 & \leq & 25 \\ x_1 & - & 5x_2 & \leq & 1 \\ x_1 & & & \leq & 7 \\ -x_1 & + & 2x_2 & \leq & 12 \\ -x_1 & - & 3x_2 & \leq & 1 \\ 2x_1 & - & x_2 & \leq & 10 \end{array}$$

Let x_2 be the variable we choose to eliminate:

$$N = \{2, 5, 6\}, \quad Z = \{3\}, \quad P = \{1, 4\}$$

We obtain $|Z \cup (N \times P)| = 7$ constraints.

By adding one variable and one inequality, Fourier-Motzkin elimination can be turned into an LP solver. How?

Chapter 2

The Simplex Method

In this chapter we study the *simplex method* or (simplex algorithm). It was the first algorithm to solve linear programming problems proposed in 1947 by George Dantzig in a technical report "Maximization of a Linear Function of Variables Subject to Linear Inequalities" [Dantzig, 1951].

2.1 Preliminaries

We recall some definitions from linear algebra that will be useful to motivate and describe the simplex algorithm.

- $\mathbb{R}$: set of real numbers
  $\mathbb{N} = \{1, 2, 3, 4, ...\}$: set of natural numbers (positive integers)
  $\mathbb{Z} = \{..., -3, -2, -1, 0, 1, 2, 3, ...\}$: set of all integers
  $\mathbb{Q} = \{p/q \mid p, q \in \mathbb{Z}, q \neq 0\}$: set of rational numbers

- We will often use matrix notation. Vectors are always meant to be column vectors. The scalar product: $\mathbf{y}^T\mathbf{x} = \sum_{i=1}^{n} y_i x_i$

- For a set of k vectors from the vector space $\mathbb{R}^n$, $\mathbf{v}_1, \mathbf{v}_2 \ldots, \mathbf{v}_k \in \mathbb{R}^n$, the vector $\mathbf{x} \in \mathbb{R}^n$ is a **linear combination** if there exist $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_k]^T \in \mathbb{R}^k$ such that

$$\mathbf{x} = \lambda_1\mathbf{v}_1 + \cdots + \lambda_k\mathbf{v}_k = \sum_{i=1}^{k} \lambda_i\mathbf{v}_i$$

  The values $\lambda_1, \ldots, \lambda_k$ are called coefficients of the linear combination.

  If $\boldsymbol{\lambda}^T\mathbf{1} = 1$, then the linear combination is an *affine combination*. Here, $\mathbf{1}$ is a vector in $\mathbb{R}^k$ with all components equal to 1. Hence, $\boldsymbol{\lambda}^T\mathbf{1} = 1$ corresponds to saying $\sum_{i=1}^{k} \lambda_i = 1$.

  If $\boldsymbol{\lambda} \geq \mathbf{0}$ then the linear combination is a *conic combination*.

  If $\boldsymbol{\lambda} \geq \mathbf{0}$ and $\boldsymbol{\lambda}^T\mathbf{1} = 1$ then the linear combination is a *convex combination*.

- A set S of vectors from $\mathbb{R}^n$ is **linearly (affine) independent** if no element of it can be expressed as linear (affine) combination of the others
  Eg: $S \subseteq \mathbb{R}^n \implies$ max n lin. indep. ($n + 1$ lin. aff. indep.)

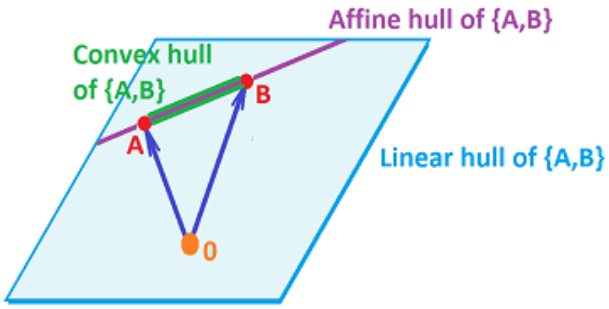
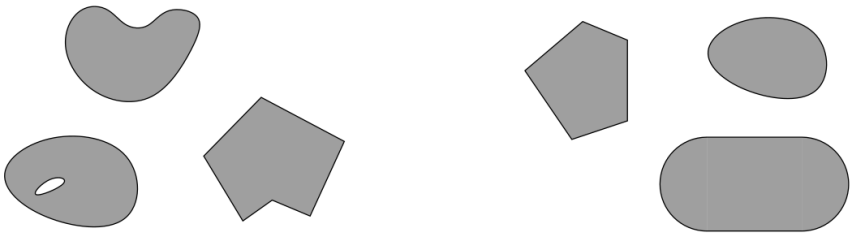


Figure 2.1: In $\mathbb{R}^2$, a plane passing through the origin and the points A and B is a linear hull of the two points. The line passing through A and B but not passing through the origin is an affine hull of the two points. Finally, the segment between A and B is the convex hull of the two points.



nonconvex convex

Figure 2.2:

- For a set of points $S \subseteq \mathbb{R}^n$ linear hull (aka linear span), affine hull, conic hull and convex hull are respectively the sets:

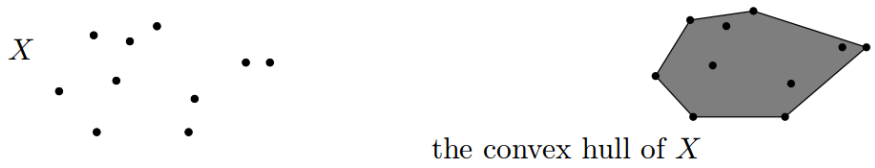
$$\begin{aligned}
\text{lin}(S) &= \{\lambda_1 \mathbf{v}_1 + \cdots + \lambda_k \mathbf{v}_k \mid k \geq 0; \mathbf{v}_1, \cdots, \mathbf{v}_k \in S; \lambda_1, \cdots, \lambda_k \in \mathbb{R}\} \\
\text{aff}(S) &= \{\lambda_1 \mathbf{v}_1 + \cdots + \lambda_k \mathbf{v}_k \mid k \geq 1; \mathbf{v}_1, \cdots, \mathbf{v}_k \in S; \lambda_1, \cdots, \lambda_k \in \mathbb{R}; \lambda_1 + \cdots + \lambda_k = 1\} \\
\text{cone}(S) &= \{\lambda_1 \mathbf{v}_1 + \cdots + \lambda_k \mathbf{v}_k \mid k \geq 0; \mathbf{v}_1, \cdots, \mathbf{v}_k \in S; \lambda_1, \cdots, \lambda_k \geq 0\} \\
\text{conv}(S) &= \{\lambda_1 \mathbf{v}_1 + \cdots + \lambda_k \mathbf{v}_k \mid k \geq 0; \mathbf{v}_1, \cdots, \mathbf{v}_k \in S; \lambda_1, \cdots, \lambda_k \geq 0; \lambda_1 + \cdots + \lambda_k = 1\}
\end{aligned}$$

  See Figure 2.1 for a geometrical interpretation of these concepts. The set of points can be the vectors made by the columns of an $n \times m$ matrix A, hence the previous definitions can refer to a matrix as well.

- **convex set**: if $\mathbf{x}, \mathbf{y} \in S$ and $0 \leq \lambda \leq 1$ then $\lambda\mathbf{x} + (1-\lambda)\mathbf{y} \in S$

- **convex function** if its epigraph $\{(x, y) \in \mathbb{R}^2 : y \geq f(x)\}$ is a convex set or $f : X \to \mathbb{R}$, if $\forall x, y \in X, \lambda \in [0, 1]$ it holds that $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$. See Fig 2.2.

- Given a set of points $X \subseteq \mathbb{R}^n$ the **convex hull conv**(X) is the convex linear combination of the points $\boxed{\textbf{conv}(X) = \{\lambda_1 \mathbf{x}_1 + \lambda_2 \mathbf{x}_2 + \ldots + \lambda_n x_n \mid \mathbf{x}_i \in X, \lambda_1, \ldots, \lambda_n \geq 0 \text{ and } \sum_i \lambda_i = 1\}}$



the convex hull of X

- **rank** of a matrix for columns (= for rows)
  if (m, n)-matrix has rank $= \min\{m, n\}$ then the matrix is full rank
  if (n, n)-matrix is full rank then it is regular and admits an inverse

- $G \subseteq \mathbb{R}^n$ is an **hyperplane** if $\exists\, \mathbf{a} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ and $\alpha \in \mathbb{R}$:

$$G = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}^T\mathbf{x} = \alpha\}$$

- $H \subseteq \mathbb{R}^n$ is an **halfspace** if $\exists\, \mathbf{a} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ and $\alpha \in \mathbb{R}$:

$$H = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}^T\mathbf{x} \leq \alpha\}$$

  ($\mathbf{a}^T\mathbf{x} = \alpha$ is a supporting hyperplane of H)

- a set $S \subset \mathbb{R}^n$ is a **polyhedron** if $\exists\, m \in \mathbb{Z}^+, A \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^m$:

$$P = \{\mathbf{x} \in \mathbb{R} \mid A\mathbf{x} \leq \mathbf{b}\} = \bigcap_{i=1}^{m} \{\mathbf{x} \in \mathbb{R}^n \mid A_{i\cdot}\mathbf{x} \leq b_i\}$$

- a polyhedron P is a **polytope** if it is bounded: $\exists\, B \in \mathbb{R}, B > 0$:

$$P \subseteq \{\mathbf{x} \in \mathbb{R}^n \mid \|\mathbf{x}\| \leq B\}$$

- Theorem: every polyhedron $P \neq \mathbb{R}^n$ is determined by finitely many halfspaces

- General optimization problem: $\max\{\varphi(\mathbf{x}) \mid \mathbf{x} \in F\}, F$ is feasible region for $\mathbf{x}$. Note: if F is open, eg, $x < 5$ then: $\sup\{x \mid x < 5\}$
  The *sumpreum* is the least element of $\mathbb{R}$ greater or equal than any element in F

- If A and $\mathbf{b}$ are made of rational numbers, $P = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \leq \mathbf{b}\}$ is a **rational polyhedron**

- A **face** of P is $F = \{\mathbf{x} \in P \mid \mathbf{ax} = \alpha\}$, where $\mathbf{a}$ is a given vector of real numbers and α is a given scalar number. Hence F is either P itself or the intersection of P with a supporting hyperplane. It is said to be **proper** if $F \neq \emptyset$ and $F \neq P$. In Figure 2.3, a face is a side of the tetrahedron, an edge and a vertex.

- A point $\mathbf{x}$ for which $\{\mathbf{x}\}$ is a face is called a **vertex** of P and also a **basic solution** of $A\mathbf{x} \leq \mathbf{b}$ (a vertex of a polytope is a face of dimension 0). There are four vertices in the polytope of Figure 2.3, left.

- A **facet** is a maximal face distinct from P. $\mathbf{cx} \leq \mathbf{d}$ is facet defining if $\mathbf{cx} = \mathbf{d}$ is a supporting hyperplane of P (a facet of a polytope in $\mathbb{R}^n$ has dimension $n-1$). In Figure 2.3, left, the four sides are the facets of the polytope.

Figure 2.3: Two examples of polytopes in $\mathbb{R}^3$: a tetrahedron and dodecahedron.

Depending on whether we study systems of linear equalities or inequalities and using integer or continuous variables we may be in a different field of mathematics:

- Linear algebra studies linear equations
- Integer linear algebra studies linear diophantine equations
- Linear programming studies linear inequalities (simplex method)
- Integer linear programming studies linear diophantine inequalities

2.1.1 Linear Programming Problem

Input: a matrix $A \in \mathbb{R}^{m \times n}$ and column vectors $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$
Task: Decide which one of the three is true:

1. $\{\mathbf{x} \in \mathbb{R}^n; A\mathbf{x} \leq \mathbf{b}\}$ is empty (**problem infeasible**), or
2. a column vector $\mathbf{x} \in \mathbb{R}^n$ such that $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{c}^T\mathbf{x}$ is max can be found
3. for all $\alpha \in \mathbb{R}$ there is an $\mathbf{x} \in \mathbb{R}^n$ with $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{c}^T\mathbf{x} > \alpha$ (**problem unbounded**)

2.1.2 Fundamental Theorem of LP

Theorem 2.1 (Fundamental Theorem of Linear Programming)**.** *Given:*

$$\min\{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} \in P\} \text{ where } P = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \leq \mathbf{b}\}$$

If P is a bounded polyhedron and not empty and $\mathbf{x}^*$ is an optimal solution to the problem, then:

- $\mathbf{x}^*$ *is an extreme point (vertex) of P, or*
- $\mathbf{x}^*$ *lies on a face $F \subset P$ of an optimal solution*

Proof. The first part of the proof shows by contradiction that $\mathbf{x}^*$ must be on the boundary of P. Then, if $\mathbf{x}^*$ is not a vertex, it is a convex combination of vertices and it shows that all points of the convex combination are also optimal.

For the first part, suppose, for the sake of contradiction, that $\mathbf{x}^* \in \text{int}(P)$, that is, is interior to P, not a vertex. Then there exists some $\epsilon > 0$ such that the ball of radius ϵ centered at $\mathbf{x}^*$ is

contained in P, that is $B_\epsilon(\mathbf{x}^*) \subset P$. Therefore, moving in the direction $\mathbf{c}$ of improvement of the objective function $\mathbf{c}^T\mathbf{x}$:

$$\mathbf{x}^* - \frac{\epsilon}{2}\frac{\mathbf{c}}{||\mathbf{c}||} \in P$$

and substituting in the objective function

$$\mathbf{c}^T\left(\mathbf{x}^* - \frac{\epsilon}{2}\frac{\mathbf{c}}{||\mathbf{c}||}\right) = \mathbf{c}^T\mathbf{x}^* - \frac{\epsilon}{2}\frac{\mathbf{c}^T\mathbf{c}}{||\mathbf{c}||} = c^T\mathbf{x}^* - \frac{\epsilon}{2}||\mathbf{c}|| < \mathbf{c}^T\mathbf{x}^*.$$

Hence $\mathbf{x}^*$ is not an optimal solution, a contradiction. Therefore, $\mathbf{x}^*$ must live on the boundary of P. If $\mathbf{x}^*$ is not a vertex itself, it must be the convex combination of vertices of P, say $\mathbf{x}_1, \ldots, \mathbf{x}_t$. Then $\mathbf{x}^* = \sum_{i=1}^t \lambda_i \mathbf{x}_i$ with $\lambda_i \geq 0$ and $\sum_{i=1}^t \lambda_i = 1$. Observe that

$$0 = \mathbf{c}^T\left(\left(\sum_{i=1}^t \lambda_i \mathbf{x}_i\right) - \mathbf{x}^*\right) = \mathbf{c}^T\left(\sum_{i=1}^t \lambda_i(\mathbf{x}_i - \mathbf{x}^*)\right) = \sum_{i=1}^t \lambda_i(\mathbf{c}^T\mathbf{x}_i - \mathbf{c}^T\mathbf{x}^*).$$

Since $\mathbf{x}^*$ is an optimal solution, all terms in the sum are non-negative. Since the sum is equal to zero, we must have that each individual term is equal to zero. Hence, $\mathbf{c}^T\mathbf{x}^* = \mathbf{c}^T\mathbf{x}_i$ for each $\mathbf{x}_i$, so every $\mathbf{x}_i$ is also optimal, and therefore all points on the face whose vertices are $\mathbf{x}_1, \ldots, \mathbf{x}_t$, are optimal solutions.

□

It follows from the theorem that the optimal solution is at the intersection of hyperplanes supporting halfspaces. Since there are finitely many halfspaces to describe a polyhedron, then there are finitely many possibilities to look for optimal solutions. A solution method could proceed as follows: write all inequalities as equalities and solve all $\binom{n}{m}$ systems of linear equalities (n is the number of variables, m is the number of equality constraints). For each point we need then to check if it is feasible and if it is best in cost (optimality condition). We can solve each system of linear equations by Gaussian elimination.

Checking all $\binom{n}{m}$ may result in a lot of work. Recall that [Cormen et al., 2009, pag. 1097]

$$\binom{n}{m} = O\left(\left(\frac{en}{m}\right)^m\right)$$

hence the asymptotic upper bound is an exponential function. The simplex method, as we will see, tries to be smarter and only visit some of the vertices. Shortly said, it finds a solution that is at the intersection of some m hyperplanes. Then it tries to systematically produce the other points by exchanging one hyperplane with another without loosing feasibility and at each point it checks optimality.

2.2 Systems of Linear Equations

Before we proceed to the treatment of the simplex algorithm we revise the solution methods for a system of linear equations $A\mathbf{x} = \mathbf{b}$. There are three methods to solve $A\mathbf{x} = \mathbf{b}$ if A is $n \times n$ and $|A| \neq 0$:

1. Gaussian elimination
2. By inverse matrix: find A^{-1}, then calculate $\mathbf{x} = A^{-1}\mathbf{b}$
3. Cramer's rule

However, if A is $m \times n$ and $m \neq n$ or if $|A| = 0$ then we can only use Gaussian elimination.

Gaussian Elimination **Gaussian Elimination** works in two steps:

1. Forward elimination: reduces the system to triangular (row echelon) form by elementary row operations
   - multiply a row by a non-zero constant
   - interchange two rows
   - add a multiple of one row to another
2. Back substitution

Alternatively, one can compute the *reduced row echelon form* (RREF) of A and read immediately the solution from there.

We illustrate this procedure on the following numerical example.

$$\begin{aligned} 2x + y - z &= 8 && (R1)\\ -3x - y + 2z &= -11 && (R2)\\ -2x + y + 2z &= -3 && (R3) \end{aligned}$$

On the right side we perform the computations. The style is taken from `emacs org mode`, that offers a convenient environment for working with tables.

$$\begin{aligned} 2x + y - z &= 8 && (R1)\\ -3x - y + 2z &= -11 && (R2)\\ -2x + y + 2z &= -3 && (R3) \end{aligned}$$

```
|----+----+----+----+-----|
| R1 |  2 |  1 | -1 |   8 |
| R2 | -3 | -1 |  2 | -11 |
| R3 | -2 |  1 |  2 |  -3 |
|----+----+----+----+-----|
```

$$\begin{aligned} 2x + y - z &= 8 && (R1)\\ + \tfrac{1}{2}y + \tfrac{1}{2}z &= 1 && (R2)\\ + 2y + 1z &= 5 && (R3) \end{aligned}$$

```
|---------------+---+-----+------+---|
| R1'=1/2 R1    | 1 | 1/2 | -1/2 | 4 |
| R2'=R2+3/2 R1 | 0 | 1/2 | 1/2  | 1 |
| R3'=R3+R1     | 0 | 2   | 1    | 5 |
|---------------+---+-----+------+---|
```

$$\begin{aligned} 2x + y - z &= 8 && (R1)\\ + \tfrac{1}{2}y + \tfrac{1}{2}z &= 1 && (R2)\\ - z &= 1 && (R3) \end{aligned}$$

```
|-------------+---+-----+------+---|
| R1'=R1      | 1 | 1/2 | -1/2 | 4 |
| R2'=2 R2    | 0 |   1 |    1 | 2 |
| R3'=R3-4 R2 | 0 |   0 |   -1 | 1 |
|-------------+---+-----+------+---|
```

$$\begin{aligned} 2x + y - z &= 8 && (R1)\\ + \tfrac{1}{2}y + \tfrac{1}{2}z &= 1 && (R2)\\ - z &= 1 && (R3) \end{aligned}$$

```
|---------------+---+-----+---+-----|
| R1'=R1-1/2 R3 | 1 | 1/2 | 0 | 7/2 |
| R2'=R2+R3     | 0 |   1 | 0 |   3 |
| R3'=-R3       | 0 |   0 | 1 |  -1 |
|---------------+---+-----+---+-----|
```

$$\begin{aligned} x &= 2 && (R1)\\ y &= 3 && (R2)\\ z &= -1 && (R3) \end{aligned}$$

```
|---------------+---+---+---+----+
| R1'=R1-1/2 R2 | 1 | 0 | 0 |  2 | => x=2
| R2'=R2        | 0 | 1 | 0 |  3 | => y=3
| R3'=R3        | 0 | 0 | 1 | -1 | => z=-1
|---------------+---+---+---+----+
```

Gaussian elimination can be implemented in polynomial time $O(n^2m)$ but some care must be applied to guarantee that all the numbers during the run can be represented by polynomially bounded bits.

By inverse matrix

$$A\mathbf{x} = \mathbf{b}$$

$$\mathbf{x} = A^{-1}\mathbf{b}$$

Calculating the inverse of a matrix is computationally very expensive. In practice (that is, in computer systems) the computation is rather performed via *LU factorization*: each matrix can be expressed as the product of a permutation matrix P, a lower triangular matrix L (with diagonal elements equal to 1) and upper triangular matrix U.

$$A = PLU$$

For our previous example:

$$\begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 8 \\ -11 \\ -3 \end{bmatrix}$$

$$\begin{aligned} A = \begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} &= P \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -3/2 & 1 & 0 \\ -1 & 4 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & -1 \\ 0 & 1/2 & 1/2 \\ 0 & 0 & -1 \end{bmatrix} \end{aligned}$$

The LU factorization can be computed efficiently by a recursive algorithm. Then to solve $A\mathbf{x} = \mathbf{b}$, we note that

$$\begin{aligned} & A = PLU \\ & \mathbf{x} = A^{-1}\mathbf{b} = U^{-1}L^{-1}P^T\mathbf{b} \\ & \mathbf{z}_1 = P^T\mathbf{b}, \quad \mathbf{z}_2 = L^{-1}\mathbf{z}_1, \quad \mathbf{x} = U^{-1}\mathbf{z}_2 \end{aligned}$$

The last two equations are solved by forward substitution $L\mathbf{z}_2 = \mathbf{z}_1$ and by backward substitution $U\mathbf{x} = \mathbf{z}_2$.

Cramer's rule To Do. See wikipedia for now. It is computationally expensive but we will use it to derive a result later.

Solving $A\mathbf{x} = \mathbf{b}$ in practice and at the computer is done:

- via LU factorization (much quicker if one has to solve several systems with the same matrix A but different vectors b)
- if A is a symmetric positive definite matrix then by Cholesky decomposition (twice as fast)
- if A is large or sparse then by iterative methods

2.3 Simplex Method

Dantzig introduced the simplex algorithm to solve LP problems that can be written in scalar form as:

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} a_{ij} x_j \ \leq b_i, \ i = 1, \ldots, m \\ & x_j \ \geq 0, \ \ j = 1, \ldots, n \end{aligned}$$

or in matrix notation as:

$$\begin{aligned} \max\ & \mathbf{c}^T\mathbf{x} \\ & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

$$\mathbf{x} \in \mathbb{R}^n, \mathbf{c} \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^m$$

The algorithm first ensures that the problem is in a standard form. Then it determines an easy way to find initial solution. We will initially assume that this initial solution is feasible. Next the algorithm proceeds by iterating through feasible solutions that are vertices of the polyhedron that represents the feasibility region. Finally, it will use an optimality condition to terminate. A few exceptions may occur, they determine initial infeasibility, unboundedness, more than one solution and cycling in case of degenerancies. We will see how this situations are handled.

Standard Form The first step of the algorithm is to put the LP problem in a standard form:

Proposition 1 (Standard form). Each linear program can be converted in the form:

$$\begin{aligned} \max\ & \mathbf{c}^T\mathbf{x} \\ & A\mathbf{x} \leq \mathbf{b} \\ & x \in \mathbb{R}^n \end{aligned}$$

$$\mathbf{c} \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^m$$

Proof. If the problem is not in the standard form already then we can transform it:

- if there are equation constraints $\mathbf{ax} = b$, then we introduce two constraints, $\mathbf{ax} \leq b$ and $\mathbf{ax} \geq b$ for each of those constraints.
- if there are inequalities of the type $\mathbf{ax} \geq b$, then we change them to the form $-\mathbf{ax} \leq -b$.
- if the objective function is $\min \mathbf{c}^T\mathbf{x}$ then we change it to $\max(-\mathbf{c}^T\mathbf{x})$

□

For now, we assume that in the standard form $\mathbf{b} \geq 0$. As we will see, if this is true then finding an initial feasible solution is not trivial.

Proposition 2 (Equational standard form). Each LP problem can be converted to the form:

$$\begin{aligned} \max\ & \mathbf{c}^T\mathbf{x} \\ & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

$$\mathbf{x} \in \mathbb{R}^n, \mathbf{c} \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^m$$

that is, the objective is to maximize, the constraints are all equalities, the variables are all non-negative.

Proof. Every LP can be transformed in equational standard form:

1. we add one non-negative slack variable x_{n+i} to the left hand side of each constraint $i = 1, \ldots, m$ of the type $\leq$:

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} a_{ij} x_j + x_{n+i} = b_i, \ i = 1, \ldots, m \\ & x_j \geq 0, \ \ j = 1, \ldots, n \\ & x_{n+i} \geq 0, \ \ i = 1, \ldots, m \end{aligned}$$

   We assume here that the problem is already in standard form. If it was not then there might be larger or equal constraints, in which case we would subtract so-called non-negative surplus variables to make them equality constraints.

2. if some variable l, $l \in \{1, \ldots, n\}$ is free, $x_l \gtreqless 0$, then we introduce two new non-negative variables:

$$\begin{aligned} x_l &= x'_l - x''_l \\ x'_l &\geq 0 \\ x''_l &\geq 0 \end{aligned}$$

3. As above, $\min c^T x \equiv \max(-c^T x)$

4. Again we assume $\mathbf{b} \geq 0$.

□

Hence, every LP problem in $n \times m$ is converted to an LP problem with at most $(m+2n)$ variables and m equations (n is the number of original variables, m is the number of constraints).

The relevant form for the simplex algorithm is the equational standard form and it is this form that most text books refer to when referring to the standard form. We call the equational standard form determined by the procedure above *canonical* if the $\mathbf{b}$ terms are all non-negative. It is not always trivial to put the problem in canonical, equational, standard form and for infeasible problems it is simply not possible, as we will see.

From the geometrical point of view the feasibility region of the problem

$$\max\{\mathbf{c}^T \mathbf{x} \mid A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$$

is the intersection of the set of solutions of $A\mathbf{x} = \mathbf{b}$, which is an affine space (a plane not passing through the origin) and the nonegative orthant $\mathbf{x} \geq \mathbf{0}$. For a case in $\mathbb{R}^3$ with $A\mathbf{x} = \mathbf{b}$ made by $x_1 + x_2 + x_3 = 0$ the situation is shown in Figure 2.4 (in $\mathbb{R}^3$ the orthant is called octant).

Note that $A\mathbf{x} = \mathbf{b}$ is a system of equations that we can solve by Gaussian elimination. Elementary row operations of $\left[A \mid \mathbf{b}\right]$ such as:

- multiplying all entries in some row of $\left[A \mid \mathbf{b}\right]$ by a nonzero real number λ
- replacing the ith row of $\left[A \mid b\right]$ by the sum of the ith row and jth row for some $i \neq j$

do not affect set of feasible solutions. We assume $\text{rank}(\left[A \mid \mathbf{b}\right]) = \text{rank}(A) = m$, ie, rows of A are linearly independent. Otherwise, we remove the linear dependent rows and change the value of m.

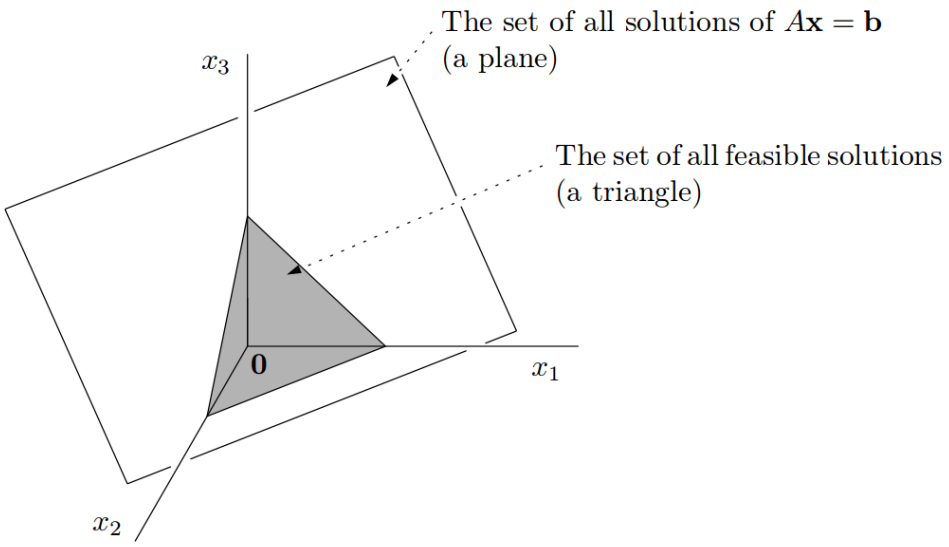


Figure 2.4: The intersection between $A\mathbf{x} = \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$ in $\mathbb{R}^3$. (Picture from Matoušek and Gärtner [2007]).

Basic Feasible Solutions Let $B = \{1 \dots m\}$ and $N = \{m+1 \dots n+m\}$ be a partition of the set of indices of the columns of the matrix A. We denote by A_B the matrix made of columns of A indexed by B.

Definition 1 (Basic Feasible Solutions). $\mathbf{x} \in \mathbb{R}^n$ is a **basic feasible solution** of the linear program $\max\{\mathbf{c}^T\mathbf{x} \mid A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0\}$ for an index set B if:

- $x_j = 0$ for all $j \notin B$.
- the square matrix A_B is non-singular, ie, all columns indexed by B are lin. indep.
- $\mathbf{x}_B = A_B^{-1}\mathbf{b}$ is non-negative, ie, $\mathbf{x}_B \geq 0$

In the definition, the last condition ensures the feasibility of the solution.

We call $x_j, j \in B$ **basic variables** and remaining variables **nonbasic variables**. Non basic variables are set to zero in the basic feasible solution determined by B.

Theorem 2.2 (Uniqueness of a basic feasible solution)**.** *A basic feasible solution is uniquely determined by the set B.*

Proof.

$$\begin{aligned} Ax =& A_B x_B + A_N x_N = b \\ & x_B + A_B^{-1} A_N x_N = A_B^{-1} b \\ & x_B = A_B^{-1} b \qquad\qquad A_B \text{ is non-singular hence one solution} \end{aligned}$$

□

Hence, we call B a **(feasible) basis**.

Extreme points of a polyhedron and basic feasible solutions are geometric and algebraic manifestations of the same concept (See Figure 2.5). Formally,

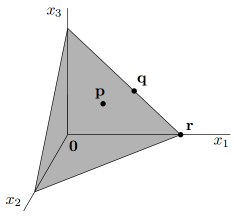


Figure 2.5: The points **p** and **q** represent feasible solutions but they are not extreme points. The point **r** is a feasible solution and an extreme point.

Theorem 2.3. *Let P be a (convex) polyhedron from LP in std. form. For a point $v \in P$ the following are equivalent:*

(i) v is an extreme point (vertex) of P

(ii) v is a basic feasible solution of LP

The proof, not shown here, goes through recognizing that vertices of P cannot be expressed as linear combinations of points in the polytope. Hence, vertices are linearly independent. Consequently, such are the columns of the matrix A_B. Conversely, since A_B is non-singular by definition then the solution to linear system is a single point, a vertex.

From the previous theorem and the fundamental theorem of linear programming, it follows that

Theorem 2.4. *Let $LP = \max\{\mathbf{c}^T\mathbf{x} \mid A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0\}$ be feasible and bounded, then the optimal solution is a basic feasible solution.*

We have thus learned how to find algebraically vertices of the polyhedron. The idea for a solution algorithm is therefore to examine all basic solutions. From what we saw, this corresponds to generating different sets B of the indices of the columns of A and checking whether the conditions for being a basic feasible solution hold. For a matrix A that after trasnformation to standard eqautional form has $n+m$ columns there are finitely many possible subsets B to examine, precisely

$$\binom{m+n}{m}.$$

If $n = m$, then $\binom{2m}{m} \approx 4^m$. Hence, even though at each iteration it might be easy to retrieve the value of the corresponding solutions, we are still left with exponentially many iterations to perform in the case that we have to see all vertices of the polyhedron, which is the worst case.

We are now ready to start working at a numerical example. Let's consider our previous problem from resource allocation. In scalar form:

$$\begin{array}{rrcrcl} \max & 6x_1 & + & 8x_2 & & \\ & 5x_1 & + & 10x_2 & \leq & 60 \\ & 4x_1 & + & 4x_2 & \leq & 40 \\ & & & x_1, x_2 & \geq & 0 \end{array}$$

and in matrix form:

$$\max \quad \begin{bmatrix} 6 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\begin{bmatrix} 5 & 10 \\ 4 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 60 \\ 40 \end{bmatrix}$$

$$x_1, x_2 \geq 0$$

We put the problem in canonical equational standard form:

$$\begin{aligned} 5x_1 + 10x_2 + x_3 &= 60 \\ 4x_1 + 4x_2 + x_4 &= 40 \end{aligned}$$

or, equivalently, in matrix form:

$$\max \quad z = \begin{bmatrix} 6 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\begin{bmatrix} 5 & 10 & 1 & 0 \\ 4 & 4 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 60 \\ 40 \end{bmatrix}$$

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

If the equational standard form is canonical one decision variable is isolated in each constraint and it does not appear in the other constraints nor in the objective function and the b terms are positive.

The advantage of the canonical form is evident: it gives immediately a basic feasible solution:

$$x_1 = 0,\ x_2 = 0,\ x_3 = 60,\ x_4 = 40$$

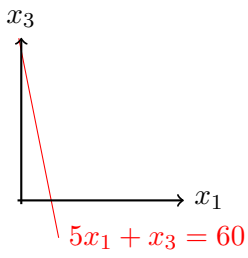
The basis of this solution is $B = \{3, 4\}$. Consequently, $N = \{1, 2\}$. If this solution is also optimal then the algorithm can terminate and return the solution. Is this solution optimal?

Looking at signs in z it seems not: since they are positive, if we can increase the variables x_1 and x_2 to become larger than zero then the solution quality would improve. Let's then try to increase a promising variable, i.e., one with positive coefficient in z. Let's take x_1 and let's consider how much we can increase its value looking at the first constraint. Since x_2 stays equal to zero, this variable does not appear in the constraint.

$$5x_1 + x_3 = 60$$

Isolating first x_1 and then x_3 we can plot the line represented by this constraint:

$$\begin{aligned} x_1 &= \tfrac{60}{5} - \tfrac{x_3}{5} \\ x_3 &= 60 - 5x_1 \geq 0 \end{aligned}$$

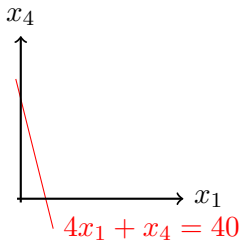


From the explicit form we see that if we increase x_1 more than 12 then x_3 becomes negative and thus the whole solution infeasible. This constraint imposes therefore an upper bound of 12 to the increase of x_1. Let's analyze the second constraint now:

$$4x_1 + x_4 = 40$$

Isolating x_1 and x_4 and plotting the line we obtain:

$$x_1 = \tfrac{40}{4} - \tfrac{x_4}{4}$$
$$x_4 = 40 - 4x_1 \geq 0$$



For a similar reasoning as above we observe that this constraint imposes an upper bound of 10 to the increase of x_1.

It follows that the value of x_1 can be increased at most up to 10. Increasing x_1 to 10 makes $x_4 = 0$ because of the second constraint. Hence, we want that x_4 exits the basis while x_1 enters in it. In order to bring the problem back in canonical standard form after the increase of x_1 we need to perform elementary row operations. To this end it is convenient to work with a particular organization of the data that is called *simplex tableau* (plural tableaux).

	x_1	x_2	x_3	x_4	$-z$	b
x_3	5	10	1	0	0	60
x_4	4	4	0	1	0	40
	6	8	0	0	1	0

The variables that label the columns remain fixed throughout the iterations, while the labels in the rows changes depending on which variables are in basis. The column $-z$ will never change throughout the iterations of the algorithm. The last row is given by the objective function. Note that some text books put this row as the first row on the top. With the new basis the new tableau that correspond to a canonical standard form looks like this:

	x_1	x_2	x_3	x_4	$-z$	b
x_3	0	?	1	?	0	?
x_1	1	?	0	?	0	?
	0	?	0	?	1	?

that is, there is a permuted identity matrix whose last column, $-z$, remains fixed while the other two columns indicate which variable is in basis.

The decisions that we have done so far: to select a variable to increase, the amount of the increase, which variable has to decrease and putting the tableau the new form, can be written in general terms as the following *pivot operations*.

Definition 2 (Pivot operations). The following operations are done at each iteration of the simplex:

1. Determine a pivot:

   column: a column s with positive coefficients in the objective function. Why? [The coefficients determine how the value of the objective function will change after an increase of x_1 or x_2. Both x_1 and x_2 have a positive coefficient in the example, hence of of them can be taken arbitrarily (later we will argue on how to decide for one of them in a more accurate way)]

   row: consider the ratio between the coefficients $\mathbf{b}$ and the coefficients $\mathbf{a}_s$ of the pivot column and choose the one with smallest ratio:

$$r = \text{argmin}_i \left\{ \frac{b_i}{a_{is}} : a_{is} > 0 \right\}$$

2. elementary row operations to update the tableau around the pivot.

Note that the choice of the row of the pivot gives us also the increase value θ of entering variable, that is,

$$\theta = \min_i \left\{ \frac{b_i}{a_{is}} : a_{is} > 0 \right\}$$

Let's get back to our numerical example and perform the simplex iterations:

- x_1 enters the basis and x_4 leaves the basis. The pivot is element at the position of the row and column selected, ie, the coefficient 4. The elementary row operations to put the tableau in the new form are:

  - Divide the pivot row by the value of the pivot
  - Send to zero the coefficient in the pivot column of the first row
  - Send to zero the coefficient of the pivot column in the third (cost) row

```
|               | x1 | x2 | x3 | x4   | -z |   b |
|---------------+----+----+----+------+----+-----|
| I'=I-5II'     |  0 |  5 |  1 | -5/4 |  0 |  10 |
| II'=II/4      |  1 |  1 |  0 | 1/4  |  0 |  10 |
|---------------+----+----+----+------+----+-----|
| III'=III-6II' |  0 |  2 |  0 | -6/4 |  1 | -60 |
```

  From the last row we read: $2x_2 - 3/2x_4 - z = -60$, that is: $z = 60 + 2x_2 - 3/2x_4$. Since x_2 and x_4 are nonbasic we have $z = 60$ and $x_1 = 10, x_2 = 0, x_3 = 10, x_4 = 0$.

- Are we done? No, there are still positive coefficients in the objective row! Let x_2 enter the basis. We determine the pivot, which is 5, hence x_3 is the variable that exists. After the row operations we obtain:

```
|              | x1 | x2 | x3   | x4   | -z |   b |
|--------------+----+----+------+------+----+-----|
| I'=I/5       |  0 |  1 | 1/5  | -1/4 |  0 |   2 |
| II'=II-I'    |  1 |  0 | -1/5 | 1/2  |  0 |   8 |
|--------------+----+----+------+------+----+-----|
| III'=III-2I' |  0 |  0 | -2/5 | -1   |  1 | -64 |
```

- Are we done? Yes! The variables not in basis have negative coefficients in the objective function that corresponds to the tableau we reached. Hence if we increased them, we would worsen the objective function. The solutions we have found is therefore the optimal one.

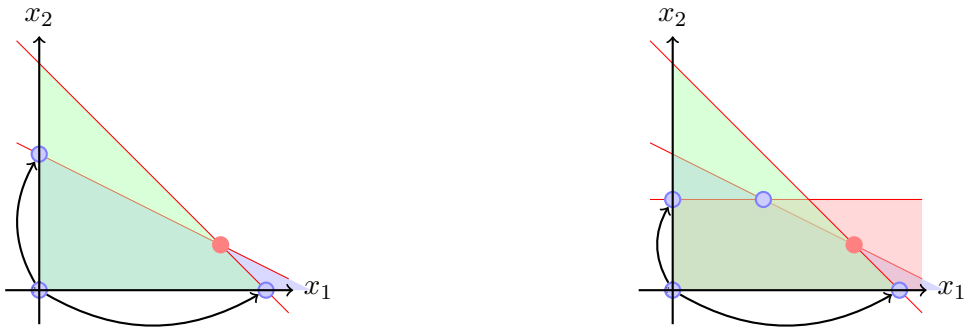


Figure 2.6: The search process of the simplex.

Definition 3 (Reduced costs). The coefficients in the objective function of the nonbasic variables, $\bar{\mathbf{c}}_N$, are called reduced costs.

Note that basic variables have always coefficients in the last row equal to zero.

Definition 4 (Improving variables). An **improving variable** is a non basic variable with positive reduced cost.

Proposition 3 (Optimality condition). A basic feasible solution is **optimal** when the **reduced costs** in the corresponding simplex tableau are **nonpositive**, ie:

$$\bar{\mathbf{c}}_N \leq 0$$

In Figure 2.6 left, we represent graphically the solution process executed by the simplex algorithm. Starting from the vertex $(0, 0)$, we moved to $(10, 0)$ and finally reached the optimal solution in $(8, 2)$. For this problem the other path with x_2 increased before x_1 would have been of the same length and hence lead to the same number of iterations of the simplex algorithm. However, the situation is not always like this. In Figure 2.6 right, and in Figure 2.7, we see that choosing one direction of increase rather than another may influence considerably the efficiency of the search.

We said earlier that trying all points implies approximately 4^m iterations. This is an asymptotic upper bound. On the other hand to find an asymptotic lower bound we should apply the clairvoyant's rule, that is, using the shortest possible sequence of steps for any pair of vertices we may choose as starting and optimal solutions. However, the length of this path for a general polyhedron in $\mathbb{R}^n$ is not known. Hirsh conjectures $O(n)$ but the best known result is $n^{1+\ln n}$.

In practice, the simplex algorithm runs in between $2m$ and $3m$ iterations. (Hence, relevant to note, the number of iterations depends on the number of constraints.)

Tableaux and Dictionaries We chose to use the tableau representation which is the original Dantzig representation of the simplex algorithm. An alternative representation by means of dictionaries due to Chvatal is equally spread and used in text books. The tableau representation is more amenable to implementations at the computer than the dictionary one. However, efficient code use the revised simplex method and hence not either a tableaux representation.

Let's consider the general LP problem:

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \ i = 1, \ldots, m \\ & x_j \geq 0, \ \ j = 1, \ldots, n \end{aligned}$$

The equational standard form can be written, perhaps more intuitively, also by isolating the slack variables:

$$\begin{aligned} \max \quad z &= \sum_{j=1}^{n} c_j x_j \\ x_{n+i} &= b_i - \sum_{j=1}^{n} a_{ij} x_j, \ i = 1, \ldots, m \\ x_j &\geq 0, \qquad\qquad j = 1, \ldots, n \\ x_{n+i} &\geq 0, \qquad\qquad i = 1, \ldots, m \end{aligned}$$

This form gives immediately the dictionary representation. We compare this representation side by side with the tableau representation:

Tableau

$$\left[\begin{array}{c|c|c|c} I & \bar{A}_N & 0 & \bar{b} \\ \hline 0 & \bar{c}_N & 1 & -\bar{d} \end{array}\right]$$

Dictionary

$$\begin{aligned} x_r &= \bar{b}_r - \sum_{s \notin B} \bar{a}_{rs} x_s, \quad r \in B \\ z &= \bar{d} + \sum_{s \notin B} \bar{c}_s x_s \end{aligned}$$

The last row of the dictionary gives us the same objective function as we have seen that it can be derived from the last row of tableau, namely:

$$\sum_{r \in B} 0 x_r + \sum_{s \notin B} \bar{c}_N x_N - z = -\bar{d}.$$

Decisions in the two cases must correspond. In the dictionary the Pivot operations are given by:

1. Determine a pivot:

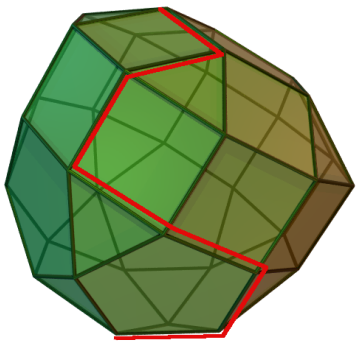


Figure 2.7: The search process in a generic polyhedron in $\mathbb{R}^3$.

column choose the column s with reduced cost strictly positive

row choose the row i with negative coefficients such that the ratio b/a_{si} is minimal

2. update: express the entering variable and substitute it in the other rows

Example 2.1.

$$\begin{aligned} \max\ & 6x_1 + 8x_2 \\ & 5x_1 + 10x_2 \leq 60 \\ & 4x_1 + 4x_2 \leq 40 \\ & x_1, x_2 \geq 0 \end{aligned}$$

	x_1	x_2	x_3	x_4	$-z$	b
x_3	5	10	1	0	0	60
x_4	4	4	0	1	0	40
	6	8	0	0	1	0

$$\begin{aligned} x_3 &= 60 - 5x_1 - 10x_2 \\ x_4 &= 40 - 4x_1 - 4x_2 \\ z &= + 6x_1 + 8x_2 \end{aligned}$$

After two iterations:

	x_1	x_2	x_3	x_4	$-z$	b
x_2	0	1	1/5	−1/4	0	2
x_1	1	0	−1/5	1/2	0	8
	0	0	−2/5	−1	1	−64

$$\begin{aligned} x_1 &= 2 - 1/5x_3 + 1/4x_4 \\ x_2 &= 8 + 1/5x_3 - 1/2x_4 \\ z &= 64 - 2/5x_3 - 1x_4 \end{aligned}$$

2.4 Exception Handling

So far we have seen the simplex only on a problem that has a unique solution. An LP problem with a feasibility region F can have the following outcomes:

1. $F \neq \emptyset$ and $\nexists$ solution
2. $F \neq \emptyset$ and $\exists$ solution
   (a) **one solution**
   (b) infinite solution
3. $F = \emptyset$

Therefore, we will now look at the behaviour of the simplex in conditions of:

1. Unboundedness
2. More than one solution
3. Infeasibility

Under the last point we will see that the assumption of a starting basic feasible solution does not always hold and that this needs some particular handling.

Moreover, the simplex algorithm can incurr in degeneracies that may be benign if they resolve by themselves or malign if the final result is cycling.

2.4.1 Unboundedness

We consider the following LP problem instance:

$$\begin{array}{rrcl} \max & 2x_1 & + & x_2 \\ & & & x_2 \leq 5 \\ & -x_1 & + & x_2 \leq 1 \\ & & & x_1, x_2 \geq 0 \end{array}$$

- We write the initial tableau

```
|    | x1 | x2 | x3 | x4 | -z | b |
|----+----+----+----+----+----+---|
| x3 |  0 |  1 |  1 |  0 |  0 | 5 |
| x4 | -1 |  1 |  0 |  1 |  0 | 1 |
|----+----+----+----+----+----+---|
|    |  2 |  1 |  0 |  0 |  1 | 0 |
```

  and perform the pivot operations:

- x_2 entering, x_4 leaving

```
|             | x1 | x2 | x3 | x4 | -z |  b |
|-------------+----+----+----+----+----+----|
| II'=II-I'   |  1 |  0 |  1 | -1 |  0 |  4 |
| I'=I        | -1 |  1 |  0 |  1 |  0 |  1 |
|-------------+----+----+----+----+----+----|
| III'=III-I' |  3 |  0 |  0 | -1 |  1 | -1 |
```

  The second row corresponds to the constraint:

$$-x_1 + x_2 + x_4 = 1$$

  , where x_2 being in the basis is set to zero. Hence, x_1 can increase without restriction. This is why when writing the maximum allowed increase, we enforced $a_{is} > 0$: $\theta = \min\{\frac{b_i}{a_{is}} : a_{is} > 0, i = 1 \ldots, n\}$. If $a_{is} \leq 0$ then the variable can increase arbitrarily.

- x_1 entering, x_3 leaving

```
|              | x1 | x2 | x3 | x4 | -z |   b |
|--------------+----+----+----+----+----+-----|
| I'=I         |  1 |  0 |  1 | -1 |  0 |   4 |
| II'=II+I'    |  0 |  1 |  1 |  0 |  0 |   5 |
|--------------+----+----+----+----+----+-----|
| III'=III-3I' |  0 |  0 | -3 |  2 |  1 | -13 |
```

  x_4 was already in basis but for both I and II

$$\begin{array}{rcrcl} x_1 & + & x_3 & - & x_4 = 4 \\ x_2 & + & x_3 & + & 0x_4 = 5 \end{array}$$

  x_4 can now be increased arbitrarily: in the first constraint it will be compensated by x_1 (x_3 is non basic and hence 0) and in the second constraint it doesn't appear at all.

We are therefore in the condition of an unbounded problem. We recognise this when a variable chosen to enter in the basis is not upper bounded in its increase. Figure 2.8 provides the geometrical view of the solution process for this example.

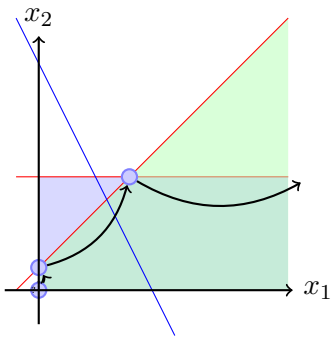


Figure 2.8: The solution process of an unbounded problem.

2.4.2 Infinite solutions

Consider the following LP problem instance:

$$\begin{array}{lrcrcl} \max & x_1 & + & x_2 & & \\ & 5x_1 & + & 10x_2 & \leq & 60 \\ & 4x_1 & + & 4x_2 & \leq & 40 \\ & & & x_1, x_2 & \geq & 0 \end{array}$$

- The initial tableau is

```
|    | x1 | x2 | x3 | x4 | -z |  b |
|----+----+----+----+----+----+----|
| x3 |  5 | 10 |  1 |  0 |  0 | 60 |
| x4 |  4 |  4 |  0 |  1 |  0 | 40 |
|----+----+----+----+----+----+----|
|    |  1 |  1 |  0 |  0 |  1 |  0 |
```

  we proceed to the pivot operations:

- x_2 enters, x_3 leaves

```
|             |  x1 | x2 | x3   | x4 | -z |  b |
|-------------+-----+----+------+----+----+----|
| I'=I/10     | 1/2 |  1 | 1/10 |  0 |  0 |  6 |
| II'=II-4I'  |   2 |  0 | -2/5 |  1 |  0 | 16 |
|-------------+-----+----+------+----+----+----|
| III'=III-I  | 1/2 |  0 | -1/6 |  0 |  1 | -6 |
```

- x_1 enters, x_4 leaves

```
|               | x1 | x2 | x3   | x4   | -z |   b |
|---------------+----+----+------+------+----+-----|
| I'=I-II'/2    |  0 |  1 | 1/5  | -1/4 |  0 |   2 |
| II'=II/2      |  1 |  0 | -1/5 | 1/2  |  0 |   8 |
|---------------+----+----+------+------+----+-----|
| III'=III-II'/2 |  0 |  0 | 0    | -1/4 |  1 | -10 |
```

  The corresponding solution is $\mathbf{x}_1 = (8, 2, 0, 0), z = 10$. Applying the optimality condition we see that the solution is optimal. However, we are used to see that nonbasic variables have reduced costs not equal to 0. Here x_3 has reduced cost equal to 0. Let's make it enter the basis.

- x_3 enters, x_2 leaves

	x1	x2	x3	x4	-z	b
I'=5I	0	5	1	-5/4	0	10
II'=II+I'/5	1	1	0	4	0	10
III'=III	0	0	0	-1/4	1	-10

  We find a different solution that has the same value: $\mathbf{x}_2 = (10, 0, 10, 0), z = 10$. Note that we use a subscript to differentiate from the first soltution.

  Hence we found two optimal solutions. If we continued from here we would again bring x_2 in the basis and x_3 out, thus cycling.

If more than one solution is optimal, then we saw that also all their convex combinations are optimal solutions. Let's then express all optimal solutions. The convex combination is:

$$\begin{aligned} \mathbf{x} &= \sum_{i=1}^{2} \alpha_i \mathbf{x}_i \\ \alpha_i &\geq 0 \qquad \forall i = 1, 2 \\ \sum_{i=1}^{2} \alpha_i &= 1 \end{aligned}$$

In our case we have:

$$\begin{aligned} \mathbf{x}_1^T &= [8, 2, 0, 0] \\ \mathbf{x}_2^T &= [10, 0, 10, 0] \end{aligned}$$

Any vector $\mathbf{x}$ resulting from the convex combination with coefficients $\alpha_1 = \alpha$ and $\alpha_2 = 1 - \alpha$ is given by:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \alpha \begin{bmatrix} 8 \\ 2 \\ 0 \\ 0 \end{bmatrix} + (1 - \alpha) \begin{bmatrix} 10 \\ 0 \\ 10 \\ 0 \end{bmatrix}$$

or

$$\begin{aligned} x_1 &= 8\alpha + 10(1 - \alpha) \\ x_2 &= 2\alpha \\ x_3 &= 10(1 - \alpha) \\ x_4 &= 0. \end{aligned}$$

A problem has infinite solutions when the objective hyperplane is parallel to one of the faces of the feasibility region with dimension larger than 0. The example is depicted in Figure 2.9. A face could have larger dimensions and the simplex would find all its extreme vertices before looping between them.

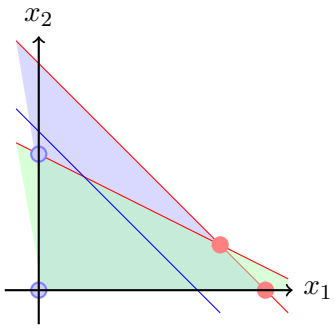


Figure 2.9: The example with infinite solutions. The objective function is parallel with the edge of the feasibility region. The solutions found by the simplex are the two extremes of the segment.

2.4.3 Degeneracy

Let this time the LP problem instance be:

$$\begin{array}{lrcl} \max & x_2 & & \\ & -x_1 + x_2 & \leq & 0 \\ & x_1 & \leq & 2 \\ & x_1, x_2 & \geq & 0 \end{array}$$

- The initial tableau is

```
|    | x1 | x2 | x3 | x4 | -z | b |
|----+----+----+----+----+----+---|
| x3 | -1 |  1 |  1 |  0 |  0 | 0 |
| x4 |  1 |  0 |  0 |  1 |  0 | 2 |
|----+----+----+----+----+----+---|
|    |  0 |  1 |  0 |  0 |  1 | 0 |
```

  The novel element here is that a right-hand side coefficient is zero, ie, $b_1 = 0$. In the pivot operations, a null b term will make such that the entering variable will not be increased. Indeed, a null b term will make the increase value θ null.

  Definition 5 (Degenerate pivot step). We call *degenerate pivot step* a pivot step in which the entering variable stays at zero.

- Let's proceed and make x_2 enter the basis and x_3 leave it.

```
|   | x1 | x2 | x3 | x4 | -z | b |
|---+----+----+----+----+----+---|
|   | -1 |  1 |  1 |  0 |  0 | 0 |
|   |  1 |  0 |  0 |  1 |  0 | 2 |
|---+----+----+----+----+----+---|
|   |  1 |  0 | -1 |  0 |  1 | 0 |
```

- in the next step we end up avoiding the constraint with the null b term and the step is not degenerate anymore. We exit from the degeneracy state and reach an optimal tableau:

```
|   | x1 | x2 | x3 | x4 | -z |  b |
|---+----+----+----+----+----+----|
|   |  0 |  1 |  0 |  1 |  0 |  2 |
|   |  1 |  0 |  0 |  1 |  0 |  2 |
|---+----+----+----+----+----+----|
|   |  0 |  0 | -1 | -1 |  1 | -2 |
```

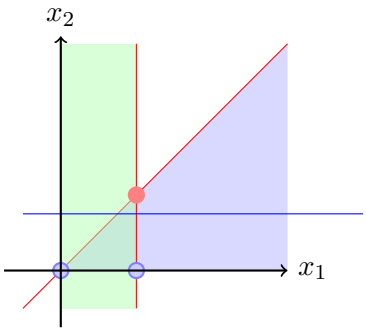


Figure 2.10: In the origin, three constraints meet. In that vertex the simplex method encounters a degeneracy, which in this case is resolved and another vertex reached.

where the solution is $x_1 = 2, x_2 = 2$ and the objective value z is 2.

The situation is represented graphically in Figure **??**. If n is the number of original variables, degeneracies arise when $n + 1$ or more constraints meet at a vertex. In other terms, there are polyhedra that have vertices that are overdetermined, that is, the number of facets that meet in those vertices is larger than $\dim(P)$. In this case, every $\dim(P)$ inequalities that define these facets determine a basis that produce a basis solution. In linear algebra terms, for $n + m$ variables of an LP problem in equational standard form, a basis solution that belongs to a basis B has n variables set to zero and the remaining m variables set to $A_B^{-1}b$. In a degenerate basis solution there are more than n variables set to zero. It follows that the same solution $\mathbf{x}$ is solution of more than one regular $n \times n$ subsystem.

Degeneracy may lead to cycling in the simplex.

Theorem 2.5. *If the simplex fails to terminate, then it must cycle.*

Proof. A tableau is completely determined by specifying which variables are basic and which are nonbasic. There are only

$$\binom{n+m}{m}$$

different possibilities, where n is the number of original variables and m is the number of constraints. The simplex method always moves to non-worsening tableaux. If the simplex method fails to terminate, it must visit some of these tableaux more than once. Hence, the algorithm cycles. □

Degenerate conditions may appear often in practice but cycling is rare and some pivoting rules prevent cycling. So far we chose an **arbitrary improving variable** to enter the basis.

For the following pivoting rule:

i the entering variable will always be the nonbasic variable that has the largest coefficient in the z-row of the dictionary.

ii if two or more basic variables compete for leaving the basis, then the candidate with the smallest subscript will be made to leave.

the following problem is the smallest possible example of cycling:

$$\begin{array}{ll} \text{maximize} & 10x_1 - 57x_2 - 9x_3 - 24x_4 \\ \text{subject to} & x_1 \leq 1 \\ & -0.5x_1 + 5.5x_2 + 2.5x_3 - 9x_4 \leq 0 \\ & -0.5x_1 + 1.5x_2 + 0.5x_3 - x_4 \leq 0 \\ & x_1, x_2, x_3, x_4 \geq 0. \end{array}$$

2.4.4 Pivot Rules

Pivot rules are rules for breaking ties in selecting improving variables to **enter** in the basis. Ties may occur also in selecting the leaving variables and the rules can regulate also how to break those ties. However, the choice of the entering variables is more important than the choice of the leaving variables.

- **Largest Coefficient**: select the improving variable with largest coefficient in last row of the tableau, ie, reduced cost. This is the original Dantzig's rule, and it was shown that it can lead to cycling.
- **Largest increase**: select the improving variable that leads to the best absolute improvement, ie, $\text{argmax}_j\{c_j\theta_j\}$. This rule is computationally more costly.
- **Steepest edge**: select the improving variable that if brought in the basis, would move the current basic feasible solution in a direction closest to the direction of the vector $\mathbf{c}$ (ie, maximizes the cosine of the angle between the two vectors):

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta \quad \Longrightarrow \quad \max \frac{\mathbf{c}^T(\mathbf{x}_{\text{new}} - \mathbf{x}_{\text{old}})}{\| \mathbf{x}_{\text{new}} - \mathbf{x}_{\text{old}} \|}$$

- **Bland's rule**: choose the improving variable with the lowest index and, if there are more than one leaving variable, the one with the lowest index. This rule prevents cycling but it is slow.
- **Random edge**: select an improving variable uniformly at random.
- **Perturbation method**: perturb the values of b_i terms to avoid $b_i = 0$, which must occur for cycling. To avoid cancellations: $0 < \epsilon_m \ll \epsilon_{m-1} \ll \cdots \ll \epsilon_1 \ll 1$. It can be shown to be the same as lexicographic method, which prevents cycling

2.4.5 Efficiency of simplex method

- The asymptotic upper bound is given by trying all basis, which is $\approx 4^m$. Klee and Minty 1978 constructed an example that requires the simplex with Dantzig's pivot rule to visit all $2^n - 1$ vertices of a polyhedron and hence the maximum number of iterations. See Figure 2.11.
- It is unknown if there exists a pivot rule that leads to polynomial time. The best would be the Clairvoyant's rule: that is, choose the pivot rule that gives the shortest possible sequence of steps. This corresponds to determining the diameter of the m dimensional polytope. The diameter of a polytope P is the maximum distance between any two vertices in the edge graph of P (Figure 2.12). The diameter gives a lower bound for any pivoting rule for the simplex algorithm. Hirsch conjectured (1957) that the diameter of any n-facet convex polytope in

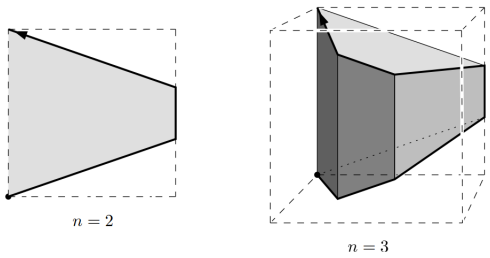


Figure 2.11: Klee and Minty's example

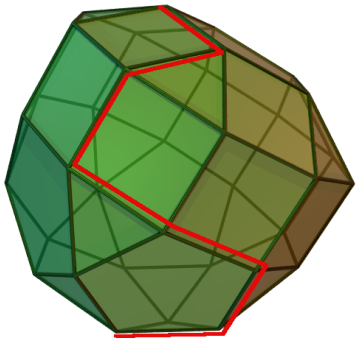


Figure 2.12: . The shortest path between any two vertices of a polyhedron may contain an exponentially growing number of vertices as the dimension grows, ie, $O(n^{\log n})$. It is however conjectured that the growth is linear.

d-dimensional Euclidean space is at most $n-d$. Kalai and Kleitman (1992) gave an $O(n^{\log n})$ upper bound on the diameter namely $n^{1+\ln n}$. Hirsh conjecture was disproved in May 2010 by Francisco Santos from the University of Cantabria in Santander. He constructed a 43-dimensional polytope with 86 facets and diameter bigger than 43. [Documenta Math. 75 Who Solved the Hirsch Conjecture? Günter M. Ziegler]. In general terms he showed the existence of polytopes with diameter $(1+\epsilon)(n-d)$. It remains open whether the diameter is polynomial, or even linear, in n and d.

- In practice the simplex runs in between $2m$ and $3m$ number of iterations (m is the number of constraints), hence the running time seems to be dependent on the number of constraints.

- Positive results are of smoothed complexity type: that is, average case on slight random perturbations of worst-case inputs. D. Spielman and S. Teng (2001), *Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time*
$O(max(n^5 \log^2 m, n^9 log^4 n, n^3 \sigma^{-4}))$

- One of the most prominent mysteries in Optimization remains the question of whether a linear program can be solved in strongly-polynomial time. A strongly polynomial-time method would be polynomial in the dimension n and in the number m of inequalities only, whereas the complexity of the known weakly-polynomial time algorithms for linear programming, like the ellipsoid method or variants of the interior-point method, also depend on the binary encoding length of the input. The simplex method, though one of the oldest methods for linear

programming, still is a candidate for such a strongly polynomial time algorithm. This would require the existence of a pivoting rule that results in a polynomial number of pivot steps. Since the famous Klee-Minty example, many techniques for deriving exponential lower bounds on the number of iterations for particular pivoting rules have been found.

Some very important pivoting rules, however, have resisted a super-polynomial lower-bound proof for a very long time. Among them the pivoting rules Random Edge (uniformly random improving pivots) Randomized Bland's Rule (random shuffle the indexes + lowest index for entering + lexicographic for leaving) Random-Facet and Zadeh's pivoting rule (least-entered rule: enter the improving variable that has been entered least often – it minimizes revisits). Random-Facet has been shown to yield sub-exponential running time of the simplex method independently by Kalai as well as by Matousek, Sharir and Welzl. For every linear program with at most n variables and at most m constraints, the expected number of pivot steps is bounded by $e^{C\sqrt{m \ln n}}$, where C is a (not too large) constant. (Here the expectation means the arithmetic average over all possible orderings of the variables.) O. Friedmann together with Hansen and Zwick have shown super-polynomial (but subexponential) lower bounds for Random Edge, Randomized Bland's rule and Zadeh's pivoting rules in 2011. The same authors in 2015 proposed an improved version of the Random-Facet rule that achieves the best known sub-exponential running time. These results are unrelated to the diameter of the polytope. (Sources: Mathematical Optimization Society, 2012 Tucker Prize Citation; Thomas Dueholm Hansen, Aarhus University, http://cs.au.dk/~tdh/;

2.5 Infeasibility and initialization

So far we have assumed that the **b** terms in the standard form were positive. If this is not the case, it might be not trivial to obtain a canonical equational standard form. Let's consider the following example:

$$\begin{array}{lrcrcl} \max & x_1 & - & x_2 & & \\ & x_1 & + & x_2 & \leq & 2 \\ & 2x_1 & + & 2x_2 & \geq & 5 \\ & & & x_1, x_2 & \geq & 0 \end{array}$$

The second constraint is of larger and equal type. To make it smaller and equal we multiply left-hand side and right-hand side by -1, yielding a negative right-hand side term. The equational standard form becomes:

$$\begin{array}{lrcrcrcrcl} \max & x_1 & - & x_2 & & & & & & \\ & x_1 & + & x_2 & + & x_3 & & & = & 2 \\ & -2x_1 & + & -2x_2 & & & + & x_4 & = & -5 \\ & & & & & x_1, x_2, x_3, x_4 & \geq & 0 & & \end{array}$$

We can now make the **b** terms all positive:

$$\begin{array}{lrcrcrcrcl} \max & x_1 & - & x_2 & & & & & & \\ & x_1 & + & x_2 & + & x_3 & & & = & 2 \\ & 2x_1 & + & 2x_2 & & & - & x_4 & = & 5 \\ & & & & & x_1, x_2, x_3, x_4 & \geq & 0 & & \end{array}$$

However, when we write the corresponding tableau we observe that it is not in canonical form, that is, we cannot recognize an identity submatrix.

	x1	x2	x3	x4	-z	b
x3	1	1	1	0	0	2
x4	2	2	0	-1	0	5
	1	-1	0	0	1	0

We note that, similarly to the canonical form, one decision variable is isolated in each constraint and it does not appear in the other constraints nor in the objective function but for x_4 the coefficient is -1. If we take x_3 and x_4 in basis then reading from the tableau, their value is $x_3 = 2$ and $x_4 = -5$. This does not comply with the definiton of basic feasible solution that asks all variables to be non-negative. Hence, we do not have an initial basic feasible solution!

In general finding any feasible solution is as difficult as finding an optimal solution, otherwise we could do binary search on the values of the objective function (that is, solving a sequence of systems of linear inequalities, one of which being the constrained objective function).

To find an initial feasible solution we formulate an auxiliary problem and solve it.

Auxiliary Problem (Phase I of the Simplex) We construct an auxiliry problm by introducing one non-negative auxiliary variable for each constraint in which there is not an isolated variable with coefficient $+1$. This will solve the problem of having an initial feasible basis but the problem would be different. If however we find a solution in which the new variables introduced are set to zero then this solution will be valid also for the original problem. The goal of the auxiliary problem is therefore to find a solution where the auxiliary variables are zero, or in other terms, to minimize the sum of the auxiliary variables. To remain consistent with our maximization form, we can rewrite the problem as a maximization problem.

In our example above we introduce the auxiliary non-negative variable x_5 in the second constraint and minimize its value:

$$\begin{aligned} w^* = \min x_5 = \max -x_5 & \\ x_1 + x_2 + x_3 &= 2 \\ 2x_1 + 2x_2 - x_4 + x_5 &= 5 \\ x_1, x_2, x_3, x_4, x_5 &\geq 0 \end{aligned}$$

If $w^* = 0$ then $x_5 = 0$ and the two problems are equivalent, if $w^* > 0$ then it is not possible to set x_5 to zero and the original problem does not have a feasible solution.

Let's solve this auxiliary problem.

- In the initial tableau we introduce a new row at the bottom for the new objective function to maximize and a new column denoted by $-w$. We keep the row for the original objective function and the column $-z$. In the pivot operations we will keep $-z$ and $-w$ always in basis.

	x1	x2	x3	x4	x5	-z	-w	b
	1	1	1	0	0	0	0	2
	2	2	0	-1	1	0	0	5
z	1	-1	0	0	0	1	0	0
w	0	0	0	0	-1	0	1	0

- The initial tableau is not yet in canonical form but it can very easily be made such by letting x_5 enter the basis and x_4 leave:

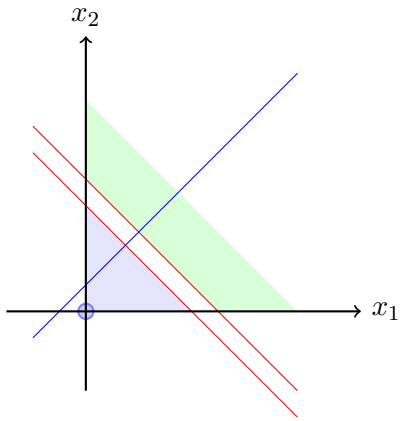


Figure 2.13: The feasibility region is the intersection of the half-spaces described by the constraints. In this case it is empty.

	x1	x2	x3	x4	x5	-z	-w	b
	1	1	1	0	0	0	0	2
	2	2	0	-1	1	0	0	5
z	1	-1	0	0	0	1	0	0
IV+II	2	2	0	-1	0	0	1	5

Now we have a basic feasible solution. It is $[0, 0, 2, 0, 5]$ and its objective value is $w = -5$. It is not optimal and therefore we proceed to find the optimal solution.

- x_1 enters the basis and x_3 leaves it:

	x1	x2	x3	x4	x5	-z	-w	b
	1	1	1	0	0	0	0	2
II-2I'	0	0	-2	-1	1	0	0	1
III-I'	0	-2	-1	0	0	1	0	-2
IV-2I'	0	0	-2	-1	0	0	1	1

The tableau is optimal. The optimal value can be read from the last row of the tableau: $w^* = -1$. Hence, we see that $x_5 = 1$ and it cannot be decreased further. Then no solution with $x_5 = 0$ exists and there is no feasible solution for our initial problem.

The original problem is infeasible. We can appreciate this also graphically from Figure 2.13 where we see that the intersection between the half-spaces that define the problem is empty.

Let's change the right-hand side of the second constraint in the previous example to be 2 instead of 5.

$$\begin{array}{rrcrcl} \max & x_1 & - & x_2 & & \\ & x_1 & + & x_2 & \leq & 2 \\ & 2x_1 & + & 2x_2 & \geq & 2 \\ & & & x_1, x_2 & \geq & 0 \end{array}$$

The equational standard form becomes:

$$\begin{array}{rrrrrrrrl} \max & x_1 & - & x_2 & & & & & \\ & x_1 & + & x_2 & + & x_3 & & = & 2 \\ & 2x_1 & + & 2x_2 & & & - & x_4 & = & 2 \\ & & & & & x_1, x_2, x_3, x_4 & \geq & 0 & \end{array}$$

Since it is not canonical we resort to the Phase I of the simplex by formulating the auxiliary problem:

$$\begin{array}{rrrrrrrrrl} w = & \min x_5 = \max -x_5 & & & & & & & & \\ & x_1 & + & x_2 & + & x_3 & & & = & 2 \\ & 2x_1 & + & 2x_2 & & & - & x_4 + x_5 & = & 2 \\ & & & & & x_1, x_2, x_3, x_4, x_5 & & & \geq & 0 \end{array}$$

- The initial tableau for the auxiliary problem is:

	x1	x2	x3	x4	x5	-z	-w	b
	1	1	1	0	0	0	0	2
	2	2	0	-1	1	0	0	2
z	1	-1	0	0	0	1	0	0
w	0	0	0	0	-1	0	1	0

  We do not have yet an initial basic feasible solution.

- we can set the problem in canonical form by making x_5 entering the basis and x_4 leaving:

	x1	x2	x3	x4	x5	-z	-w	b
	1	1	1	0	0	0	0	2
	2	2	0	-1	1	0	0	2
z	1	-1	0	0	0	1	0	0
IV+II	2	2	0	-1	0	0	1	2

  The basic feasible solution is $[0, 0, 2, 0, 2]$ and yields an objective value $w = -2$. This solution is not optimal.

- x_1 enters the basis and x_5 leaves it:

	x1	x2	x3	x4	x5	-z	-w	b
	0	0	1	1/2	-1/2	0	0	1
	1	1	0	-1/2	1/2	0	0	1
z	0	-2	0	1/2	-1/2	1	0	-1
w	0	0	0	0	-1	0	1	0

  The solution is optimal and $w^* = 0$ hence $x_5 = 0$ and we have a starting feasible solution for the original problem.

From now we can proceed with the Phase II of the simplex method, which works on the original problem as we have learned.

- First we rewrite the tableau that we reached by keeping only what we need:

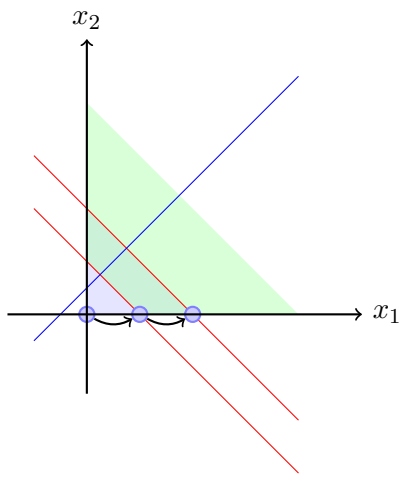


Figure 2.14: The geometric representation of the feasible example. The blue line represents the objective function.

```
|    | x1 | x2 | x3 | x4   | -z |  b |
|----+----+----+----+------+----+----|
|    |  0 |  0 |  1 | 1/2  |  0 |  1 |
|    |  1 |  1 |  0 | -1/2 |  0 |  1 |
|----+----+----+----+------+----+----|
| z  |  0 | -2 |  0 | 1/2  |  1 | -1 |
```

- x_4 enters the basis and x_3 leaves it:

```
|    | x1 | x2 | x3 | x4 | -z |  b |
|----+----+----+----+----+----+----|
|    |  0 |  0 |  2 |  1 |  0 |  2 |
|    |  1 |  1 |  1 |  0 |  0 |  2 |
|----+----+----+----+----+----+----|
| z  |  0 | -2 | -1 |  0 |  1 | -2 |
```

  The tableau is now optimal and the corresponding solution is: $x_1 = 2, x_2 = 0, x_3 = 0, x_4 = 2$, $z = 2$.

The solution process is geometrically depicted in Figure 2.14. Phase I starts from the origin, which for this problem is not a feasible solution, it then jumps to the first feasible solution and from there with the Phase II to the optimal solution.

Dictionary form In dictionary form, the auxiliary problem can be seen below:

$$\begin{aligned} \max \quad & x_1 - x_2 \\ & x_1 + x_2 \leq 2 \\ & 2x_1 + 2x_2 \geq 5 \\ & x_1, x_2 \geq 0 \end{aligned} \qquad \begin{aligned} x_3 &= 2 - x_1 - x_2 \\ x_4 &= -5 + 2x_1 + 2x_2 \\ \hline z &= x_1 + x_2 \end{aligned}$$

We introduce corrections of infeasibility

$$\begin{array}{rcl} \max\ -x_0 & \equiv & \min\ x_0 \\ x_1 + x_2 & & \leq 2 \\ 2x_1 + 2x_2 - x_0 & & \geq 5 \\ x_1, x_2, x_0 & \geq & 0 \end{array} \qquad \begin{array}{rcl} x_3 &=& 2 - x_1 - x_2 \\ x_4 &=& -5 + 2x_1 + 2x_2 + x_0 \\ \hline z &=& -x_0 \end{array}$$

This new problem is still infeasible but it can be made feasible by letting x_0 enter the basis. Which variable should leave? The most infeasible one: the variable that has the negative b term with the largest absolute value.

Chapter 3

Duality

In the previous chapter, we saw that the economical interpretation of a given LP problem leads us to define a dual problem. In this chapter, we look at the theory of duality from a more general and systematic perspective. We present four analytic ways to derive mathematically a dual problem. Then, we look at four important theorems that are at the foundation of linear programming. Finally, we present important practical uses of duality such as the dual simplex method, the sensitivity analysis and infeasibility proofs.

3.1 Derivation and Motivation

We consider three alternative ways to derive a dual problem for a given LP problem. The bounding approach, that we see first, is the most intuitive one and gives the opportunity to discuss a geometrical interpretation of duality. The multiplier approach is based on the theory of the simplex method. The last derivation is an application of the Lagrangian relaxation approach, an approach that can be also used as a general method for solving difficult problems and which is therefore worth learning. Once the reasons behind duality are understood, one can relay on the recipe approach, which is a mechanical way to write the dual.

3.1.1 Bounding approach

Suppose we have the following instance of LP problem:

$$\begin{array}{rrrrrrrl} \max & 4x_1 & + & x_2 & + & 3x_3 & = & z \\ & x_1 & + & 4x_2 & & & \leq & 1 \\ & 3x_1 & + & x_2 & + & x_3 & \leq & 3 \\ & & & & & x_1, x_2, x_3 & \geq & 0 \end{array}$$

Any feasible solution to this problem provides a **lower bound** to the objective value. By attempts, we can get

$$(x_1, x_2, x_3) = (1, 0, 0) \rightsquigarrow z^* \geq 4$$
$$(x_1, x_2, x_3) = (0, 0, 3) \rightsquigarrow z^* \geq 9$$

Which is the best one? Clearly the largest lower bound 9 is the best lower bound. If we knew that we cannot do better then we could claim the solution $(0, 0, 3)$ optimal. How can we know what is the best we can do? We could look at **upper bounds**. Let's try to derive one upper bound. We

multiply left and right hand sides of the constraint inequalities by a positive factor and sum the inequalities. This will not change the sense of the inequality.

$$\begin{array}{rrcrcrcr} & 2\cdot(& x_1 & + & 4x_2 & &) & \leq 2{\cdot}1 \\ & +\ 3\cdot(& 3x_1 & + & x_2 & + & x_3) & \leq 3{\cdot}3 \\ \hline 4x_1+x_2+3x_3 & \leq & \multicolumn{5}{c}{11x_1+11x_2+3x_3} & \leq\ 11 \end{array}$$

In the left-most side of the last row we rewrote the objective function of our problem. The two right-hand sides of the inequality obtained by summing left and right hand sides of the original constraints is certainly larger than the objective function. Indeed, the three variables must all be non-negative and their coefficients in the objective function are one by one smaller than their corresponding coefficients in the left-hand side of the obtained inequality. Hence $z^* \leq 11$. Is this the best upper bound we can find?

To obtain this upper bound we chose two arbitrary multipliers $y_1, y_2 \geq 0$ that preserve the sign of the inequalities and made a linear combination of the inequalities:

$$\begin{array}{lcl} y_1\cdot(\ x_1 + 4x_2\ ) & \leq & y_1(1) \\ +y_2\cdot(\ 3x_1 + x_2 + x_3) & \leq & y_2(3) \\ \hline (y_1+3y_2)x_1+(4y_1+y_2)x_2+y_2x_3 & \leq & y_1+3y_2 \end{array}$$

We aim at

$$c^Tx \leq y^TAx \leq y^Tb.$$

hence we have to impose some restrictions on the multipliers, namely that the coefficients of the variables x_i, $i = 1, 2, 3$ in the left-hand side of the linear combination of the constraints do not exceed the coefficient of the same variables in the objective function, that is,

$$\begin{array}{rcrcl} y_1 & + & 3y_2 & \geq & 4 \\ 4y_1 & + & y_2 & \geq & 1 \\ & & y_2 & \geq & 3 \end{array}$$

Thus

$$z = 4x_1 + x_2 + 3x_3 \leq (y_1 + 3y_2)x_1 + (4y_1 + y_2)x_2 + y_2x_3 \leq y_1 + 3y_2.$$

Then, to attain the best upper bound we need to solve the following problem:

$$\begin{array}{lrcrcl} \min & y_1 & + & 3y_2 & = & w \\ & y_1 & + & 3y_2 & \geq & 4 \\ & 4y_1 & + & y_2 & \geq & 1 \\ & & & y_2 & \geq & 3 \\ & & & y_1, y_2 & \geq & 0 \end{array}$$

This is the dual problem of our original instance of LP problem. We will soon prove that $z^* = w^*$.

3.1.2 Geometric Interpretation of Duality

Let's consider the following example:

$$\begin{array}{lrcrcr} \max & x_1 & + & x_2 & = & z \\ & 2x_1 & + & x_2 & \leq & 14 \\ & -x_1 & + & 2x_2 & \leq & 8 \\ & 2x_1 & - & x_2 & \leq & 10 \\ & & & x_1, x_2 & \geq & 0 \end{array}$$

The feasibility region and the objective function are plotted in Figure 3.1, left. The feasible solution $x^* = (4, 6)$ yields $z^* = 10$. To prove that this solution is optimal we need to show that no other feasible solution can do better. To do this we need to verify that $y^* = (3/5, 1/5, 0)$ is a feasible solution of the dual problem:

$$\begin{array}{rrrrrrr} \min & 14y_1 & + & 8y_2 & + & 10y_3 & = & w \\ & 2y_1 & - & y_2 & + & 2y_3 & \geq & 1 \\ & y_1 & + & 2y_2 & - & y_3 & \geq & 1 \\ & & & & & y_1, y_2, y_3 & \geq & 0 \end{array}$$

and that $w^* = 10$. To put it differently, we multiply the constraints $2x_1 + x_2 \leq 14$ by $3/5$ and multiply $-x_1 + 2x_2 \leq 8$ by $1/5$. Since the sum of the resulting two inequalities reads $x_1 + x_2 \leq 10$, we conclude that every feasible solution x_1, x_2 of the original problem satisfies $x_1 + x_2 \leq 10$,

$$\begin{array}{rrrrrr} \frac{3}{5}\cdot & 2x_1 & + & x_2 & \leq & 14 \\ \frac{1}{5}\cdot & -x_1 & + & 2x_2 & \leq & 8 \\ \hline & x_1 & + & x_2 & \leq & 10 \end{array}$$

Interpreted geometrically, this conclusion means that the entire region of feasibility is contained in the half plane $x_1 + x_2 \leq 10$; this fact is evident from Figure 3.1, center. Actually, we have support for a stronger conclusion: the set of all points (x_1, x_2) satisfying

$$\begin{array}{rrrrr} 2x_1 & + & x_2 & \leq & 14 \\ -x_1 & + & 2x_2 & \leq & 8 \end{array} \tag{3.1}$$

is a subset of the half plane $x_1 + x_2 \leq 10$. Now let us consider linear combinations of these two inequalities, Each of these combinations has the form:

$$(2v - w)x_1 + (v + 2w)x_2 \leq 14v + 8w \tag{3.2}$$

for some non-negative v and w. Geometrically, it represents a half-plane that contains the set represented by (3.1) and whose boundary line passes through the point $(4, 6)$. Examples of these boundary lines are

$$\begin{aligned} v = 1, w = 0 &\implies 2x_1 + x_2 = 14 \\ v = 1, w = 1 &\implies x_1 + 3x_2 = 22 \\ v = 2, w = 1 &\implies 3x_1 + 4x_2 = 36 \end{aligned}$$

and are shown in Figure 3.1, right.

The family of all lines (3.2) may be thought of as a single line that is attached to a hinge at $(4, 6)$ but is free to rotate on the hinge. Continuous changes of the multipliers v and w amount to continuous rotations of the line. One can choose non-negative v and w so as to make (3.2) coincide with the line $x_1 + x_2 = 10$. This claim will be shown by the Strong Duality Theorem.

We consider an alternative geometrical interpretation of duality for a general linear programming problem. This interpretation follows the same lines as the physical interpretation in [Matoušek and Gärtner, 2007]. Consider the LP problem:

$$\max\{\mathbf{c}^T\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\} \tag{3.3}$$

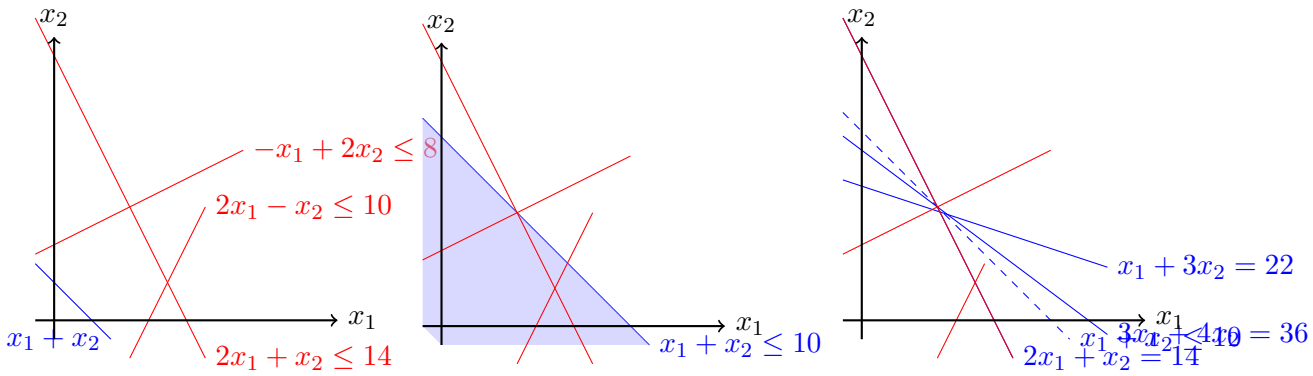


Figure 3.1:

and let $P = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$ be the feasible region, which is a polyhedron. Finding the maximum of (3.3) can be seen as shifting the hyperplane orthogonal to the vector $\mathbf{c}$ (recall from linear algebra that an hyperplane $\mathbf{c}^T\mathbf{x} = d$ has vector $\mathbf{c}$ orthogonal to it), as long as it contains points in P.

Suppose the maximum is finite, say its value is z^* and attained by the element $\mathbf{x}^*$ of P. Let

$$\begin{aligned} \mathbf{a}_1\mathbf{x} &\leq b_1 \\ &\vdots \\ \mathbf{a}_k\mathbf{x} &\leq b_k \end{aligned}$$

be the inequalities from $A\mathbf{x} \leq \mathbf{b}$ satisfied with equality by $\mathbf{x}^*$.

Now, geometric insight tells us that $\mathbf{c}^T\mathbf{x}^* = z^*$ is a *non-negative* combination of $\mathbf{a}_1\mathbf{x} = b_1, \ldots, \mathbf{a}_k\mathbf{x} = b_k$. Say

$$\begin{aligned} c &= y_1^*\mathbf{a}_1\mathbf{x} + \ldots + y_k^*\mathbf{a}_k\mathbf{x} \\ z^* &= y_1^*b_1 + \ldots + y_k^*b_k \end{aligned}$$

for $y_1^* \ldots y_k^* \geq 0$. This implies

$$\max\{\mathbf{c}^T\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\} = z^* = y_1^*b_1 + \ldots + y_k^*b_k \geq \min\{\mathbf{b}^T\mathbf{y} \mid \mathbf{y} \geq \mathbf{0}, A^T\mathbf{y} = \mathbf{c}\}$$

(the inequality follows since $\mathbf{y}^*$ gives a feasible solution for the minimum). The inequality holds also in the opposite verse:

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

This yields the LP-duality equation

$$\max\{\mathbf{c}^T\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\} = \min\{\mathbf{b}^T\mathbf{y} \mid \mathbf{y} \geq \mathbf{0}, A^T\mathbf{y} = \mathbf{c}\}.$$

3.1.3 Multipliers Approach

An alternative way to derive the dual problem arises from the simplex method. Throughout its iterations the simplex method performs linear combinations of the rows of the tableau aiming at a final tableau that is optimal. Given the LP problem $\max\{\mathbf{c}^T\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0\}$, the conditions of optimality for the tableau are $\bar{\mathbf{c}} \leq 0$. The multipliers of the rows of the tableau that have to be added to the last row can be multiplied by each other throughout the iterations of the simplex yielding a unique multiplier for each row to go in one step from the initial to the final tableau such that, when the algorithm terminates, the last row of the final tableau will be a linear combination of

the rows of the initial tableau. Let's call $\pi_1, \ldots, \pi_m, \pi_{m+1}$ the multipliers of the linear combination from the first to the last tableau.

$$\begin{array}{c} \pi_1 \\ \vdots \\ \pi_m \\ \pi_{m+1} \end{array} \left[\begin{array}{cccc:cccc:c:c} a_{11} & a_{12} & \ldots & a_{1n} & a_{1,n+1} & a_{1,n+2} & \ldots & a_{1,m+n} & 0 & b_1 \\ \vdots & \ddots & & & & & & & & \\ a_{m1} & a_{m2} & \ldots & a_{mn} & a_{m,n+1} & a_{m,n+2} & \ldots & a_{m,m+n} & 0 & b_m \\ \hdashline c_1 & c_2 & \ldots & c_n & 0 & 0 & \ldots & 0 & 1 & 0 \end{array}\right]$$

Working columnwise, the conditions of optimality since $\bar{c}_k \leq 0$ for all $k = 1, \ldots, n+m$ can be written as

$$\left\{\begin{array}{rcccccccl} \pi_1 a_{11} & + & \pi_2 a_{21} & \ldots + & \pi_m a_{m1} & + \pi_{m+1} c_1 & \leq & 0 \\ \vdots & \ddots & & & & & & \vdots \\ \pi_1 a_{1n} & + & \pi_2 a_{2n} & \ldots + & \pi_m a_{mn} & + \pi_{m+1} c_n & \leq & 0 \\ \hdashline \pi_1 a_{1,n+1}, & & \pi_2 a_{2,n+1}, & \ldots & \pi_m a_{m,n+1} & & \leq & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ \pi_1 a_{1,n+m}, & & \pi_2 a_{2,n+m}, & \ldots & \pi_m a_{m,n+m} & & \leq & 0 \\ \hdashline & & & & & \pi_{m+1} & = & 1 \\ \hdashline \pi_1 b_1 & + & \pi_2 b_2 & \ldots + & \pi_m b_m & & (\leq & 0) \end{array}\right. \tag{3.4}$$

For the columns that correspond to the variables in basis (eg, from $n+1$ to $n+m$, ie, the second block of constraints in the system (3.4)) we will have an identity matrix and hence $\pi_i \leq 0$, $i = 1..m$. From the last row of the final tableau we read $z = -\boldsymbol{\pi}\mathbf{b}$. Since we want to maximize z then we would try to solve for $\min(-\boldsymbol{\pi}\mathbf{b})$ or equivalently for $\max \boldsymbol{\pi}\mathbf{b}$. We can then rewrite:

$$\begin{array}{rcccccccl} \max & \pi_1 b_1 & + & \pi_2 b_2 & \ldots + & \pi_m b_m & & & \\ & \pi_1 a_{11} & + & \pi_2 a_{21} & \ldots + & \pi_m a_{m1} & \leq & -c_1 \\ & \vdots & \ddots & & & & & \vdots \\ & \pi_1 a_{1n} & + & \pi_2 a_{2n} & \ldots + & \pi_m a_{mn} & \leq & -c_n \\ & & & & & \pi_1, \pi_2, \ldots \pi_m & \leq & 0 \end{array}$$

and setting $\mathbf{y} = -\boldsymbol{\pi}$ and substituting we obtain:

$$\begin{array}{rcccccccl} \max & -y_1 b_1 & + & -y_2 b_2 & \ldots + & -y_m b_m & & & \\ & -y_1 a_{11} & + & -y_2 a_{21} & \ldots + & -y_m a_{m1} & \leq & -c_1 \\ & \vdots & \ddots & & & & & \vdots \\ & -y_1 a_{1n} & + & -y_2 a_{2n} & \ldots + & -y_m a_{mn} & \leq & -c_n \\ & & & & & -y_1, -y_2, \ldots - y_m & \leq & 0 \end{array}$$

which leads to the same dual as we saw in the previous section:

$$\begin{array}{rcl} \min \quad \mathbf{b}^T\mathbf{y} & = & w \\ A^T\mathbf{y} & \geq & \mathbf{c} \\ \mathbf{y} & \geq & 0 \end{array}$$

Example 3.1. We sketch a numerical example:

$$\begin{array}{rcccccl} \max & 6x_1 & + & 8x_2 & & \\ & 5x_1 & + & 10x_2 & \leq & 60 \\ & 4x_1 & + & 4x_2 & \leq & 40 \\ & & & x_1, x_2 & \geq & 0 \end{array}$$

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

Figure 3.2: The recipe to transform a primal problem in the into its dual.

$$\begin{cases} 5\pi_1 + 4\pi_2 + 6\pi_3 \leq 0 \\ 10\pi_1 + 4\pi_2 + 8\pi_3 \leq 0 \\ 1\pi_1 + 0\pi_2 + 0\pi_3 \leq 0 \\ 0\pi_1 + 1\pi_2 + 0\pi_3 \leq 0 \\ 0\pi_1 + 0\pi_2 + 1\pi_3 = 1 \\ 60\pi_1 + 40\pi_2 \end{cases}$$

which we transform by setting:

$$\begin{aligned} y_1 &= -\pi_1 \geq 0 \\ y_2 &= -\pi_2 \geq 0 \end{aligned}$$

3.1.4 Duality Recipe

We saw earlier that any problem can be put in the standard form and now we derived the dual problem for this form. By using the same transformation techniques we can write the mechanical rules to write the dual of a certain LP problem. They are listed in Figure 3.2, which is taken from [Matoušek and Gärtner, 2007].

3.2 Duality Theory

Theorem 3.1 (Symmetry of duality)**.** *The dual of the dual is the primal.*

Proof. Let's write the dual pair:

Primal problem:

$$\text{(P)} \qquad \begin{array}{rcl} \max \quad z & = & \mathbf{c}^T\mathbf{x} \\ A\mathbf{x} & \leq & \mathbf{b} \\ \mathbf{x} & \geq & \mathbf{0} \end{array}$$

Dual problem:

$$\text{(D)} \qquad \begin{array}{rcl} \min \quad w & = & \mathbf{b}^T\mathbf{y} \\ A^T\mathbf{y} & \geq & \mathbf{c} \\ \mathbf{y} & \geq & \mathbf{0}. \end{array}$$

We proceed to derive the dual of (D). We put first the dual in the standard form:

$$\begin{array}{rcl} \min \quad \mathbf{b}^T\mathbf{y} & \equiv & -\max -\mathbf{b}^T\mathbf{y} \\ -A^T\mathbf{y} & \leq & -\mathbf{c} \\ \mathbf{y} & \geq & \mathbf{0} \end{array}$$

then we use the same rules used to derive (D) from (P) to derive the dual of (D)

$$\begin{array}{rcl} -\min \quad -\mathbf{c}^T\mathbf{x} & & \\ -A\mathbf{x} & \geq & -\mathbf{b} \\ \mathbf{x} & \geq & \mathbf{0} \end{array}$$

□

From the derivations we saw already that the dual problem yields upper bounds (to maximization primal problems). This is true in general:

Theorem 3.2 (Weak Duality Theorem)**.** *Given:*

$$\begin{array}{l} \text{(P)} \ \max\{\mathbf{c}^T\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\} \\ \text{(D)} \ \min\{\mathbf{b}^T\mathbf{y} \mid A^T\mathbf{y} \geq \mathbf{c}, \mathbf{y} \geq \mathbf{0}\} \end{array}$$

for any feasible solution $\mathbf{x}$ *of (P) and any feasible solution* $\mathbf{y}$ *of (D):*

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

Proof. In scalar form, from the feasibility of $\mathbf{x}$ in (P) we have that $\sum_{j=1}^{n} a_{ij}x_i \leq b_i$ for $i = 1..m$ and $x_j \geq 0$ for $j = 1..n$. From the feasibility of $\mathbf{y}$ in (D) we have that $c_j \leq \sum_{i=1}^{m} y_i a_{ij}$ for $j = 1..n$ and $y_i \geq 0$ for $i = 1..m$. It follows that:

$$c_j x_j \leq \left(\sum_{i=1}^{m} y_i a_{ij}\right) x_j \qquad \forall j = 1..n.$$

since we just multiplied by a positive value left and right hand side of each dual constraint. Summing all left hand sides and right hand sides:

$$\sum_{j=1}^{n} c_j x_j \leq \sum_{j=1}^{n} \left(\sum_{i=1}^{m} y_i a_{ij}\right) x_j$$

and commuting the summations:

$$\sum_{j=1}^{n} c_j x_j \leq \sum_{j=1}^{n} \left(\sum_{i=1}^{m} y_i a_{ij}\right) x_j = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_i\right) y_i \leq \sum_{i=1}^{m} b_i y_i.$$

□

The following theorem is due to Von Neumann and Dantzig, 1947, and Gale, Kuhn and Tucker, 1951.

Theorem 3.3 (Strong Duality Theorem)**.** *Given:*

$$\text{(P)} \quad \max\{\mathbf{c}^T\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$$
$$\text{(D)} \quad \min\{\mathbf{b}^T\mathbf{y} \mid A^T\mathbf{y} \geq \mathbf{c}, \mathbf{y} \geq \mathbf{0}\}$$

exactly one of the following occurs:

1. *(P) and (D) are both infeasible*
2. *(P) is unbounded and (D) is infeasible*
3. *(P) is infeasible and (D) is unbounded*
4. *(P) has feasible solution* $\mathbf{x}^* = [x_1^*, \ldots, x_n^*]^T$*, (D) has feasible solution* $\mathbf{y}^* = [y_1^*, \ldots, y_m^*]^T$ *and*

$$\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*$$

Proof. All other combinations of the 3 possible outcomes (Optimal, Infeasible, Unbounded) for (P) and (D) are ruled out by the weak duality theorem. For example, the combination (P) unbounded ($+\infty$) and (D) unbounded ($-\infty$) would clearly violate the weak duality theorem. To show 4, we use the simplex method (other proofs independent of the simplex method exist, eg, via Farkas Lemma and convex polyhedral analysis.)

To prove the statement 4. for an optimal solution $\mathbf{x}^*$ we need to exhibit a dual feasible solution $\mathbf{y}^*$ satisfying $\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*$. Suppose we apply the simplex method. We know that the simplex method produces an optimal solution whenever one exists, and in statement 4. we are assuming that one does indeed exist. Let $\mathbf{x}^*$ be this optimal solution. The final tableau will be an optimal tableau for the primal problem. The objective function in this final tableau is ordinarily written (see page 34) as

$$z = \bar{d} + \sum_{k=1}^{n+m} \bar{c}_k x_k = \bar{d} + \sum_{r \in B} \bar{c}_r x_r + \sum_{s \notin B} \bar{c}_s x_s$$

But, since this is the optimal tableau and we prefer stars to bars for denoting optimal "stuff", let us write z^* instead of $\bar{d}$. Also, the reduced costs of the basic variables will be zero and nonbasic variables will generally consist of a combination of original variables as well as slack variables. Instead of using $\bar{c}_k$ for the coefficients of all these variables, let us differentiate and use $\bar{c}_j$ for the objective coefficients corresponding to original variables, and let us use $\bar{c}_i$ for the objective coefficients corresponding to slack variables. Also, for those original variables that are basic we put $\bar{c}_j = 0$, and for those slack variables that are basic we put $\bar{c}_i = 0$. With these new notations, we can rewrite the objective function as

$$z = z^* + \sum_{j=1}^{n} \bar{c}_j x_j + \sum_{i=1}^{m} \bar{c}_{n+i} x_{n+i} \tag{3.5}$$

Note that since we know that x^* is the optimal solution to the primal, then we can also write:

$$z^* = \sum_{j=1}^{n} c_j x_j^*$$

(This is just the original objective function with the substitution of the optimal value for the original variables.)

We now define

$$\boxed{y_i^* = -\bar{c}_{n+i}, \quad i = 1, 2, \ldots, m.}$$

and we claim that $\mathbf{y}^* = [y_1^*, y_2^*, \ldots, y_m^*]^T$ is a **dual feasible solution satisfying** $\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*$.

Let's verify the claim. We substitute in (3.5):

- $z = \sum_{j=1}^{n} c_j x_j$,
- $\bar{c}_{n+i} = -y_i^*$ for $i = 1, 2, \ldots, m$ and
- $x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j$ for $i = 1, 2, \ldots, m$ (from the definition of the m slack variables)

and obtain

$$\begin{aligned}\sum_{j=1}^{n} c_j x_j &= z^* + \sum_{j=1}^{n} \bar{c}_j x_j - \sum_{i=1}^{m} y_i^* \left(b_i - \sum_{j=1}^{n} a_{ij} x_j \right) \\ &= \left(z^* - \sum_{i=1}^{m} y_i^* b_i \right) + \sum_{j=1}^{n} \left(\bar{c}_j + \sum_{i=1}^{m} a_{ij} y_i^* \right) x_j\end{aligned}$$

This must hold for every $[x_1, x_2, \ldots, x_n]$ hence:

$$z^* = \sum_{i=1}^{m} b_i y_i^* \tag{3.6}$$

$$c_j = \bar{c}_j + \sum_{i=1}^{m} a_{ij} y_i^*, j = 1, 2, \ldots, n \tag{3.7}$$

Equation (3.6) implies that $\mathbf{y}^*$ satisfies $\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*$. Since $\bar{c}_k \leq 0$ for all $k = 1, 2, \ldots, n+m$ (optimality condition for the final tableau), for the original variables we have:

$$\bar{c}_j \leq 0 \implies c_j - \sum_{i=1}^{m} y_i^* a_{ij} \leq 0 \implies \sum_{i=1}^{m} y_i^* a_{ij} \geq c_j \qquad j = 1, 2, \ldots, n \tag{3.8}$$

and for the slack variables:

$$\bar{c}_{n+i} \leq 0 \implies y_i^* = -\bar{c}_{n+i} \geq 0 \implies y_i^* \geq 0 \qquad i = 1, 2, \ldots, m \tag{3.9}$$

Equations (3.8)-(3.9) imply that y^* is also dual feasible solution. □

Theorem 3.4 (Complementary Slackness)**.** *Given a feasible solution* $\mathbf{x}^*$ *for (P) and a feasible solution* $\mathbf{y}^*$ *for (D), necessary and sufficient conditions for the optimality of both are:*

$$\left(c_j - \sum_{i=1}^{m} y_i^* a_{ij} \right) x_j^* = 0, \quad j = 1, \ldots, n$$

This implies that for any $j = 1..n$*, if* $x_j^* \neq 0$ *then* $\sum y_i^* a_{ij} = c_j$ *(no slack nor surplus) and if* $\sum y_i^* a_{ij} > c_j$ *then* $x_j^* = 0$.

Proof. From the weak duality theorem:

$$z^* = \mathbf{c}^T\mathbf{x}^* \leq \mathbf{y}^{*T}A\mathbf{x}^* \leq \mathbf{b}^T\mathbf{y}^* = w^*.$$

Then, from the strong duality theorem $z^* = w^*$, hence:

$$\mathbf{c}^T\mathbf{x}^* - \mathbf{y}^{*T}A\mathbf{x}^* = 0$$

In scalars:

$$\sum_{j=1}^{n} \underbrace{\left(c_j - \sum_{i=1}^{m} y_i^* a_{ij}\right)}_{\leq 0} \underbrace{x_j^*}_{\geq 0} = 0$$

Hence each term of the sum for $j = 1..n$ must be $= 0$. □

Application: Economic Interpretation

$$\begin{array}{rrcrcrcl} \max & 5x_0 & + & 6x_1 & + & 8x_2 & & \\ & 6x_0 & + & 5x_1 & + & 10x_2 & \leq & 60 \\ & 8x_0 & + & 4x_1 & + & 4x_2 & \leq & 40 \\ & 4x_0 & + & 5x_1 & + & 6x_2 & \leq & 50 \\ & & & & & x_0, x_1, x_2 & \geq & 0 \end{array}$$

final tableau:

$x0$	$x1$	$x2$	$s1$	$s2$	$s3$	$-z$	b
	0	1		0			5/2
	1	0		0			7
	0	0		1			2
−1/5	0	0	−1/5	0	−1		−62

- Which are the values of the original variables? $(0, 7, 5/2)$
- Which are the values of the reduced costs? $(-1/5, 0, 0, -1/5, 0, -1)$
- Which are the values of the tableau multipliers? $(-1/5, 0, -1)$
- Which are the values of the dual variables? $(1/5, 0, 1)$ - for the proof of the Strong duality theorem.
- What is the value of an extra unit of resource capacity (shadow prices or marginal values of the resources)? Strictly speaking we are interested in the marginal values of the capacities of each resource, that is, the effect of very small increases or decreases in capacity Williams [2013]. $(1/5, 0, 1)$
- Which is the value of the slack variables? $(0, 2, 0)$
- If one slack variable > 0 then there is overcapacity, that is, the constraint to which the slack variable belongs is not tight. This can be assessed also via complementary slackness theorem: y_2 is dual variable associated with the second constraint, $y_2 = 0$ from the tableau, hence the second constraint is not binding.
- How many products can be produced at most? at most $m = 3$.

Game: Suppose two economic operators:

- P owns the factory and produces goods
- D is the market buying and selling raw material and resources
- D asks P to close and sell him all resources
- P considers if the offer is convenient
- D wants to spend least possible
- $\mathbf{y}$ are prices that D offers for the resources
- $\sum y_i b_i$ is the amount D has to pay to have all resources of P
- $\sum y_i a_{ij} \geq c_j$, ie, the total value to produce j must be larger than the price per unit of product for the offer to be convinient for P
- P either sells all resources $\sum y_i a_{ij}$ or produces product j and earns c_j
- without constraint $\sum y_i a_{ij} \geq c_j$ there would not be negotiation because P would be better off producing and selling
- at optimality the situation is indifferent (strong duality theorem)
- if for a product j it is $\sum_i y_i a_{ij} > c_j$ then it is not profitable producing j (complementary slackness theorem). Example, product 0.
- resource 2 that was not totally utilized in the primal has been given value 0 in the dual. (complementary slackness theorem) Plausible, since we do not use the whole resource available, then it is likely that we do not place so much value on it.

3.3 Lagrangian Duality

If a problem is hard to solve then a possible approach is to find an easier problem resembling the original one that provides information in terms of bounds. In this context one then wishes to find the strongest bounds. A **relaxed** problem is a new problem in which some requirements present in the original problem have been omitted.

Consider the following example:

$$\begin{array}{rrrrrrrrl} \min & 13x_1 & + & 6x_2 & + & 4x_3 & + & 12x_4 & \\ & 2x_1 & + & 3x_2 & + & 4x_3 & + & 5x_4 & = 7 \\ & 3x_1 & + & & + & 2x_3 & + & 4x_4 & = 2 \\ & & & & & x_1, x_2, x_3, x_4 & & & \geq 0 \end{array}$$

We wish to reduce this problem to an easier one, ie:

$$\begin{array}{rrrrrrl} \min & c_1x_1 & + & c_2x_2 & + & \ldots + c_nx_n & \\ & & & & & x_1, x_2, \ldots, x_n & \geq 0 \end{array}$$

This problem is solvable by inspection: if $c_i < 0$ then $x_i = +\infty$, if $c_i \geq 0$ then $x_i = 0$.

Let's then relax the constraints of our problem by adding to the function to minimize a measure of the violations of each constraint, ie:

$$\begin{array}{rcl} 7 & - & (2x_1 + 3x_2 + 4x_3 + 5x_4) \\ 2 & - & (3x_1 + \quad + 2x_3 + 4x_4) \end{array}$$

We relax these measures in the objective function with **Lagrangian multipliers** y_1, y_2. We obtain a family of problems, one problem for each value of y_1 and y_2:

$$PR(y_1, y_2) = \min_{x_1,x_2,x_3,x_4 \geq 0} \left\{ \begin{array}{r} 13x_1 + 6x_2 + 4x_3 + 12x_4 \\ +y_1(7 - (\ 2x_1 + 3x_2 + 4x_3 + 5x_4)) \\ +y_2(2 - (\ 3x_1 + \quad + 2x_3 + 4x_4)) \end{array} \right\}$$

The following two properties hold:

1. for all $y_1, y_2 \in \mathbb{R} : \text{opt}(PR(y_1, y_2)) \leq \text{opt}(P)$
2. $\max_{y_1,y_2 \in \mathbb{R}}\{\text{opt}(PR(y_1, y_2))\} \leq \text{opt}(P)$

The problem PR is easy to solve:

$$PR(y_1, y_2) = \min_{x_1,x_2,x_3,x_4 \geq 0} \left\{ \begin{array}{rcl} & (13 - 2y_1 - 3y_2) & x_1 \\ + & (6 - 3y_1 \quad\quad) & x_2 \\ + & (4 - 4y_1 - 2y_2) & x_3 \\ + & (12 - 5y_1 - 4y_2) & x_4 \\ + & 7y_1 + 2y_2 & \end{array} \right\}$$

if the coefficients of x_i is < 0 then the bound is $-\infty$ and hence useless to our purposes of finding the strongest bound. Hence,

$$\begin{array}{rcl} (13 - 2y_1 - 3y_2) & \geq & 0 \\ (6 - 3y_1 \quad\quad) & \geq & 0 \\ (4 - 4y_1 - 2y_2) & \geq & 0 \\ (12 - 5y_1 - 4y_2) & \geq & 0 \end{array}$$

If they all hold then we are left with $7y_1 + 2y_2$ because all other terms go to 0. Thus,

$$\begin{array}{rrcl} \max & 7y_1 + 2y_2 & & \\ & 2y_1 + 3y_2 & \leq & 13 \\ & 3y_1 & \leq & 6 \\ & 4y_1 + 2y_2 & \leq & 4 \\ & 5y_1 + 4y_2 & \leq & 12 \end{array}$$

which is the dual problem we were trying to derive.

The general derivation in vector notation is the following:

$$\begin{array}{rrcll} \min & z & = & \mathbf{c}^T\mathbf{x} & \mathbf{c} \in \mathbb{R}^n \\ & A\mathbf{x} & = & \mathbf{b} & A \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^m \\ & \mathbf{x} & \geq & \mathbf{0} & \mathbf{x} \in \mathbb{R}^n \end{array}$$

$$\max_{y \in \mathbb{R}^m}\{\min_{x \in \mathbb{R}^n_+}\{\mathbf{c}^T\mathbf{x} + \mathbf{y}^T(\mathbf{b} - A\mathbf{x})\}\}$$

$$\max_{y \in \mathbb{R}^m}\{\min_{x \in \mathbb{R}^n_+}\{(\mathbf{c} - \mathbf{y}A)\mathbf{x} + \mathbf{y}\mathbf{b}\}\}$$

$$\begin{array}{rrcl} \max & \mathbf{b}^T\mathbf{y} & & \\ & A^T\mathbf{y} & \leq & \mathbf{c} \\ & \mathbf{y} \in \mathbb{R}^m & & \end{array}$$

3.4 Dual Simplex

We saw that as the simplex method solves the primal problem, it also implicitly solves the dual problem. Indeed, the value of the dual variables is the value of the reduced cost of the slack variables with a change of sign. This fact is a consequence of the strong duality theorem or of the multiplier method for the derivation of the dual. The idea is then to apply the simplex method to the dual problem and observe what happens in the primal tableau. We obtain a new algorithm for the primal problem: the *dual simplex* (Lemke, 1954). The dual simplex corresponds to the primal simplex applied to the dual.

The derivation of this new algorithm can be obtained by following step by step the meaning of the pivot iterations in the primal and their corresponding meaning in the dual when it is put in standard form:

$$\begin{aligned}\max\{\mathbf{c}^T\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\} &= \min\{\mathbf{b}^T\mathbf{y} \mid A^T\mathbf{y} \geq \mathbf{c}^T, \mathbf{y} \geq \mathbf{0}\} \\ &= -\max\{-\mathbf{b}^T\mathbf{y} \mid -A^T\mathbf{x} \leq -\mathbf{c}^T, \mathbf{y} \geq 0\}\end{aligned}$$

The resulting pivot operation is given here in the right column:

Primal simplex on primal problem:

1. pivot > 0
2. col c_j with wrong sign[a]
3. row: $\min\left\{\frac{b_i}{a_{ij}} : a_{ij} > 0, i = 1, .., m\right\}$

[a] Wrong in terms of optimality, eg, for a maximization problem a positive sign is a wrong sign because it does not prove optimality.

Dual simplex on primal problem:

1. pivot < 0
2. row $b_i < 0$
   (condition of feasibility)
3. col: $\min\left\{\left|\frac{c_j}{a_{ij}}\right| : a_{ij} < 0, j = 1, 2, .., n + m\right\}$
   (least worsening solution)

The dual simplex can sometimes be preferable to the primal simplex. Since the running time of the primal simplex is in practice between $2m$ and $3m$ iterations, then if $m = 99$ and $n = 9$ it is better to transform the problem in its dual and solve by the primal algorithm, or even better leave the problem as it is but solve it with the dual simplex.

Another application is the following. Since the last terms of the tableau become the right hand side terms of the constraints, whenever a tableau in the primal problem is not optimal, the corresponding tableau in the dual problem is non canonical (or infeasible). Iterating in the two algorithms we observe that:

- the primal simplex works with feasible solutions towards optimality
- the dual simplex works with optimal solutions towards feasibility.

While in the primal simplex we increase a variable that can improve the objective function, in the dual simplex we take a constraint that is not yet satisfied and use it to diminish the value of a variable until the constraint becomes satisfied. See Figure 3.3.

Hence, the dual simplex applied on the primal problem can be used for resolving an infeasible start. This yields a dual based Phase I algorithm (*Dual-primal algorithm*).

If b has all nonnegative components and c_N has all nonpositive components, then this dictionary is optimal – the problem was trivial. Suppose, however, that one of these two vectors (but not

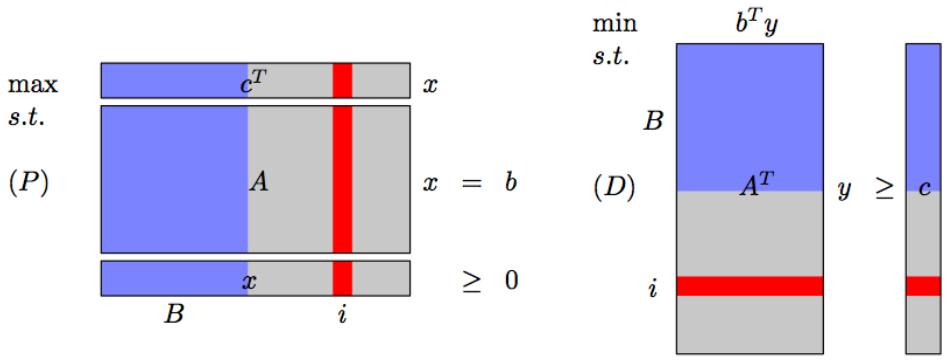


Figure 3.3: The figure shows an iteration of the simplex in the primal problem and the corresponding step in the dual problem.

both) has components of the wrong sign. For example, suppose that b is okay (all nonnegative components) but c_N has some positive components. Then this dictionary is primal feasible, and we can start immediately with the primal simplex method. On the other hand, suppose that c_N has all nonpositive components but b has some negative ones. Then the starting dictionary is dual feasible, and we can commence immediately with the dual simplex algorithm.

The last, and most common, case is where both b and c_N have components of the wrong sign. In this case, we must employ a two-phase procedure. There are two choices. We could temporarily replace c_N with another vector that is nonpositive. Then the modified problem is dual feasible, and so we can apply the dual simplex method to find an optimal solution of this modified problem. After that, the original objective function could be reinstated. With the original objective function, the optimal solution from Phase I is most likely not optimal, but it is feasible, and therefore the primal simplex method can be used to find the optimal solution to the original problem.

The other choice would be to modify b instead of c_N, thereby obtaining a primal feasible solution to a modified problem. Then we would use the primal simplex method on the modified problem to obtain its optimal solution, which will then be dual feasible for the original problem, and so the dual simplex method can be used to finish the problem Vanderbei [2008].

Example 3.2. Consider the following LP problem with its dual:

$$\begin{aligned} \max \quad & -x_1 - x_2 \\ & -2x_1 - x_2 \leq 4 \\ & -2x_1 + 4x_2 \leq -8 \\ & -x_1 + 3x_2 \leq -7 \\ & x_1, x_2 \geq 0 \end{aligned} \qquad \begin{aligned} \min \quad & 4y_1 - 8y_2 - 7y_3 \\ & -2y_1 - 2y_2 - y_3 \geq -1 \\ & -y_1 + 4y_2 + 3y_3 \geq -1 \\ & y_1, y_2, y_3 \geq 0 \end{aligned}$$

We solve in parallel both problems:

- The initial tableau

	x1	x2	w1	w2	w3	-z	b
	-2	-1	1	0	0	0	4
	-2	4	0	1	0	0	-8
	-1	3	0	0	1	0	-7
	-1	-1	0	0	0	1	0

  exhibits an infeasible start.
  Hence, we use the dual simplex.

- The initial tableau ($\min by \equiv -\max -by$)

	y1	y2	y3	z1	z2	-p	b
	2	2	1	1	0	0	1
	1	-4	-3	0	1	0	1
	-4	8	7	0	0	1	0

  has a feasible start (thanks to $-x_1 - x_2$)
  Hence, we use the primal simplex.

- x_1 enters, w_2 leaves

	x1	x2	w1	w2	w3	-z	b
	0	-5	1	-1	0	0	12
	1	-2	0	-0.5	0	0	4
	0	1	0	-0.5	1	0	-3
	0	-3	0	-0.5	0	1	4

- y_2 enters, z_1 leaves

	y1	y2	y3	z1	z2	-p	b
	1	1	0.5	0.5	0	0	0.5
	5	0	-1	2	1	0	3
	-4	0	3	-12	0	1	-4

- w_2 enters, w_3 leaves (note that we kept $c_j < 0$, ie, optimality)

	x1	x2	w1	w2	w3	-z	b
	0	-7	1	0	-2	0	18
	1	-3	0	0	-1	0	7
	0	-2	0	1	-2	0	6
	0	-4	0	0	-1	1	7

- y_3 enters, y_2 leaves

	y1	y2	y3	z1	z2	-p	b
	2	2	1	1	0	0	1
	7	2	0	3	1	0	3
	-18	-6	0	-7	0	1	-7

The Phase I is thus terminated. We note that the tableaux are optimal also with respect to the Phase II hence we are done.

3.5 Sensitivity Analysis

The *sensitivity analysis* is done when the final tableau has been reached. Hence, it is also called *postoptimality analysis*. It aims at assessing the robustness of the solution or the effects of small changes to the input data.

We recall the economic interpretation of an LP problem. We reconsider our starting example of resource allocation problem to determine the best product mix.

$$\begin{array}{rrrrrrl} \max & 5x_0 & + & 6x_1 & + & 8x_2 & \\ & 6x_0 & + & 5x_1 & + & 10x_2 & \leq 60 \\ & 8x_0 & + & 4x_1 & + & 4x_2 & \leq 40 \\ & 4x_0 & + & 5x_1 & + & 6x_2 & \leq 50 \\ & & & & & x_0, x_1, x_2 & \geq 0 \end{array}$$

The final tableau for this problem is the following (we show only the numbers that are relevant for our analysis):

x_0	x_1	x_2	s_1	s_2	s_3	$-z$	b
	0	1		0			5/2
	1	0		0			7
	0	0		1			2
−1/5	0	0	−1/5	0	−1		−62

What-if analysis: what changes in the solution if some input data changes? Instead of solving each modified problems from scratch, exploit results obtained from solving the original problem.

- How much more expensive a product not selected should be? Look at reduced costs, we want: $c_i - \pi \mathbf{a}_j > 0$ hence we must increase the original cost c_i.
- What is the value of extra capacity of manpower? Adding $1 + 1$ units of the first and third resource we obtain an increase in objective value of $1/5 + 1$, respectively.

Let's consider the possibilities for a general case:

$$\max\{\mathbf{c}^T\mathbf{x} \mid A\mathbf{x} = \mathbf{b}, \mathbf{l} \le \mathbf{x} \le \mathbf{u}\} \tag{3.10}$$

with optimal solution $\mathbf{x}^*$.

(I) changes to coefficients of the objective function: $\max\{\tilde{\mathbf{c}}^T\mathbf{x} \mid A\mathbf{x} = \mathbf{b}, \mathbf{l} \le \mathbf{x} \le \mathbf{u}\}$
$\mathbf{x}^*$ of (3.10) remains feasible hence we can restart the simplex from $\mathbf{x}^*$ (primal iteration).

(II) changes to RHS terms: $\max\{\mathbf{c}^T\mathbf{x} \mid A\mathbf{x} = \tilde{\mathbf{b}}, \mathbf{l} \le \mathbf{x} \le \mathbf{u}\}$
from $\mathbf{x}^*$ optimal feasible solution of (3.10) construct a basic sol $\bar{\mathbf{x}}$ of (II): $\bar{\mathbf{x}}_N = \mathbf{x}^*_N$, $A_B\bar{\mathbf{x}}_B = \tilde{\mathbf{b}} - A_N\bar{\mathbf{x}}_N$. $\bar{\mathbf{x}}$ is dual feasible and we can start the dual simplex from there (dual iteration). If $\tilde{\mathbf{b}}$ differs from $\mathbf{b}$ only slightly it may be we are already optimal.

(III) introduce a new variable:

$$\begin{array}{rr} \max & \sum_{j=1}^{6} c_j x_j \\ & \sum_{j=1}^{6} a_{ij}x_j = b_i,\ i = 1,\dots,3 \\ & l_j \le x_j \le u_j,\ j = 1,\dots,6 \\ & [x_1^*,\dots,x_6^*] \text{ feasible} \end{array} \qquad \begin{array}{rr} \max & \sum_{j=1}^{7} c_j x_j \\ & \sum_{j=1}^{7} a_{ij}x_j = b_i,\ i = 1,\dots,3 \\ & l_j \le x_j \le u_j,\ j = 1,\dots,7 \\ & [x_1^*,\dots,x_6^*,0] \text{ feasible} \end{array}$$

A new feasible solution is easily derived by setting the new variable to zero. We need to check whether it is worth increasing it (primal iteration).

(IV) introduce a new constraint:

$$\begin{array}{l} \sum_{j=1}^{6} a_{4j}x_j = b_4 \\ \sum_{j=1}^{6} a_{5j}x_j = b_5 \\ \quad l_j \le x_j \le u_j \quad j = 7,8 \end{array} \qquad \begin{array}{l} (x_1^*,\dots,x_6^*) \text{ optimal} \\ (x_1^*,\dots,x_6^*,x_7^*,x_8^*) \text{ feasible} \\ x_7^* = b_4 - \sum_{j=1}^{6} a_{4j}x_j^* \\ x_8^* = b_5 - \sum_{j=1}^{6} a_{5j}x_j^* \end{array}$$

It may render $\mathbf{x}^*$ infeasible (dual iteration).

Example 3.3. (I) Variation of a coefficient in the objective function.

$$\begin{array}{rrrrrrr} \max & 6x_1 & + & 8x_2 & & & \\ & 5x_1 & + & 10x_2 & \le & 60 \\ & 4x_1 & + & 4x_2 & \le & 40 \\ & & & x_1, x_2 & \ge & 0 \end{array}$$

	x_1	x_2	x_3	x_4	$-z$	b
x_3	5	10	1	0	0	60
x_4	4	4	0	1	0	40
	6	8	0	0	1	0

	x_1	x_2	x_3	x_4	$-z$	b
x_2	0	1	1/5	−1/4	0	2
x_1	1	0	−1/5	1/2	0	8
	0	0	**-2/5**	**-1**	**1**	−64

The last tableau on the right gives the possibility to estimate the effect of variations. For a variable in basis the perturbation goes unchanged in the reduced costs. Eg:

$$\max(6+\delta)x_1 + 8x_2 \implies \bar{c}_1 = -\frac{2}{5}\cdot 5 - 1\cdot 4 + 1(6+\delta) = \delta.$$

If $\delta > 0$ then the variable must enter in basis and we need to bring the tableau in canonical form for the new basis and hence δ changes the obj value. For a variable not in basis, if it changes the sign of the reduced cost then it is worth bringing in basis. The δ term propagates to other columns via pivot operations.

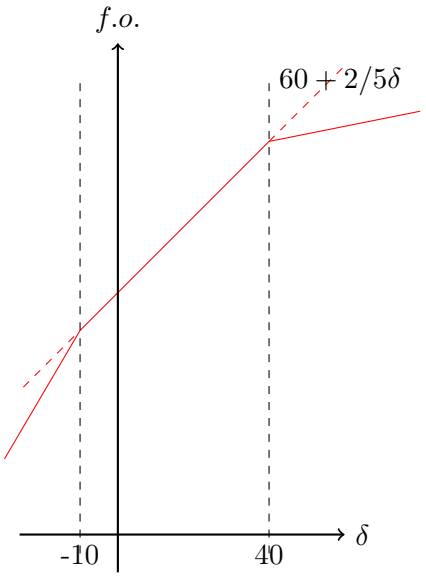


Figure 3.4: The influence of δ on the objective value.

Example 3.4. (II) Changes in the right-hand side (RHS) terms

	x_1	x_2	x_3	x_4	$-z$	b
x_3	5	10	1	0	0	$60+\delta$
x_4	4	4	0	1	0	$40+\epsilon$
	6	8	0	0	1	0

	x_1	x_2	x_3	x_4	$-z$	b
x_2	0	1	**1/5**	**-1/4**	**0**	$2+1/5\delta-1/4\epsilon$
x_1	1	0	$-1/5$	$1/2$	0	$8-1/5\delta+1/2\epsilon$
	0	0	**-2/5**	**-1**	**1**	$-64-2/5\delta-\epsilon$

Looking at the cell in the bottom-left corner of the tableau, $-64-2/5\delta-\epsilon$, we see what would be the contribution to the objective value of an increase of δ and ϵ of the resources. If both $\delta=\epsilon=1$ then it would be more convenient to augment the second resources.

Let's analyze the situation when only one of the resources changes, ie, let $\epsilon=0$. If $60+\delta$ $\Longrightarrow$all RHS terms change and we must check feasibility. Which are the multipliers for the first row?$\pi_1=\frac{1}{5},\pi_2=-\frac{1}{4},\pi_3=1$.

I: $1/5(60+\delta)-1/4\cdot 40+1\cdot 0=12+\delta/5-10=2+\delta/5$

II: $-1/5(60+\delta)+1/2\cdot 40+1\cdot 0=-60/5+20-\delta/5=8-1/5\delta$

Risk that the RHS becomes negative. Eg: if $\delta=-20$ then the tableau stays optimal but not feasible. We need to apply the dual simplex and the increase in the objective value would therefore be less than what prospected by the marginal values. In Figure 3.4, we plot the objective value as a function of the increase δ.

Example 3.5. (III) Introduction of a new variable

$$\begin{array}{rrrrrrr} \max & 5x_0 & + & 6x_1 & + & 8x_2 & \\ & 6x_0 & + & 5x_1 & + & 10x_2 & \leq & 60 \\ & 8x_0 & + & 4x_1 & + & 4x_2 & \leq & 40 \\ & & & & & x_0, x_1, x_2 & \geq & 0 \end{array}$$

What will be the reduced cost of x_0 in the final tableau?

$$c_j + \sum \pi_i a_{ij} = +1 \cdot 5 - \frac{2}{5} \cdot 6 + (-1)8 = -\frac{27}{5}$$

To make the variable worth entering in basis:

- increase its profit
- decrease the amount in constraint II: $-2/5 \cdot 6 - a_{20} + 5 > 0$

Example 3.6. (IV) Introduction of a new constraint

$$\begin{array}{rrrrrrr} \max & 6x_1 & + & 8x_2 & & \\ & 5x_1 & + & 10x_2 & \leq & 60 \\ & 4x_1 & + & 4x_2 & \leq & 40 \\ & 5x_1 & + & 6x_2 & \leq & 50 \\ & & & x_1, x_2 & \geq & 0 \end{array}$$

Final tableau not in canonical form, need to iterate

	x_1	x_2	x_3	x_4	x_5	$-z$	b
x_2	0	1	$1/5$	$-1/4$		0	2
x_1	1	0	$-1/5$	$1/2$		0	8
	5	6	0	0	1	0	50
	0	0	$-2/5$	-1	0	1	-64

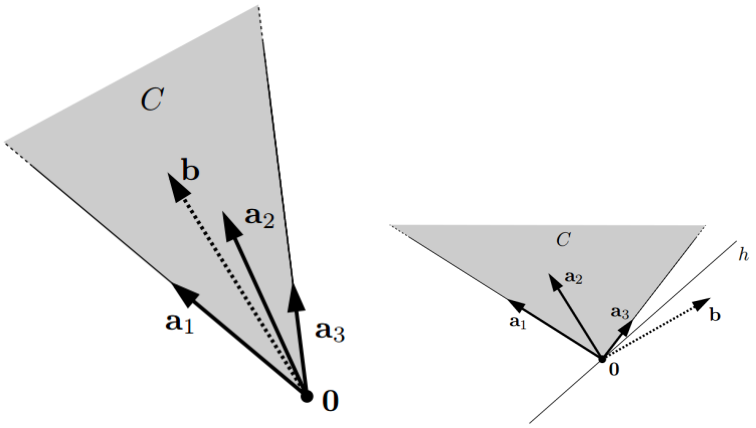
After bringing it in canonical form:

	x_1	x_2	x_3	x_4	x_5	$-z$	b
x_2	0	1	$1/5$	$-1/4$		0	2
x_1	1	0	$-1/5$	$1/2$		0	8
	0	0	$-1/5$	-1	1	0	-2
	0	0	$-2/5$	-1	0	1	-64

Example 3.7. (V) Change in a technological coefficient:

	x_1	x_2	x_3	x_4	$-z$	b
x_3	5	$10+\delta$	1	0	0	60
x_4	4	4	0	1	0	40
	6	8	0	0	1	0

- first effect on its column
- then look at c
- finally look at b



	x_1	x_2	x_3	x_4	$-z$	b
x_2	0	$(10+\delta)1/5+4(-1/4)$	$1/5$	$-1/4$	0	2
x_1	1	$(10+\delta)(-1/5)+4(1/2)$	$-1/5$	$1/2$	0	8
	0	$-2/5\delta$	$-2/5$	-1	1	-64

The dominant application of LP is mixed integer linear programming. In this context it is extremely important being able to begin with a model instantiated in one form followed by a sequence of problem modifications (such as row and column additions and deletions and variable fixings) interspersed with resolves

3.6 Farkas Lemma

Farkas Lemma gives us another proof of the strong duality theorem. Moreover, it provides a way to certificate the infeasibility of an LP instance.

Lemma 1 (Farkas). Let $A \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$. Then,

$$\begin{aligned} \text{either} \quad & I. && \exists \mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0} \\ \text{or} \quad & II. && \exists \mathbf{y} \in \mathbb{R}^m : \mathbf{y}^T A \geq 0^T \text{ and } \mathbf{y}^T\mathbf{b} < \mathbf{0} \end{aligned}$$

Easy to see that both I and II cannot occur together:

$$(0 \leq) \qquad \mathbf{y}^T A\mathbf{x} = \mathbf{y}^T\mathbf{b} \qquad (< 0)$$

Geometric interpretation of Farkas Lemma Linear combination of $\mathbf{a}_i$ with nonnegative terms generates a **convex cone**:

$$\{\lambda_1\mathbf{a}_1 + \ldots + \lambda_n\mathbf{a}_n, \mid \lambda_1, \ldots, \lambda_n \geq \mathbf{0}\}$$

Polyhedral cone: $C = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{0}\}$, intersection of many $\mathbf{ax} \leq 0$. Convex hull of rays $\mathbf{p}_i = \{\lambda_i\mathbf{a}_i, \lambda_i \geq 0\}$

Either point $\mathbf{b}$ lies in convex cone C

or $\exists$ hyperplane h passing through point 0 $h = \{\mathbf{x} \in \mathbb{R}^m : \mathbf{y}^T\mathbf{x} = 0\}$ for $\mathbf{y} \in \mathbb{R}^m$ such that all vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n$ (and thus C) lie on one side and $\mathbf{b}$ lies (strictly) on the other side (ie, $\mathbf{y}^T\mathbf{a}_i \geq 0, \forall i = 1 \ldots n$ and $\mathbf{y}^T\mathbf{b} < 0$).

Variants of Farkas Lemma

Corollary. *(i)* $A\mathbf{x} = \mathbf{b}$ *has sol* $\mathbf{x} \geq \mathbf{0} \iff \forall \mathbf{y} \in \mathbb{R}^m$ *with* $\mathbf{y}^T A \geq \mathbf{0}^T, \mathbf{y}^T \mathbf{b} \geq \mathbf{0}$

(ii) $A\mathbf{x} \leq \mathbf{b}$ *has sol* $\mathbf{x} \geq \mathbf{0} \iff \forall \mathbf{y} \geq \mathbf{0}$ *with* $\mathbf{y}^T A \geq \mathbf{0}^T, \mathbf{y}^T \mathbf{b} \geq \mathbf{0}$

(iii) $A\mathbf{x} \leq \mathbf{0}$ *has sol* $\mathbf{x} \in \mathbb{R}^n \iff \forall \mathbf{y} \geq \mathbf{0}$ *with* $\mathbf{y}^T A = \mathbf{0}^T, \mathbf{y}^T \mathbf{b} \geq \mathbf{0}$

Proof. We show only that i) $\implies$ ii) since it will be useful in our proof of the duality theorem.
$\bar{A} = [A \mid I_m]$
$A\mathbf{x} \leq \mathbf{b}$ has sol $\mathbf{x} \geq \mathbf{0} \iff \bar{A}\bar{\mathbf{x}} = \mathbf{b}$ has sol $\bar{\mathbf{x}} \geq \mathbf{0}$
By (i):

$$\forall \mathbf{y} \in \mathbb{R}^m$$
$$\mathbf{y}^T \mathbf{b} \geq \mathbf{0},\ \mathbf{y}^T \bar{A} \geq \mathbf{0}$$

which implies:

$$\mathbf{y}^T A \geq \mathbf{0}$$
$$\mathbf{y} \geq \mathbf{0}$$

□

Strong Duality by Farkas Lemma

$$(P) \qquad \max\{\mathbf{c}^T \mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$$

Assume P has opt sol $\mathbf{x}^*$ with value z^*. We find that D has opt sol as well and its value coincide with z^*.

Opt value for P:

$$\gamma = \mathbf{c}^T \mathbf{x}^*$$

We know by assumption:

$\begin{matrix} A\mathbf{x} \leq \mathbf{b} \\ \mathbf{c}^T \mathbf{x} \geq \gamma \end{matrix}$ has sol $\mathbf{x} \geq \mathbf{0}$

and $\forall \epsilon > 0$
$\begin{matrix} A\mathbf{x} \leq \mathbf{b} \\ \mathbf{c}^T \mathbf{x} \geq \gamma + \epsilon \end{matrix}$ has no sol $x \geq \mathbf{0}$

Let's define:

$$\hat{A} = \begin{bmatrix} A \\ -\mathbf{c}^T \end{bmatrix} \qquad \hat{\mathbf{b}} = \begin{bmatrix} \mathbf{b} \\ -\gamma - \epsilon \end{bmatrix}$$

and consider $\hat{A}\mathbf{x} \leq \hat{\mathbf{b}}_0$ and $\hat{A}\mathbf{x} \leq \hat{\mathbf{b}}_\epsilon$
We apply variant (ii) of Farkas' Lemma:

For $\epsilon = 0$, $\hat{A}\mathbf{x} \leq \hat{\mathbf{b}}_0$ has sol $\mathbf{x} \geq \mathbf{0}$
is equivalent to:
there exists $\hat{\mathbf{y}}^T = [\mathbf{u}, z] \in \mathbb{R}^{m+1}$,

$$\hat{\mathbf{y}} \geq \mathbf{0}$$
$$\hat{\mathbf{y}}^T \hat{A} \geq \mathbf{0}$$
$$\hat{\mathbf{y}}^T \mathbf{b}_0 \geq 0$$

Then

$$A^T \mathbf{u} \geq z\mathbf{c}$$
$$\mathbf{b}^T \mathbf{u} \geq z\gamma$$

For $\epsilon > 0$, $\hat{A}\mathbf{x} \leq \hat{\mathbf{b}}_\epsilon$ has no sol $\mathbf{x} \geq \mathbf{0}$
is equivalent to:
there exists $\hat{\mathbf{y}}^T = [\mathbf{u}, z] \in \mathbb{R}^{m+1}$,

$$\hat{\mathbf{y}} \geq \mathbf{0}$$
$$\hat{\mathbf{y}}^T \hat{A} \geq \mathbf{0}$$
$$\hat{\mathbf{y}}^T \mathbf{b}_\epsilon < 0$$

Then

$$A^T \mathbf{u} \geq z\mathbf{c}$$
$$\mathbf{b}^T \mathbf{u} < z(\gamma + \epsilon)$$

Hence, $z > 0$ or $z = 0$ would contradict the separation of cases.

We can set $\mathbf{v} = \frac{1}{z}\mathbf{u} \geq 0$

$$A^T\mathbf{v} \geq \mathbf{c}$$
$$\mathbf{b}^T\mathbf{v} < \gamma + \epsilon$$

By weak duality γ is lower bound for D. Since D bounded and feasible then there exists $\mathbf{y}^*$:

$$\gamma \leq \mathbf{b}^T\mathbf{y}^* < \gamma + \epsilon \qquad \forall \epsilon > 0$$

$\mathbf{v}$ is feasible sol of D with objective value $< \gamma + \epsilon$ which implies $\mathbf{b}^T\mathbf{y}^* = \gamma$

Certificate of Infeasibility Farkas Lemma provides a way to certificate infeasibility.

Theorem 4. Given a certificate $\mathbf{y}^*$ it is easy to check the conditions (by linear algebra):

$$\begin{aligned} A^T\mathbf{y}^* &\geq \mathbf{0} \\ \mathbf{b}\mathbf{y}^* &< 0 \end{aligned}$$

Why would $\mathbf{y}^*$ be a certificate of infeasibility?

Proof. (By contradiction)
Assume, $A^T\mathbf{y}^* \geq \mathbf{0}$ and $\mathbf{b}\mathbf{y}^* < 0$.
Moreover assume $\exists \mathbf{x}^*$: $A\mathbf{x}^* = \mathbf{b}$, $\mathbf{x}^* \geq \mathbf{0}$,then:

$$(\geq 0) \;\; (\mathbf{y}^*)^T A\mathbf{x}^* = (\mathbf{y}^*)^T\mathbf{b} \;\; (< 0)$$

Contradiction □

General form:

$$\begin{aligned} \max \;\; & c^T x \\ A_1 x &= b_1 \\ A_2 x &\leq b_2 \\ A_3 x &\geq b_3 \\ x &\geq 0 \end{aligned}$$

infeasible $\Leftrightarrow \exists y^*$

$$\begin{aligned} b_1^T y_1 + b_2^T y_2 + b_3^T y_3 &> 0 \\ A_1^T y_1 + A_2^T y_2 + A_3^T y_3 &\leq 0 \\ y_2 &\leq 0 \\ y_3 &\geq 0 \end{aligned}$$

Example 3.8.

$$\begin{aligned} \max \;\; & c^T x \\ x_1 &\leq 1 \\ x_1 &\geq 2 \end{aligned}$$

$$\begin{aligned} b_1^T y_1 + b_2^T y_2 &> 0 \\ A_1^T y_1 + A_2^T y_2 &\leq 0 \\ y_1 &\leq 0 \\ y_2 &\geq 0 \end{aligned}$$

$$\begin{aligned} y_1 + 2y_2 &> 0 \\ y_1 + y_2 &\leq 0 \\ y_1 &\leq 0 \\ y_2 &\geq 0 \end{aligned}$$

$y_1 = -1, y_2 = 1$ is a valid certificate.

Note that the Farkas' infeasibility certificate is not unique! It can be reported in place of the dual solution because they have the same dimension. To repair infeasibility we should change the primal at least so much so that the certificate of infeasibility is no longer valid. Only constraints with $y_i \neq 0$ in the certificate of infeasibility cause infeasibility.

3.7 Summary

In this chapter we have presented the following topics regarding LP duality:

- Derivation:
  1. bounding
  2. multipliers
  3. recipe
  4. Lagrangian
- Theory:
  - Symmetry
  - Weak duality theorem
  - Strong duality theorem
  - Complementary slackness theorem
  - Farkas Lemma: Strong duality + Infeasibility certificate
- Dual Simplex
- Economic interpretation
- Geometric Interpretation
- Sensitivity analysis

The main advantages of considering the dual formulation are:

- proving optimality (although the simplex tableau can already do that)
- checking the correctness of results easily
- attaining an alternative solution method (ie, primal simplex on dual)
- making analysis of sensitivity with respect to the parameters.
- solving P or D we solve the other for free
- attaining a certificate of infeasibility

Chapter 4

Revised Simplex Method

The running time of the simplex algorithms depends on the total number of pivot operations and by the cost of each single operation. We have already discussed in the previous chapter what is known about the number of iterations. Let's now focus on the cost of a single iteration.

The complexity of a single pivot operation in the standard simplex is determined by:

- entering variable: $O(n)$
- leaving variable: $O(m)$
- updating the tableau: $O(mn)$

Hence, the most costly operation in the simplex is updating the tableau in the pivot operation.

We can observe that we are doing operations that are not actually needed. For example, in the tableau the only columns that really matters is the one of the entering variable. Moreover, we have space issues: we need to store the whole tableau, that is, $O(mn)$ floating point numbers, this can become a lot: for 1000 constraints and 50000 variables in double precision floating point storing the whole tableau yields 400MB. Further, most problems have sparse matrices (they contain many zeros). Sparse matrices are typically handled efficiently by special storing ways and specialized operators. Instead, the standard simplex immediately disrupts sparsity. The problem with an iterated method like the simplex is that floating point errors accumulate and may become very important.

There are several ways to improve the efficiency of the pivot operation. To gain a better insight we need a matrix description of simplex. As (we have mostly tried) in the previous chapter, all vectors are column vectors and denoted by lowercase letters in bold face. Matrices are denoted in upper case letters.

We consider a general LP problem in standard form

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j \\ & \sum_{j=1}^{n} a_{ij} x_j \;\leq\; b_i \quad i = 1..m \\ & x_j \;\geq\; 0 \quad j = 1..n \end{aligned}$$

After the introduction of the slack variables $x_{n+1}, x_{n+2}, \ldots, x_{n+m}$ the problem can be written in vector form as

$$\begin{aligned} \max \; \mathbf{c}^T\mathbf{x} & \\ A\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

or

$$\max\{\mathbf{c}^T\mathbf{x} \mid A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$$

where $\mathbf{x} \in \mathbb{R}^{n+m}$, $A \in \mathbb{R}^{m\times(n+m)}$, $\mathbf{c} \in \mathbb{R}^{n+m}$, and $\mathbf{b} \in \mathbb{R}^m$.

We aim at understanding the relationship between each tableau and the initial data. We saw that every tableau corresponds to a basic feasible solution (the reverse is not true because of the possible degenerancies). For each basic feasible solution:

- $B = \{1 \dots m\}$ is denoted as the basis
- $N = \{m+1 \dots m+n\}$ is the set of indices of the variables not in basis
- $A_B = [\mathbf{a}_1 \dots \mathbf{a}_m]$ is the basis matrix (or basis)
- $A_N = [\mathbf{a}_{m+1} \dots \mathbf{a}_{m+n}]$ is the remaining matrix after removal of A_B from A.

Moreover,

- $\mathbf{x}_N = 0$
- $\mathbf{x}_B \geq 0$

We can represent this information in tableau form:

$$\left[\begin{array}{cc|c|c} & & & \\ A_N & A_B & \mathbf{0} & \mathbf{b} \\ & & & \\ \hline \mathbf{c}_N^T & \mathbf{c}_B^T & 1 & 0 \end{array}\right]$$

We can now rewrite $A\mathbf{x} = \mathbf{b}$ as:

$$A\mathbf{x} = A_N\mathbf{x}_N + A_B\mathbf{x}_B = \mathbf{b} \tag{4.1}$$

$$A_B\mathbf{x}_B = \mathbf{b} - A_N\mathbf{x}_N \tag{4.2}$$

Theorem 4.1. *Unique basic feasible solution $\iff$ A_B is non-singular (ie, the rows are linearly independent and* $\det(A) \neq 0$*).*

Proof. We have already shown previously that if A_B is non-singular then there is a unique basic feasible solution given by $\mathbf{x}_B = A^{-1}\mathbf{b}$. Now we set out to prove that if $\mathbf{x}$ is a basic feasible solution for B then A_B is non-singular. Since a basic feasible solution $\mathbf{x}$ satisfies $A\mathbf{x} = \mathbf{b}$ and $\mathbf{x}_N = \mathbf{0}$ then it satisfies $A_B\mathbf{x}_B = \mathbf{b} - A_N\mathbf{x}_N = \mathbf{b}$. Hence, $A_B\mathbf{x}_B = \mathbf{b}$. From linear algebra we know that if $\mathbf{x}_B$ is unique then the matrix of the system $A_B\mathbf{x}_B = \mathbf{b}$ must be full rank, or, equivalently, non-singular. To verify that there are no other basic feasible solutions for B, consider an arbitrary vector $\tilde{\mathbf{x}}$ such that $A_B\tilde{\mathbf{x}} = \mathbf{b}$ and $\tilde{\mathbf{x}}_N = \mathbf{0}$. Since the resulting vector satisfies $A\tilde{\mathbf{x}} = A_B\tilde{\mathbf{x}}_B + A_N\tilde{\mathbf{x}}_N = \mathbf{b}$, it must satisfy the top m equations in the tableau representing $\mathbf{x}$. But then $\tilde{\mathbf{x}}_N = \mathbf{0}$ implying $\tilde{\mathbf{x}} = \mathbf{x}_B$. □

Hence, having shown that A_B is non-singular, we can write from (4.1):

$$\mathbf{x}_B = A_B^{-1}\mathbf{b} - A_B^{-1}A_N\mathbf{x}_N \tag{4.3}$$

and for the objective function:

$$z = \mathbf{c}^T\mathbf{x} = \mathbf{c}_B^T\mathbf{x}_B + \mathbf{c}_N^T\mathbf{x}_N.$$

Substituting for $\mathbf{x}_B$ from (4.3):

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

and collecting together:

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

which is the dictionary corresponding to a basis B. In tableau form, for a basic feasible solution B we have:

$$\left[\begin{array}{c|c|c|c} A_B^{-1}A_N & I & \mathbf{0} & A_B^{-1}\mathbf{b} \\ \hline \mathbf{c}_N^T - \mathbf{c}_B^TA_B^{-1}A_N & 0 & 1 & -\mathbf{c}_B^TA_B^{-1}\mathbf{b} \end{array}\right]$$

The identity matrix I of size $n \times n$ occupies the columns of the variables that are in basis. The other terms of the matrix A are given by$\bar{A} = A_B^{-1}A_N$.

The cost of one iteration of the revised simplex in a trivial implementation is determined by the matrix operations needed to write the values in the tableau. These operations are:

- Inverting the matrix A_B: in $O(m^3)$ by Gauss-Jordan
- Compute $A_B^{-1}A_N$: $O(m^2n)$
- Compute $A_B^{-1}\mathbf{b}$: $O(m^2)$

The overall complexity is $O(m^2(m+n))$. This is apparently more than the standard simplex, however smart implementations can be more efficient. The most important gain can be achieved by noting that at each iteration of the simplex we do not need to compute all elements of $\bar{A}$.

Let's see this with an example.

Example 4.1. The LP problem is given on the left. Its equational standard form is derived on the right:

$$\begin{array}{rrcl} \max & x_1 + x_2 & & \\ & -x_1 + x_2 & \leq & 1 \\ & x_1 & \leq & 3 \\ & x_2 & \leq & 2 \\ & x_1, x_2 & \geq & 0 \end{array}$$

$$\begin{array}{rrcl} \max & x_1 + x_2 & & \\ & -x_1 + x_2 + x_3 & = & 1 \\ & x_1 + x_4 & = & 3 \\ & x_2 + x_5 & = & 2 \\ & x_1, x_2, x_3, x_4, x_5 & \geq & 0 \end{array}$$

The initial tableau is:

$x1$	$x2$	$x3$	$x4$	$x5$	$-z$	b
-1	1	1	0	0	0	1
1	0	0	1	0	0	3
0	1	0	0	1	0	2
1	1	0	0	0	1	0

After two iterations this is how the tableau looks like:

$x1$	$x2$	$x3$	$x4$	$x5$	$-z$	b
1	0	−1	0	1	0	1
0	1	0	0	1	0	2
0	0	1	1	−1	0	2
0	0	1	0	−2	1	−3

The basic variables are x_1, x_2, x_4 and the non basic ones: x_3, x_5. With this information we can write, looking at the **initial tableau**:

$$A_B = \begin{bmatrix} -1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad A_N = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \quad x_B = \begin{bmatrix} x_1 \\ x_2 \\ x_4 \end{bmatrix} \quad x_N = \begin{bmatrix} x_3 \\ x_5 \end{bmatrix}$$

$$c_B^T = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \quad c_N^T = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

The tableau is not optimal hence we proceed to the next pivot operation. We describe the operations of selecting a variable to enter the basis, one to leave the basis and updating the tableau in terms of matrix calculations.

Entering variable : In the standard simplex we look at the reduced costs in the tableau. In the revised simplex we need to calculate: $\mathbf{c}_N^T - \mathbf{c}_B^T A_B^{-1} A_N$. This is decomposed into two steps:

Step 1. Find $\mathbf{y}^T$ by solving the linear system $\mathbf{y}^T A_B = \mathbf{c}_B^T$. It is possible to calculate $\mathbf{y}^T = \mathbf{c}_B^T A_B^{-1}$ but the system can be solved more efficiently without calculating the inverse of A_B.

Step 2. Calculate $\mathbf{c}_N^T - \mathbf{y}^T A_N$ (each term $\mathbf{c}_j - \mathbf{y}^T \mathbf{a}_j$ can be calculated independently and for some pivoting rules one does not need to calculate them all).

Let's carry out these two steps in our example:

Step 1: we solve the linear system $\mathbf{y}^T A_B = \mathbf{c}_B^T$, which looks like:

$$\begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} -1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}$$

via $c_B^T A_B^{-1} = \mathbf{y}^T$:

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

Step 2: we calculate $\mathbf{c}_N^T - \mathbf{y}^T A_N$

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

The first element is the vector is positive and the calculation can stop. It corresponds to the variable x_3, which therefore has a positive reduced cost and is selected to enter in the basis.

Leaving variable In the standard simplex we need now to determine the largest feasible amount θ to increase the entering variable without producing an infeasible solution. We do the constraint analysis, looking at the tableau and knowing that x_5 will remain non-basic and hence zero:

$$\begin{aligned} &\text{R1: } x_1 - x_3 + x_5 = 1 && x_1 = 1 + x_3 \geq 0 \\ &\text{R2: } x_2 + x_5 = 2 && x_2 = 2 \\ &\text{R3: } x_3 + x_4 - x_5 = 2 && x_4 = 2 - x_3 \geq 0 \end{aligned}$$

Hence, the first and the second constraints do not pose any limit to the increase of x_3. The third constraint is the most restrictive. It determines that the largest increase θ is 2.

Translating these operations in matrix operations we observe that they can be expressed as:

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

where $\mathbf{x}'$ is the current solution and $\mathbf{x}$ the adjacent solution to which we are moving. Since in the new solution only one non basic variable changes its value, the selected one x_3, then not all the terms of $A_B^{-1} A_N \mathbf{x}_N$ need to be calculated but only those that correspond to the entering variable. Let denote by $\mathbf{d}$ the column of $A_B^{-1} A_N$ and by $\mathbf{a}$ the column of A_N in the initial tableau, that correspond to this variable. We have

$$\mathbf{d} = A_B^{-1} \mathbf{a}$$

and we calculate the update simply by

$$\mathbf{x}_B = \mathbf{x}'_B - \mathbf{d}\theta$$

We can thus describe the calculation that we have to carry out in the revised simplex to find θ such that $\mathbf{x}_B$ stays positive.

Step 3. Find $\mathbf{d}$ by solving $A_B \mathbf{d} = \mathbf{a}$. It is possible to calculate $\mathbf{d} = A_B^{-1}\mathbf{a}$ but the system can be solved more efficiently without calculating the inverse of A_B.

Step 4. Determine the largest θ such that $\mathbf{x}_B = \mathbf{x}'_B - \mathbf{d}\theta \geq \mathbf{0}$. If there is no such θ, then the problem is unbounded. Otherwise, at least one component of $\mathbf{x}'_B - \mathbf{d}\theta$ equals zero and the corresponding variable is leaving the basis.

In our numerical example, these two steps yield the following:

Step 3:

$$\begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \implies \mathbf{d} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

Step 4:

$$\mathbf{x}_B = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} - \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \theta \geq \mathbf{0}$$

The first two terms do not pose any limit, while for the third term it must be $2 - \theta \geq 0$, which implies $\theta \leq 2$ Hence, it is x_4 that goes to zero and thus leaves the basis.

Updating the tableau This part is the computationally heaviest part of the standard simplex. In the revised simplex, instead, this step comes for free.

Step 5. The update of x_B is done by setting the value found for θ in $\mathbf{x}'_B - \mathbf{d}\theta \geq \mathbf{0}$ and replacing $\mathbf{x}'_B$ with $\mathbf{x}_B$. A_B is updated by replacing the leaving column by the entering column.

In our example this step yields:

Step 5.

$$x'_B = \begin{bmatrix} x_1 - d_1\theta \\ x_2 - d_2\theta \\ \theta \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 2 \end{bmatrix} \qquad A_B = \begin{bmatrix} -1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

Incidentally, note that the order of the columns of A_B is not important as long as it matches the order of the components of $\mathbf{x}_B$. Hence, the next iteration could just as well be entered with

$$x'_B = \begin{bmatrix} 2 \\ 3 \\ 2 \end{bmatrix} \qquad A_B = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

The *basis heading* is an ordered list of basic variables that specifies the actual order of the m columns of A_B. For simplicity, during the solution process the basis heading is updated by replacing the leaving variable with the entering variable.

The revised simplex allows to save many operations especially if there are many variables! Also in terms of space the revised simplex is convenient, note indeed that we do not need to store the matrix A_B but a vector containing the basis heading is enough. Special ways to call the matrix A from memory help to provide then further speeds up. Finally, the revised simplex provides a better control over numerical issues since A_B^{-1} can be recomputed at the end once the variables that are in basis are known.

There are different implementations of the revised simplex, depending on how $\mathbf{y}^T A_B = \mathbf{c}_B^T$ and $A_B\mathbf{d} = \mathbf{a}$ are solved. They are in fact solved from scratch every time. The next section provides the general idea behind these implementations.

4.1 Efficiency Issues

We saw that computing the inverse of a matrix is a costly operation and likely to introduce numerical inaccuracies. If you do not remember the details, go back to your notes from Linear Algebra. Hence, the two linear systems that must be solved in the revised simplex, $\mathbf{y}^T A_B = \mathbf{c}^T$ and $A_B\mathbf{d} = \mathbf{a}$ are solved without computing A_B^{-1}.

Eta Factorization of the Basis Let $A_B = B$ and let's consider the kth iteration. The matrix B_k will differ from the matrix B_{k-1} by the column p. The column p is the $\mathbf{a}$ column appearing in $B_{k-1}\mathbf{d} = \mathbf{a}$ solved in Step 3. Hence:

$$B_k = B_{k-1}E_k$$

where E_k is the **eta matrix**[1] differing from the identity matrix only in one column:

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

No matter how we solve $\mathbf{y}^T B_{k-1} = \mathbf{c}_B^T$ and $B_{k-1}\mathbf{d} = \mathbf{a}$, their update always relays on $B_k = B_{k-1}E_k$ with E_k available. Moreover, when the initial basis is made of slack variables then $B_0 = I$ and $B_1 = E_1, B_2 = E_1E_2, \ldots$:

$$B_k = E_1E_2 \ldots E_k \qquad \text{eta factorization}$$

$$\begin{aligned} ((((\mathbf{y}^T E_1)E_2)E_3)\cdots)E_k = \mathbf{c}_B^T, \qquad & \mathbf{u}^T E_4 = \mathbf{c}_B^T,\ \mathbf{v}^T E_3 = \mathbf{u}^T,\ \mathbf{w}^T E_2 = \mathbf{v}^T,\ \mathbf{y}^T E_1 = \mathbf{w}^T \\ (E_1(E_2 \cdots E_k\mathbf{d})) = \mathbf{a}, \qquad & E_1\mathbf{u} = \mathbf{a},\ E_2\mathbf{v} = \mathbf{u},\ E_3\mathbf{w} = \mathbf{v},\ E_4\mathbf{d} = \mathbf{w} \end{aligned}$$

When $B_0 \neq I$:

$$B_k = B_0E_1E_2 \ldots E_k \qquad \text{eta factorization}$$

$$((((\mathbf{y}^T B_0)E_1)E_2)\cdots)E_k = \mathbf{c}_B^T$$

$$(B_0(E_1 \cdots E_k\mathbf{d})) = \mathbf{a}$$

In this case, it helps having the LU factorization of B_0.

LU Factorization To solve the system $A\mathbf{x} = \mathbf{b}$ by Gaussian Elimination we put the A matrix in row echelon form by means of elementary row operations. Each row operation corresponds to multipling left and right side by a lower triangular matrix L and a permutation matrix P. Hence, the method throughout its iterations is equivalent to:

$$\begin{aligned} A\mathbf{x} &= \mathbf{b} \\ L_1P_1A\mathbf{x} &= L_1P_1\mathbf{b} \\ L_2P_2L_1P_1A\mathbf{x} &= L_2P_2L_1P_1\mathbf{b} \\ &\vdots \\ L_mP_m \ldots L_2P_2L_1P_1A\mathbf{x} &= L_mP_m \ldots L_2P_2L_1P_1\mathbf{b} \end{aligned}$$

thus

$$U = L_mP_m \ldots L_2P_2L_1P_1A \qquad \text{triangular factorization of } A$$

where U is an upper triangular matrix whose entries in the diagonal are ones (if A is nonsingular such triangularization is unique).

For a square matrix A the LU decomposition is given by:

$$\mathbf{A} = \mathbf{LU}$$
$$\mathbf{PA} = \mathbf{LU}$$

From an LU decomposition it is easy to solve $A\mathbf{x} = LU\mathbf{x} = \mathbf{b}$: set $\mathbf{y} = U\mathbf{x}$ then

1. $L\mathbf{y} = \mathbf{b}$ can be solved easily by forward substitution
2. $U\mathbf{x} = \mathbf{y}$ can be solved easily by backward substitution.

[1] Eta matrices are used to represent elementary row operations in the Gaussian elmination method as a product between matrices, an eta matrix and the matrix to put in row echelon form.

We can compute the triangular factorization of B_0 before the initial iterations of the simplex:

$$L_m P_m \dots L_2 P_2 L_1 P_1 B_0 = U$$

We can then rewrite U as

$$U = U_m U_{m-1} \dots, U_1$$

Hence, for $B_k = B_0 E_1 E_2 \dots E_k$:

$$L_m P_m \dots L_2 P_2 L_1 P_1 B_k = U_m U_{m-1} \dots U_1 E_1 E_2 \cdots E_k$$

Then $\mathbf{y}^T B_k = \mathbf{c}_B^T$ can be solved by first solving:

$$((((\mathbf{y}^T U_m) U_{m-1}) \cdots) E_k = \mathbf{c}_B^T$$

and then replacing $\mathbf{y}^T$ by $((\mathbf{y}^T L_m P_m) \cdots) L_1 P_1$. To show this, we can express

$$B_k = \underbrace{(L_m P_m \cdots L_1 P_1)}_{L}{}^{-1} \underbrace{U_m \cdots E_k}_{U}$$

and in this notation

$$\mathbf{y}^T L^{-1} U = \mathbf{c}_B^T$$
$$\mathbf{w}^T U = \mathbf{c}_B^T$$
$$\mathbf{w}^T = \mathbf{y}^T L^{-1} \implies \mathbf{y}^T = L \mathbf{w}^T.$$

In the literature, solving $\mathbf{y}^T B_k = \mathbf{c}_B^T$ is also called backward transformation (BTRAN) while solving $B_k \mathbf{d} = \mathbf{a}$ is also called forward transformation (FTRAN).

Some further remarks:

- The E_k matrices can be stored by only storing the column and the position
- If sparse columns then they can be stored in compact mode, ie only nonzero values and their indices
- The triangular eta matrices L_j, U_j can also be stored efficiently due to their sparsity.
- To store P_j just two indices are needed

Efficient Implementations

- Dual simplex with steepest descent
- Linear Algebra:
  - Dynamic LU-factorization using Markowitz threshold pivoting (Suhl and Suhl, 1990)
  - Sparse linear systems: Typically these systems take as input a vector with a very small number of nonzero entries and output a vector with only a few additional nonzeros.
- Presolve, ie problem reductions: removal of redundant constraints, fixed variables, and other extraneous model elements.
- Dealing with degeneracy, stalling (long sequences of degenerate pivots), and cycling:
  - bound-shifting (Paula Harris, 1974)
  - Hybrid Pricing (variable selection): start with partial pricing, then switch to devex (approximate steepest-edge, Harris, 1974)
    In Gurobi: the parameter `SimplexPricing` determines the simplex variable pricing strategy: the available options are Automatic, Partial Pricing, Steepest Edge, Devex, and Quick-Start Steepest Edge.
- A model that might have taken a year to solve 10 years ago can now be solved in less than 30 seconds (Bixby, 2002).

4.2 More on Polyhedra

Basic Geometric Facts From Linear Algebra we know that in 2D we need 2 lines to intersect to give us a point. In 3D, we need three planes. In 4D, we need 4 hyperplanes to intersect to give a point. In 2D, lines can also be parallel or overlapping and hence they do not necessarily intersect in a point. In 3D planes can also be intersecting in a line or not intersecting at all if they are parallel. Similarly in 4D, 4 hyperplanes do not necessarily intersect in a point.

On the other hand, like a point in 3D can be described by more than 3 intersecting planes, think of the top vertex of a pyramid, similarly a point in 4D can be described by more than 4 hyperplanes.

These considerations can be generalized to n dimensions. We cannot anymore visualize the situation but we can use abstract algebra to understand what is going on. In n dimensions we need n hyperplanes to determine a point. They uniquely identify a point when the rank of the matrix A of the linear system is n (or A is nonsingular)

Vertices of Polyhedra A **vertex** of a polyhedron in $\mathbb{R}^n$ is a **point** that is a feasible solution to the system:

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & \leq & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & \leq & b_2 \\ & & \vdots & & & & & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & \leq & b_m \end{array}$$

For a point $\bar{\mathbf{x}}$ we say that a constraint, ie an inequality of the system, is **active** or **tight** or *binding* when the left hand side $\overrightarrow{\mathbf{a}}_i\bar{\mathbf{x}}$ is equal to the right hand side b_i ($\overrightarrow{\mathbf{x}}_i$ is the ith row of the matrix A).

In a vertex of a polyhedron $A\mathbf{x} \leq \mathbf{b}$, $A \in \mathbb{R}^{m\times n}, \mathbf{x} \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}^m$ there are at least n active constraints. This implies that the rank of the matrix of active constraints is n. The viceversa is not necessarily true. A point $\bar{\mathbf{x}}$ that activates n constraints does not necessarily form a vertex of the polyhedron. Some of these points may be not feasible, ie, the intersection of the n supporting hyperplanes of the polyhedron happens outside of the polyhedron itself.

As we saw for the pyramid in 3D, a vertex can activate more than n constraints. The rank of the matrix of active constraints is still n.

If there are more constraints than variables, ie $m > n$, then we can find a subset and determine the intersecting point of the corresponding supporting hyperplanes. But what happens if there are more variables than constraints, ie $m < n$, can we have a vertex? Not necessarily. In LP we deal with this issue by adding slack variables, they make us choose arbitrarily a vertex. Consider the case of one constraint for two variables:

$$x_1 + x_2 \leq 0$$

We add x_3 and find that the two variables are not in the basis, ie, their value is 0. Geometrically, this corresponds to selecting a point in the line represented by the supporting hyperplane of the constraint.

To define a cube we need 6 constraints and there are 2^3 vertices. For an n-hypercube we need $2n$ constraints and there are 2^n constraints. This shows that the number of vertices can give rise to a combinatorial explosion. The upper bound to the number of vertices when there are m constraints and n variables, $m > n$, is given by the number of possibilities of having n active constraints, ie

$$\binom{m}{n}.$$

This is an upper bound because:

- some combinations of constraints will not define a vertex, ie, if the rows of the corresponding matrix are not independent
- some vertices may activate more than n constraints and hence the same vertex can be given by more than n constraints

Tableaux and Vertices To each tableau there is associated exactly one vertex of the feasibility region. The reverse is not always true. One vertex of the feasibility region can have more than one tableau associated. For example degenerate vertices have several tableaux associated.

$$\begin{array}{rrcl} \max & 6x_1 + 8x_2 & & \\ & 5x_1 + 10x_2 & \leq & 60 \\ & 4x_1 + 4x_2 & \leq & 40 \\ & x_1, x_2 & \geq & 0 \end{array} \qquad \begin{array}{c|cccccc} & x_1 & x_2 & x_3 & x_4 & -z & b \\ \hline x_2 & 0 & 1 & 1/5 & -1/4 & 0 & 2 \\ x_1 & 1 & 0 & -1/5 & 1/2 & 0 & 8 \\ \hline & 0 & 0 & -2/5 & -1 & 1 & -64 \end{array}$$

The slack variables $(x_3, x_4) = (0, 0)$ are non basic hence the corresponding constraints are active. Indeed, it means that no slack is needed between the left hand side and the right hand side of the constraints for which they have been introduced.

$$\begin{array}{rrcl} \max & 6x_1 + 8x_2 & & \\ & 5x_1 + 10x_2 & \leq & 60 \\ & 4x_1 + 4x_2 & \leq & 40 \\ & x_1, x_2 & \geq & 0 \end{array} \qquad \begin{array}{c|cccccc} & x_1 & x_2 & x_3 & x_4 & -z & b \\ \hline x_3 & 0 & 0 & 1 & 1/2 & 0 & 1 \\ x_1 & 1 & 1 & 0 & -1/2 & 0 & 1 \\ \hline & 0 & -2 & 0 & 1/2 & 1 & -1 \end{array}$$

Now, the non basic variables are $(x_2, x_4) = (0, 0)$. The constraints that are active are $4x_1 + 4x_2 \leq 40$ and $x_2 \geq 0$. Hence, there are still two active constraints when the non basic variables are two. If in the original space of the problem we have 3 variables and there are 6 constraints we would have 3 constraints active in the vertices. After we add the slack variables we have 6 variables in all. If any of the slack variables is positive then some constraints $x_i \geq 0$ of the original variables are active, otherwise the corresponding constraint of the original problem are active. Hence, we can generalize: the non basic variables are always n and they tell which constraints (among the original and the variable feasibility constraints $x_i \geq 0$) are active. A basic feasible solution implies a matrix of active constraints with rank n, some of which may be due to the original variables being zero. Let a tableau be associated with a solution that makes exactly $n+1$ constraints active. Then, one basic variable is zero.

Definition 6. In a polyhedron in $\mathbb{R}^n$, two vertices are **adjacent** iff:

- they have at least $n-1$ active constraints in common
- rank of common active constraints is $n-1$

In terms of tableaux, this condition means that between two adjacent vertices there are $n-1$ variables in common in the basis.

4.3 More on LP

4.3.1 LP: Rational Solutions

- A precise analysis of running time for an algorithm includes the number of bit operations together with the number of arithmetic operations.

  Example 4.2. The knapsack problem aka, budget allocation problem, that asks to choose among a set of n investments those that maximize the profit and cost in total less than B, can be solved by dynamic programming in

  $$O(n|B|)$$

  The number B needs $b = \log|B|$ bits hence the running time is exponential in the number of bits needed to represent B, ie, $O(n2^b)$

- **Weakly polynomial time algorithms** have running time that are independent of the sizes of the numbers involved in the problem and hence on the number of bits needed to represent them.
- **Strongly polynomial time algorithms**: the running time of the algorithm is independent of the number of bit operations. Eg: same running time for input numbers with 10 bits as for inputs with a million bits.

- Running time depends on the sizes of numbers. We have to restrict attention to rational instances when analyzing the running time of algorithms and assume they are coded in binary.

  Theorem 4.2 (Rational Solutions)**.** *Optimal feasible solutions to LP problems are always rational as long as all coefficient and constants are rational.*

  Proof: derives from the fact that in the simplex we only perform multiplications, divisions and sums of rational numbers

- In spite of this: No strongly polynomial-time algorithm for LP is known.

4.3.2 Interior Point Algorithms

- Ellipsoid method: cannot compete in practice but it has a weakly polynomial running time (Khachyian, 1979)
- **Interior point algorithm(s)** (Karmarkar, 1984) competitive with simplex and polynomial in some versions:
  - affine scaling algorithm (Dikin)
  - logarithmic barrier algorithm (Fiacco and McCormick) $\equiv$ Karmakar's projective method

  They operate as follows:

  1. Start at an interior point of the feasible region
  2. Move in a direction that improves the objective function value at the fastest possible rate while ensuring that the boundary is not reached
  3. Transform the feasible region to place the current point at the center of it

  Moreover:

  - because of patents reasons, now mostly known as **barrier algorithms**
  - one single iteration is computationally more intensive than the simplex (matrix calculations, sizes depend on number of variables)
  - particularly competitive in presence of many constraints (eg, for $m = 10,000$ may need less than 100 iterations)
  - bad for post-optimality analysis $\rightsquigarrow$ crossover algorithm to convert a solution of the barrier method into a basic feasible solution for the simplex

How Large Problems Can We Solve? The speed up due to algorithmic improvements has been more important than the one due to technology and machine architecture improvements (Bixby, 2007).

4.3.3 Further topics in LP

- Numerical stability and ill conditioning
- Lagrangian relaxation
- Column generation
- Decomposition methods:
  - Dantzig Wolfe decomposition
  - Benders decomposition

Very large model

	Rows	Columns	Nonzeros
Original size	5034171	7365337	25596099
After presolve	1296075	2910559	10339042

Solution times were as follows:

Very large model—solution times

	Algorithm		
Version	Barrier	Dual	Primal
CPLEX 5.0	8642.6	350000.0	71039.7
CPLEX 7.1	5642.6	6413.1	1880.0

Figure 4.1: Source: Bixby, 2002

Chapter 5

Modeling in Mixed Integer Linear Programming

Often we need to deal with integral inseparable quantities. For example, if we are modeling the presence of a bus on a line a value of 0.3 would not have a meaning in practice. Sometimes it may be enough to round fractional solutions to their nearest integers but other times rounding is not a feasible option, as it may be costly and a feasible solution is not ensured.

Discrete Optimization is the mathematical field that deals with optimization where the variables must be integers. Combinatorial optimization is also a kind of discrete optimization, although the term refers to problems that have a particular structure, like selecting subgraphs, patterns, etc.

In this chapter we will study *mixed integer linear programming* (MILP). The world is not linear but we will see that MILP constitutes a very powerful tool and that many situations, apparently non linear, can actually be modeled in linear terms. In other cases it is possible to linearize by approximation. After all "Operations Research is the art and science of obtaining bad answers to questions to which otherwise worse answers would be given."

5.1 Introduction to Integer Linear Programming

An *integer linear programming* (ILP) problem has a linear objective function, linear constraints and integer variables. A *mixed integer linear programming* (MILP) problem has a linear objective function, linear constraints and both integer and real valued variables. A *binary programming or 0–1 programming* (BIP) problem is an ILP problem where variables can take only two values, 0 and 1. *Non-linear programming* (NLP) refers to all problems that although written in mathematical programming terms may have a non linear objective function and/or non linear constraints.

Here is a non exhaustive list of mathematical programming formulations. We will not see NLP in this course.

Linear Programming (LP)

$$\begin{aligned} \max\ & \mathbf{c}^T\mathbf{x} \\ & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

Integer (Linear) Programming (ILP)

$$\begin{aligned} \max\ & \mathbf{c}^T\mathbf{x} \\ & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \\ & \mathbf{x} \text{ integer} \end{aligned}$$

Binary Integer Program (BIP)
0–1 Integer Programming

$$\begin{aligned} \max\ & \mathbf{c}^T\mathbf{x} \\ & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \in \{0,1\}^n \end{aligned}$$

Mixed Integer (Linear) Programming (MILP)

$$\begin{aligned} \max\ & \mathbf{c}^T\mathbf{x} + \mathbf{h}^T\mathbf{y} \\ & A\mathbf{x} + G\mathbf{y} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \\ & \mathbf{y} \geq \mathbf{0} \\ & \mathbf{y} \text{ integer} \end{aligned}$$

Non-linear Programming (NLP)

$$\begin{aligned} \max\ & f(\mathbf{x}) \\ & g(\mathbf{x}) \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

Recall that the sets of integers are:

- $\mathbb{Z}$ set of integers
- $\mathbb{Z}^+$ set of positive integers
- $\mathbb{Z}_0^+$ set of nonnegative integers ($\{0\} \cup \mathbb{Z}^+$)
- $\mathbb{N}_0$ set of natural numbers, ie, nonnegative integers $\{0, 1, 2, 3, 4, ...\}$

Whenever different types of variables are present, we will try to comply with the convention used in the MIPLIB 2003 and use the letters:

- x to denote binary variables
- y to denote general integer variables
- z to denote continuous variables

5.1.1 Combinatorial Optimization Problems

Definition 5.1. Combinatorial Optimization Problem (COP)
Input: Given a finite set $N = \{1, \ldots, n\}$ of objects, a weight c_j for each $j \in N$, and a collection $\mathcal{F}$ of feasible subsets of N.
Task: Find a minimum weight feasible subset, ie,

$$\min_{S \subseteq N} \left\{ \sum_{j \in S} c_j \mid S \in \mathcal{F} \right\}.$$

Note that the definition above is not an MILP formulation. However, many COP can be formulated as IP or BIP. Typically, one defines an *incidence vector* of S, $\mathbf{x}^S \in \mathbb{B}^n$ such that:

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

That is, an element i of N is selected if $x_i^S = 1$ and not selected if $x_i^S = 0$. Then, one expresses the structural constraints in function of $\mathbf{x}^S$.

5.1.2 Solution Approaches

MILP problems are solved primarily using *linear programming relaxation*. That is, relaxing the integrality constraints (the requirements that the variables are to be integer) we obtain a linear programming problem that we can solve with the simplex method or the barrier method. The solutions will be in the general case rational. Then to derive integer solutions one can use heuristics or exact methods such as branch and bound or cutting planes.

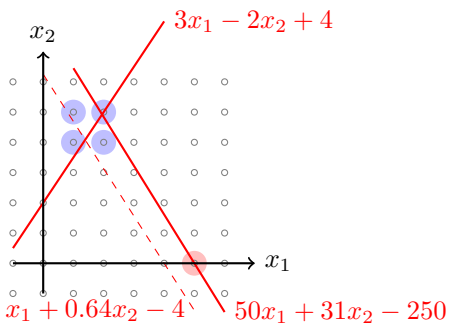


Figure 5.1: . An unfortunate situation in a rounding approach to MILP solving.

Rounding

A trivial heuristic to find integer solutions to an MILP problem is to relax the integrality constraint, solve the linear programming problem thus derived and then round up or down each single fractional value of the solution found.

Example 5.1. Let's consider the following example:

$$\begin{array}{rrrcl} \max & 100x_1 & + & 64x_2 & \\ & 50x_1 & + & 31x_2 & \leq 250 \\ & 3x_1 & - & 2x_2 & \geq -4 \\ & & & x_1, x_2 & \in \mathbb{Z}^+ \end{array}$$

The situation is represented in Figure 5.1. The feasibility region is made of the dots that represent integer solutions contained in the convex region defined by the constraints. The feasibility region is not continuous: now the optimum can be on the border (vertices) of the polytope but also *internal*.

The *linear programming relaxation* of this problem is obtained by substituting the integrality constraints on the two variables with the requirements that they must be non-negative, ie, $x_1, x_2 \geq 0$. The problem obtained is a linear programming problem that we can solve with the simplex method.

Let's denote by (ILP) the original problem and by (LPR) its linear relaxation. If the solution to (LPR) is *integer* then the (ILP) is solved. On the other hand if the solution is *rational* then we can try to round the values of the variables to their nearest integers.

The solution of (LPR) is $(376/193, 950/193)$. The situation is depicted in the figure. The circles filled in blue represent the solutions obtained by rounding down or up the values of the variables. For each rounded solution we need to test whether it is feasible. If we are in $\mathbb{R}^2$ then there are 2^2 possible roundings (up or down) of the variables and solutions to test. If we are in $\mathbb{R}^n$ then there are 2^n possible solutions. Hence, in large problems checking all possible roundings may become computationally costly. Moreover, rounding does not guarantee that a feasible solution is found and it can be arbitrarily bad with respect to the optimal solution. In our example, the optimum of (ILP) is $(5, 0)$ (the red circle in the figure), while any rounded solution is quite far from that.

There are two main techniques to solve MILP problems exactly: branch and bound and cutting planes.

Cutting Planes: sketch

Example 5.2. Suppose we have to solve the problem:

$$\begin{array}{rrrcl} \max & x_1 & + & 4x_2 & \\ & x_1 & + & 6x_2 & \leq 18 \\ & x_1 & & & \leq 3 \\ & & & x_1, x_2 & \geq 0 \\ & & & x_1, x_2 & \text{integers} \end{array}$$

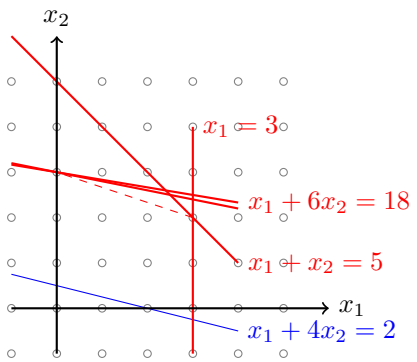


Figure 5.2: Cutting plane approach for MILP.

We solve the linear programming relaxation. The situation is depicted in Figure 5.2.The optimal solution is fractional. If we knew the constraint represented by the dashed line in the figure then solving the linear programming relaxation would give us an integer value, which would be optimal for the original problem. If we do not know that constraint we can devise a method to add constraints to our problem that would cut out the rational solution of the current linear relaxation but not cut out any integer feasible solution. This would bring us closer to the integer optimal solution. An example of such an added *cut* is given in the figure by the line just below the constraint $x_1 + 6x_2 \leq 18$. Iterating the procedure a needed number of times would eventually lead us to an integer solution.

Branch and Bound: sketch

We sketch here the branch and bound procedure and we postpone to a later chapter its full treatment.

Example 5.3. Suppose we have to solve the problem:

$$\begin{array}{rrcrcl} \max & x_1 & + & 2x_2 & & \\ & x_1 & + & 4x_2 & \leq & 8 \\ & 4x_1 & + & x_2 & \leq & 8 \\ & & & x_1, x_2 & \geq & 0, \text{integer} \end{array}$$

The branch and bound technique works by solving the linear relaxation and then branching on the fractional values. Branching corresponds to splitting the problem into two subproblems each one taking a different part of the feasibility region. The solution to the linear relaxation is cut out by restricting a fractional variable to be:

- larger than the smallest integer larger than the fractional value of the variable (floor), or
- smaller than largest integer smaller than the fractional value of the variable (ceil).

Each new subproblem is then a linear programming problem with an added constraint and we have seen in sensitivity analysis how a solution can be derived from an optimal tableau after the introduction of a constraint. When at a node the solution of the linear programming relaxation is integer, then we can stop branching on that node. The optimal solution will be the best one found at the leaves of the branch and bound tree.

The branch and bound process for our example problem is shown in Figure 5.3. Each node bears the information of the best feasible encountered in its subtree and the value of the linear relaxation, which in this case represents the best possible that can be achieved. The optimal solution has value 4 and is shown on the right hand side of the corresponding node.

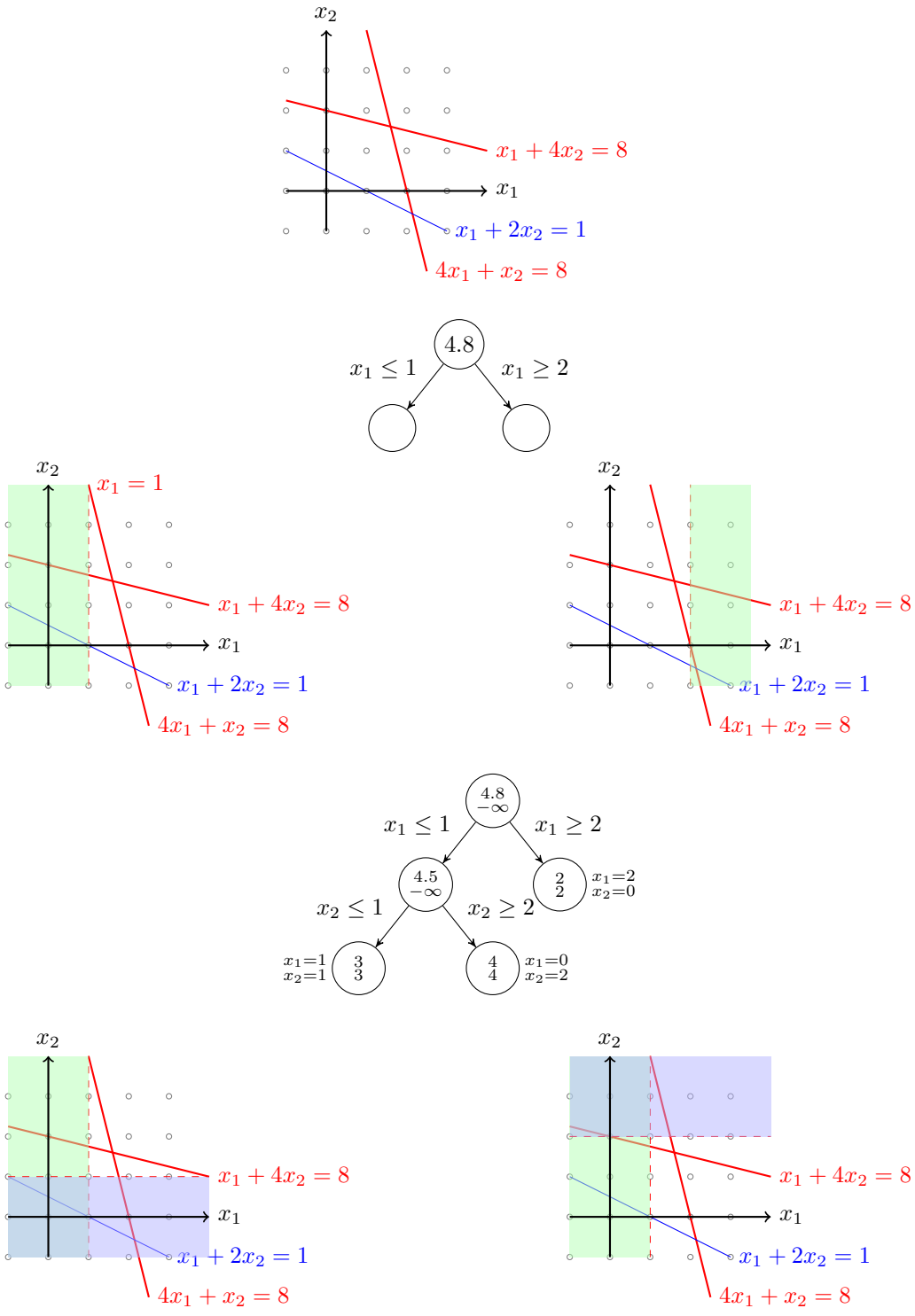


Figure 5.3: Branching in a branch and bound approach for MILP.

5.2 MILP Modeling

Find out exactly what the decision maker needs to know. For example: which investment, which product mix, which job j should a person i do. Then, define:

- the *parameters* that represent the values that are fixed and known;
- the *decision variables* that answer the questions of the decision maker. They must be of a suitable type according to the decisions they represent (continuous, integer valued, binary).

Finally, using parameters and variables, formulate mathematically:

- the **objective function** computing the benefit/cost;
- the **constraints** indicating the interplay between the different variables.

It is helpful, in order to formulate constraints, to first write down the relationship between the variables in plain words. Then, these constraints can be transformed in logical sentences using connectives such as *and, or, not, implies*. Finally, logical sentences can be converted to mathematical constraints.

Example 5.4. The decisions to take are whether to produce in a time period or not. This can be modeled by using **binary** integer variables, $x_i = 1$ or $x_i = 0$, for any period i. With these variables we can formulate the constraint in three steps:

- Plain English: "The power plant must not work in both of two neighboring time periods"
- Logical sentence: $x_i = 1$ implies $\Rightarrow x_{i+1} = 0$
- Mathematical constraint: $x_i + x_{i+1} \leq 1$

We now provide the formulation of a series of relevant problems with several real life applications.

5.2.1 Assignment Problem

Common application: *assignees* are to be assigned to perform *tasks*. Suppose we have n persons and n jobs Each person has a certain proficiency at each job. Formulate a mathematical model that can be used to find an assignment that maximizes the total proficiency.

Parameters: We use the letter I to indicate the set of persons, indexed by $i = 1..n$ and the letter J to indicate the set of jobs, indexed by $j = 1..n$. We represent the proficiency by numerical value ρ_{ij}, the higher the value is the higher the proficiency for the job.

Decision Variables: We use binary variables:

$$x_{ij} = \begin{cases} 1 \text{ if person } i \text{ is assigned job } j \\ 0 \text{ otherwise,} \end{cases} \quad \text{for } i, j = 1, 2, \ldots, n$$

Objective Function:

$$\max \quad \sum_{i=1}^{n} \sum_{j=1}^{n} \rho_{ij} x_{ij}$$

where ρ_{ij} is person i's proficiency at job j

Constraints:
Each person is assigned one job:

$$\sum_{j=1}^{n} x_{ij} = 1 \text{ for all } i \in I$$

e.g. for person 1 we get $x_{11} + x_{12} + x_{13} + \cdots + x_{1n} = 1$

Each job is assigned to one person:

$$\sum_{i=1}^{n} x_{ij} = 1 \text{ for all } j \in J$$

e.g. for job 1 we get $x_{11} + x_{21} + x_{31} + \cdots + x_{n1} = 1$.

5.2.2 Knapsack Problem

Definition 5.2 (Knapsack Problem)**.**
Input: a set of n item types, each with a value v_i and weight w_i for $i = 1, \ldots, n$.
Task: determine the number of items per type to include in a collection so that the total weight is less than a given limit, W, and the total value is as large as possible.

Without loss of generality (why?) we can assume that we can take at most one item per type. We also assume $\sum_i w_i > W$. The problem has applications in, for example, capital budgeting, project selection, etc.

Next we present a mathematical model to determine which items give the largest value.

Parameters: v_i the value (or profit) of an item; w_i the weight of item i; W the knapsack capacity.

Decision Variables:

$$x_i = \begin{cases} 1 \text{ if item } i \text{ is taken} \\ 0 \text{ otherwise,} \end{cases} \quad \text{for } i = 1, 2 \ldots, n$$

Objective Function: we want to maximize the total value of the selected items:

$$\max \quad \sum_{i=1}^{n} v_i x_i$$

Constraints: we cannot exceed knapsack capacity:

$$\sum_{i=1}^{n} w_i x_i \leq W$$

5.2.3 Set Problems

Consider the following application:

Example 5.5. Given: a set of regions, a set of possible construction locations for emergency centers, for each location a set of regions that can be served in less than 8 minutes, and the cost of installing an emergency center in that location.
Task: decide where to install a set of emergency centers such that the total cost is minimized and all regions are safely served.

For example you may think of the following numerical data:

- regions: $M = \{1, \ldots, 5\}$
- centers: $N = \{1, \ldots, 6\}$
- cost of centers: $c_j = 1$ for $j = 1, \ldots, 6$
- coverages: $S_1 = (1, 2), S_2 = (1, 3, 5), S_3 = (2, 4, 5), S_4 = (3), S_5 = (1), S_6 = (4, 5)$

We can model the problem as a combinatorial optimization problem.

Let $M = \{1, \ldots, m\}$ be the set of regions, $N = \{1, \ldots, n\}$ be the set of centers (objects) with costs $c_1, ..., c_n$, and $S_j \subseteq M$ regions serviced by $j \in N$ in 8 min. Let $\mathcal{F} \subseteq 2^N$ be a collection of sets of centers each of which would guarantee to safely serve all regions, i.e., $\bigcup_{j \in T} S_j = M$ for all $T \in \mathcal{F}$, $T \subseteq N$. The problem of finding the cheapest set of centers to safely serve all regions can be formalized as:

$$\min_{T \subseteq N} \left\{ \sum_{j \in T} c_j \mid \bigcup_{j \in T} S_j = M \right\}.$$

Let's now formulate the problem as a BIP problem.

Parameters: The set of regions M, the set of centers N, the coverage of regions for each center, S_j, the cost of each center c_j.

Variables: $\mathbf{x} \in \mathbb{B}^n$, $x_j = 1$ if center j is selected, 0 otherwise

Objective:

$$\min \sum_{j=1}^{n} c_j x_j$$

Constraints: All regions must be safely served: We define an incidence matrix A of size $m \times |\mathcal{F}|$:

$$a_{ij} = \begin{cases} 1 & \text{if center } j \text{ can cover region } i \\ 0 & \text{otherwise} \end{cases}$$

The constraint can then be expressed as:

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1.$$

In our numerical example the incidence matrix would look like:

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ S_1 & S_2 & S_3 & S_4 & S_5 & S_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

where we labeled the columns with the variables and the sets they represent and the rows by the region identifier. A feasible solution is a selection of the columns of the matrix such that they identify a submatrix that has at least a 1 in each row, or in other terms such that all rows are covered by the selected columns.

The one we formulated is a *set covering* problem. Two other variants, *set packing* and *set partitioning* are also relevant for real life applications. We sketch the formulations here.

Definition 5.3 (Set covering)**.** Cover each of M at least once:

1. min, $\geq$ type constraints
2. all RHS terms are 1
3. all matrix elements are 0 or 1

$$\begin{aligned} \min\ & \mathbf{c}^T\mathbf{x} \\ & A\mathbf{x} \geq \mathbf{1} \\ & \mathbf{x} \in \mathbb{B}^n \end{aligned}$$

Definition 5.4 (Set packing)**.** Cover as many of M without overlap:

1. max, $\leq$ type constraints
2. all RHS terms are 1
3. all matrix elements are 0 or 1

$$\begin{aligned} \max\ & \mathbf{c}^T\mathbf{x} \\ & A\mathbf{x} \leq \mathbf{1} \\ & \mathbf{x} \in \mathbb{B}^n \end{aligned}$$

Definition 5.5 (Set partitioning)**.** Cover exactly once each element of M:

1. max or min, $=$ type constraints
2. all RHS terms are 1
3. all matrix elements are 0 or 1

$$\begin{aligned} \max\ & \mathbf{c}^T\mathbf{x} \\ & A\mathbf{x} = \mathbf{1} \\ & \mathbf{x} \in \mathbb{B}^n \end{aligned}$$

These problems can be generalized to the cases where the coverage must be larger than 1, that is, where the right hand side of the constraints are larger than 1.

These problems have several applications. Examples are:

- Aircrew scheduling: the legs to cover define the set M and the rosters (crew shifts during which a crew can cover a number of legs) defines the set N.
- Vehicle routing: the customers to visit define the set M, the routes that visit customers define the set N.

Here is an example that can be modeled as a generalized set covering.

Example 5.6 (Manpower Planning)**.**
Input A set of workers, a set of 15 working hours per day with a required staffing per hour. Each person works in shifts that covers 7 hours. A person starting in hour i contributes to the workload in hours $i, \ldots, i+6$ (Eg: A person starting in hour 3 contributes to the workload in hours 3,4,5,6,7,8,9).
Task: Determine the number of people needed to cover the workload.

Decision Variables:

- $x_i \in \mathbb{N}_0$: number of people starting to work in hour i ($i = 1, \ldots, 15$). For easiness of expressing the constraints we also define the variables x_i, $i = -5, ..., -1, 0$.

Objective Function:

$$\min \sum_{i=1}^{9} x_i$$

Constraints:

- Demand:

$$\sum_{i=t-6}^{i=t} x_i \geq d_t \text{ for } t = 1, \ldots, 15$$

- Bounds:

$$x_{-5}, \ldots, x_0 = 0$$

5.2.4 Graph Problems

Matching

Definition 5.6 (Matching Theory – Terminology)**.** A *Matching* M of a graph $G = (V, E)$ is a set of pairwise non adjacent edges. A vertex is *covered* by the matching M if it is incident to an edge in M. A matching is *perfect* if it covers all vertices in G. A matching is *maximal* if it cannot be extended any further. A *maximum matching* is a matching that covers as many vertices as possible. A graph G is *matchable* it has a perfect matching.

For a graph with weights on the edges, the weight of a matching is the sum of the weights on the edges of the matching.

Definition 5.7 (Maximum weighted matching problem)**.** Given a graph $G = (V, E)$ and weights w_e on the edges $e \in E$ find the matching of maximum weight.

The MILP formulation of the *maximum weighted matching* problem is:

$$\begin{aligned} \max \quad & \sum_{v \in V} w_e x_e & & \\ & \sum_{e \in E : v \in e} x_e \;\leq\; 1 & & \forall v \in V \\ & x_e \;\in\; \{0,1\} & & \forall e \in E \end{aligned}$$

Binary variables indicate whether an edge is selected or not. The constraint ensures that for each vertex the number of selected edges that are incident to the vertex are not more than 1.

A particular case is a *bipartite matching* that arises when the graph is bipartite. A bipartite matching is equivalent to an assignment problem.

Vertex Cover

Definition 5.8 (Vertex cover problem)**.** Given a graph G, select a subset $S \subseteq V$ such that each edge has at least one end vertex in S.

The MILP formulation is

$$\begin{aligned} \min \quad & \sum_{v \in V} x_v & & \\ & x_v + x_u \;\geq\; 1 & & \forall u, v \in V, uv \in E \\ & x_v \;\in\; \{0,1\} & & \forall v \in V \end{aligned}$$

Roughly said, an **approximation algorithm** is an algorithm that runs in polynomial time and that guarantees in the worst case a certain approximation ratio with respect to the optimal solution. Formally, if $OPT(\pi)$ is the optimal solution of an instance π of a minimization problem, and $\mathcal{A}(\pi)$ is the solution found by the approximation algorithm, the approximation ratio AR is defined as:

$$AR = \max_{\pi} \frac{\mathcal{A}(\pi)}{OPT(\pi)}$$

An approximation algorithm for vertex cover can be easily derived from the linear programming solution. Let $\mathbf{x}^*$ be the optimal solution of the linear programming relaxation of the MILP formulation of the vertex cover problem. Then, a cover S_{LP} can be constructed by selecting the vertices whose variables received a value larger than $1/2$, that is:

$$S_{LP} = \{v \in V : x_v^* \geq 1/2\}.$$

The set S_{LP} is a cover since $x_v^* + x_u^* \geq 1$ implies $x_v^* \geq 1/2$ or $x_u^* \geq 1/2$.

Proposition 5.1. *The LP rounding approximation algorithm described above gives a 2-approximation:* $|S_{LP}| \leq 2|S_{OPT}|$ *(at most as bad as twice the optimal solution)*

Proof. Let $\bar{x}$ be the optimal solution for the MILP formulation of the vertex cover. Then $\sum x_v^* \leq \sum \bar{x}_v$. Moreover,

$$|S_{LP}| = \sum_{v \in S_{LP}} 1 \leq \sum_{v \in V} 2x_v^*$$

since $x_v^* \geq 1/2$ for each $v \in S_{LP}$ and thus

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

□

Maximum independent Set

Definition 5.9 (Maximum independent set problem)**.** Given a graph $G = (V, E)$, find the largest subset $S \subseteq V$ such that the induced graph, i.e., the graph $(S, \{uv : uv \in E, u, v \in S\})$ has no edges.

We denote by x_v for $v \in V$ the decision variables that indicate whether a vertex is part of the independent set or not. The MILP formulation is:

$$\begin{array}{lrll} \max & \sum\limits_{v \in V} x_v & & \\ & x_v + x_u & \leq 1 & \forall u, v \in V, uv \in E \\ & x_v & \in \{0, 1\} & \forall v \in V \end{array}$$

Also in this case we could design an algorithm that rounds the LP relaxation of the MILP formulation. The optimal solution to this LP problem sets $x_v = 1/2$ for all variables and has value $|V|/2$. This fact implies that the LP relaxation rounding algorithm gives an $O(n)$-approximation (almost useless). (To prove this fact think about the worst possible instance which is a complete graph. What is the optimal integer max independent set solution for a complete graph?)

Traveling Salesman Problem

The traveling salesman problem has several applications. Here is one. Find the cheapest movement for a drilling, welding, drawing, soldering arm as, for example, in a printed circuit board manufacturing process or car manufacturing process.

Definition 5.10 (Traveling salesman problem)**.** Given a set of n locations and costs c_{ij} of travelling from one location i to another location j, find the cheapest tour that visits all locations.

The problem is modeled in graph terms by defining a directed graph $D = (V, A)$. In this context a tour that visit all vertices is called a Hamiltonian tour. Note that if the costs are symmetric everywhere then the graph can be undirected.

The problem can be formulated as a MILP problem as follows.

Parameters The set of locations identified by $1, ..., n$ indexed by i and j and the set of traveling costs c_{ij}.

Variables

$$x_{ij} = \begin{cases} 1 & \text{if the edge } ij \text{ is part of the tour} \\ 0 & \text{otherwise} \end{cases}$$

Objective

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

Constraints Each location must have an entering and an outgoing arc:

$$\sum_{j:j\neq i} x_{ij} = 1 \qquad \forall i = 1,\dots,n$$

$$\sum_{i:i\neq j} x_{ij} = 1 \qquad \forall j = 1,\dots,n$$

The previous constraints alone do not remove the possibility that subtours (cycles) are found. To eliminate this possibility there are two ways:

- cut set constraints

$$\sum_{i\in S}\sum_{j\notin S} x_{ij} \geq 1 \qquad \forall S \subset N, S \neq \emptyset$$

- subtour elimination constraints

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 1 \qquad \forall S \subset N, 2 \leq |S| \leq n-1$$

The problem with these constraints is that there are exponentially many (look at the quantifiers on the right side). One can learn how to deal with this issue in one of the assignments.

5.3 Modeling Tricks

In this section we review a few modeling tricks. This material is taken from Chapter 9 of Williams [2013]. MILP problems can be defined also when the objective function and/or constraints do not appear to be linear at first sight. Consider for example the following cases:

- Absolute values
- Minimize the largest function value
- Maximize the smallest function value
- Constraints including variable division
- Constraints are either/or
- A variable must take one of several candidate values

Modeling Trick I: "Min max" Minimize the largest of a number of function values:

$$\min \ \max\{f(x_1),\dots,f(x_n)\}$$

Introduce an auxiliary variable z:

$$\begin{aligned} \min \quad & z \\ \text{s. t.} \quad & f(x_1) \leq z \\ & f(x_2) \leq z \\ & \dots \\ & f(x_n) \leq z \end{aligned} \tag{5.1}$$

(5.2)

Example 5.7.

$$\min \max 3y_1 + 4y_2 + 2y_3$$

Reformulate as:

$$\begin{aligned} \min \quad & z \\ \text{s. t.} \quad & 3y_1 \leq z \\ & 4y_2 \leq z \\ & 2y_3 \leq z \end{aligned}$$

Modeling Trick II: "Division" Constraints including variable division:

$$\frac{a_1x + a_2y + a_3z}{d_1x + d_2y + d_3z} \leq b$$

Rearrange:

$$a_1x + a_2y + a_3z \leq b(d_1x + d_2y + d_3z)$$

which gives:

$$(a_1 - bd_1)x + (a_2 - bd_2)y + (a_3 - bd_3)z \leq 0$$

Example 5.8. Constraint of the form

$$\frac{3x + 4y + 6z}{x + y + z} \leq 10$$

Rearrange:

$$3x + 4y + 6z \leq 10(x + y + z)$$

which gives:

$$7x + 6y + 4z \geq 0$$

Modeling Trick III: "Either/Or Constraints" In conventional mathematical models, the solution must satisfy all constraints. Suppose that your constraints are of the type "either/or":

$$\begin{array}{ll} a_1x_1 + a_2x_2 \leq b_1 & \text{or} \\ d_1x_1 + d_2x_2 \leq b_2 & \end{array}$$

Introduce a new variable $y \in \{0, 1\}$ and a large number M:

$$\begin{array}{ll} a_1x_1 + a_2x_2 \leq b_1 + My & \text{if } y = 0 \text{ then this is active} \\ d_1x_1 + d_2x_2 \leq b_2 + M(1 - y) & \text{if } y = 1 \text{ then this is active} \end{array}$$

Hence, binary integer programming allows to model alternative choices. For example, we can model the case of two disjoint feasible regions, ie, disjunctive constraints, which are not possible in LP.

We introduce an auxiliary binary variable y and M, a big number:

$$\begin{array}{ll} Ax \leq b + My & \text{if } y = 0 \text{ then this is active} \\ A'x \leq b' + M(1 - y) & \text{if } y = 1 \text{ then this is active} \end{array}$$

Example 5.9. At least one of the two constraints must be satisfied:

$$3x_1 + 2x_2 \leq 18 \quad \text{or} \quad x_1 + 4x_2 \leq 16$$

Introduce new variable $y \in \{0, 1\}$ and a large number M:

$$3x_1 + 2x_2 \leq 18 + My$$

$$x_1 + 4x_2 \leq 16 + M(1 - y)$$

If $y = 1$ then $x_1 + 4x_2 \leq 16$ is the active constraint and the other is always satisfied.
If $y = 0$ then $3x_1 + 2x_2 \leq 18$ is the active constraints and the other is always satisfied.

Modeling Trick IV: "Either/Or Constraints" We can generalize the previous trick to the case where **Exactly** K of the N constraints:

$$\begin{array}{ll} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1m}x_m \leq d_1 & \text{or} \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2m}x_m \leq d_2 & \text{or} \\ \vdots & \\ a_{m1}x_1 + a_{N2}x_2 + a_{N3}x_3 + \ldots + a_{Nm}x_m \leq d_N & \end{array}$$

must be satisfied. We need to introduce binary variables $y_1, y_2, \ldots, y_N$ and a large number M and impose:

$$\begin{array}{l} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1m}x_m \leq d_1 + My_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2m}x_m \leq d_2 + My_2 \\ \vdots \\ a_{m1}x_1 + a_{N2}x_2 + a_{N3}x_3 + \ldots + a_{Nm}x_m \leq d_N + My_N \end{array}$$

$$y_1 + y_2 + \ldots y_N = N - K$$

Since in a feasible solution K of the y-variables will be 0, then K constraints will be satisfied.

Similarly we can model the case where *at least* $h \leq k$ of $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$, $i = 1, \ldots, k$ must be satisfied. We introduce y_i, $i = 1, ..., k$ auxiliary binary variables and impose:

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i + My_i$$

$$\sum_{i} y_i \leq k - h$$

Modeling Trick V: "Possible Constraints Values" A constraint must take on one of N given values:

$$\begin{array}{ll} a_1x_1 + a_2x_2 + a_3x_3 + \ldots + a_mx_m = d_1 & \text{or} \\ a_1x_1 + a_2x_2 + a_3x_3 + \ldots + a_mx_m = d_2 & \text{or} \\ \vdots & \\ a_1x_1 + a_2x_2 + a_3x_3 + \ldots + a_mx_m = d_N & \end{array}$$

We introduce the binary variables $y_1, y_2, \ldots, y_N$ and impose:

$$a_1x_1 + a_2x_2 + a_3x_3 + \ldots + a_mx_m = d_1y_1 + d_2y_2 + \ldots d_Ny_N$$

$$y_1 + y_2 + \ldots y_N = 1$$

Example 5.10. The constraint must equal 6 *or* 8 *or* 12:

- $3x_1 + 2x_2 = 6$ or
- $3x_1 + 2x_2 = 8$ or
- $3x_1 + 2x_2 = 12$ or

Reformulate with auxiliary variables $y_1, y_2, y_3 \in \{0, 1\}$:

- $3x_1 + 2x_2 = 6y_1 + 8y_2 + 12y_3$ and
- $y_1 + y_2 + y_3 = 1$

Example 5.11 (Dijunctive constraints in scheduling)**.** Two tasks, P and Q, must be performed by the same person. The duration of P (resp. Q) is d_p units. The start time of P (resp. Q) is denoted as s_p (s_q).

We want to enforce either $s_p + d_p \leq s_q$ or $s_q + d_q \leq s_p$.

Trick: Define binary variable i_{pq}, indicating if P precedes Q Introduce the following constraints

$$s_p + d_p \leq s_q + M(1 - i_{pq})$$

$$s_q + d_q \leq s_p + Mi_{pq}$$

5.4 Formulations

Problems can have more than one MILP formulation. Let's start by considering the *Uncapacited Facility Location* problem.

Definition 5.11 (Uncapacited Facility Location (UFL))**.**
Given:

- depots $N = \{1, \dots, n\}$
- clients $M = \{1, \dots, m\}$
- clients demand is $d_i = 1$ for all $i \in M$
- f_j fixed cost to use depot j
- transport cost for all orders c_{ij}

Task: Determine which depots are most convient to open and which depots serve which client.

An MILP formulation for this problem is:

Variables

$$y_j = \begin{cases} 1 & \text{if depot open} \\ 0 & \text{otherwise} \end{cases},$$

x_{ij} fraction of demand $d_i = 1$ of client i satisfied by depot j.

Objective

$$\min \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} + \sum_{j \in N} f_j y_j$$

Constraints

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i = 1, \dots, m$$

$$\sum_{i \in M} x_{ij} \leq m y_j \qquad \forall j \in N$$

An alternative formulation for the last constraints is the following:

$$x_{ij} \leq y_j \quad \forall i \in M, j \in N$$

Then which formulation should we prefer?

5.4.1 Alternative Formulations

Definition 5.12 (Formulation)**.** A polyhedron $P \subseteq \mathbb{R}^{n+p}$ is a **formulation** for a set $X \subseteq \mathbb{Z}^n \times \mathbb{R}^p$ if and only if $X = P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$

That is, if it does not leave out any of the solutions of the feasible region X.

There are **infinite** formulations.

Definition 5.13 (Convex Hull)**.** Given a set $X \subseteq \mathbb{Z}^n$ the **convex hull** of X is defined as:

$$\text{conv}(X) = \Big\{ \mathbf{x} : \mathbf{x} = \sum_{i=1}^{t} \lambda_i \mathbf{x}^i, \sum_{i=1}^{t} \lambda_i = 1, \lambda_i \geq 0, \text{for } i = 1, \dots, t, \\ \text{for all finite subsets } \{\mathbf{x}^1, \dots, \mathbf{x}^t\} \text{ of } X \Big\}$$

Proposition 5.2. *conv(X) is a polyhedron (ie, representable as* $A\mathbf{x} \leq \mathbf{b}$*)*

Proposition 5.3. *Extreme points of conv(X) all lie in X*

Hence:

$$\max\{\mathbf{c}^T\mathbf{x} : \mathbf{x} \in X\} \equiv \max\{\mathbf{c}^T\mathbf{x} : \mathbf{x} \in \text{conv}(X)\}$$

This is an important result, it means that we can solve the integer programming problem by solving its linear programming problem relaxation. However the description of the convex hull $\text{conv}(X)$ may require an exponential number of inequalities to describe and it may be not known.

What makes a formulation better than another? Let's suppose that we have two formulations P_1 and P_2 and that

$$X \subseteq \text{conv}(X) \subseteq P_1 \subset P_2$$

Then we can conclude that:

$$P_1 \text{ is better than } P_2$$

Definition 5.14. Given a set $X \subseteq \mathbb{R}^n$ and two formulations P_1 and P_2 for X, P_1 is a better formulation than P_2 if $P_1 \subset P_2$.

We can now get back to our two alternative formulations for the UFL problem.

Example 5.12 (Formulations for the UFL)**.**

- $P_1 = \text{UFL with } \sum_{i \in M} x_{ij} \leq m y_j \quad \forall j \in N$
- $P_2 = \text{UFL with } x_{ij} \leq y_j \quad \forall i \in M, j \in N$

We show that

$$P_2 \subset P_1$$

- $P_2 \subseteq P_1$ because summing $x_{ij} \leq y_j$ over $i \in M$ we obtain $\sum_{i \in M} x_{ij} \leq m y_j$

- $P_2 \subset P_1$ because there exists a point in P_1 but not in P_2: for example, let $m = 6 = 3 \cdot 2 = k \cdot n$ The following solution

  $$x_{10} = 1,\ x_{20} = 1,\ x_{30} = 1,$$
  $$x_{41} = 1,\ x_{51} = 1,\ x_{61} = 1$$

  under formulation P_1 would admit a fractional value for y_0 and y_1

  $$\begin{array}{ll} \sum_i x_{i0} \leq 6y_0 & y_0 = 1/2 \\ \sum_i x_{i1} \leq 6y_1 & y_1 = 1/2 \end{array}$$

  while under the formulation P_2 the variables y could not take a fractional value. Since they must be integer for their proper use in the objective function, then we showed that there is a solution in P_1 but not in P_2 while not removing any feasible solution.

Chapter 6

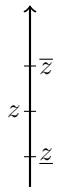
Well Solved Problems

6.1 Relaxations

Suppose we have the following ILP problem:

$$z = \max\{c(\mathbf{x}) : \mathbf{x} \in X \subseteq \mathbb{Z}^n\}$$

The set X represents the set of all feasible solutions. In an ILP this set is a subset of $\mathbb{Z}^n$. Since the problem is a maximization problem any feasible solution $\mathbf{x}^*$ of value $\underline{z}$ gives a *lower bound* to z. Then, to prove the optimality of a feasible solution we need also an *upper bound*, $\overline{z}$. Then if $\underline{z} = \overline{z}$, the solution that gives $\underline{z}$ is optimal. Alternatively, we can stop our search process when $\overline{z} - \underline{z} \leq \epsilon$ for a given reasonably small ϵ.



The concepts, roles and determination of upper and lower bounds are linked to the sense of the optimization function. In a minimization problem their roles are exchanged. To avoid this dependency on the sense of the objective function, the following concepts are instead used.

- **Primal bounds**: Every feasible solution gives a primal bound. In some problems it may be easy or hard to find feasible solutions. Heuristics are used to provide such type of solutions.
- **Dual bounds**: They are obtained through relaxations of the problem formulation.

In our initial maximization problem, the lower bounds are primal bounds and the upper bounds are dual bounds.

Given a primal bound pb and a dual bound db it is possible to calculate the optimality gap:

$$\text{gap} = \frac{|pb - db|}{pb + \epsilon} \cdot 100$$

The ϵ is added to avoid division by zero when $pb = 0$. To avoid a confusing behaviour when 0 lies in between pb and db a different definition, which includes the above one, is often used. For a minimization problem, this is:

$$\text{gap} = \frac{pb - db}{\inf\{|z|, z \in [db, pb]\}} \cdot 100$$

If $pb \geq 0$ and $db \geq 0$ then $\frac{pb-db}{db}$. If $db = pb = 0$ then gap $= 0$. If no feasible solution is found or $db \leq 0 \leq pb$ then the gap is not computed.

Proposition 6.1.

$$\begin{array}{ll} (RP) & z^R = \max\{f(\mathbf{x}) : \mathbf{x} \in T \subseteq \mathbb{R}^n\} \text{ is a relaxation of a problem} \\ (IP) & z = \max\{c(\mathbf{x}) : \mathbf{x} \in X \subseteq \mathbb{Z}^n\} \text{ if :} \end{array}$$

(i) $X \subseteq T$ *or*

(ii) $f(\mathbf{x}) \geq c(\mathbf{x}) \; \forall \mathbf{x} \in X$

In other terms:

$$\max_{\mathbf{x} \in T} f(\mathbf{x}) \geq \left\{ \begin{array}{l} \max_{\mathbf{x} \in T} c(\mathbf{x}) \\ \max_{\mathbf{x} \in X} f(\mathbf{x}) \end{array} \right\} \geq \max_{\mathbf{x} \in X} c(\mathbf{x})$$

- T is the set of candidate solutions;
- $X \subseteq T$ is the set of feasible solutions;
- $f(\mathbf{x}) \geq c(\mathbf{x})$

Proposition 6.2. *(i) If a relaxation RP is infeasible, the original problem IP is infeasible.*

(ii) Let $\mathbf{x}^*$ *be an optimal solution for RP. If* $\mathbf{x}^* \in X$ *and* $f(\mathbf{x}^*) = c(\mathbf{x}^*)$ *then* $\mathbf{x}^*$ *is optimal for IP.*

There are at least four ways to construct relaxations.

1. Linear relaxation Given

$$\text{(IP)} : \max\{\mathbf{c}^T\mathbf{x} : \mathbf{x} \in P \cap \mathbb{Z}^n\}, \text{ where } P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\}$$

the linear relaxation is

$$\text{(LPR)} : \max\{\mathbf{c}^T\mathbf{x} : \mathbf{x} \in P\}$$

Better formulations give better bounds ($P_1 \subseteq P_2$)

Combinatorial relaxations Some complicated constraints are removed leaving a problem easy (that is, in polynomial time) to solve. For example, the TSP can be reduced to an Assignment problem by dropping the subtour elimination constraints.

Lagrangian relaxation It is obtained by bringing all or some constraints into the objective function via multipliers. That is,

$$\begin{array}{ll} IP: & z = \max\{\mathbf{c}^T\mathbf{x} : A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in X \subseteq \mathbb{Z}^n\} \\ LR: & z(\mathbf{u}) = \max\{\mathbf{c}^T\mathbf{x} + \mathbf{u}(\mathbf{b} - A\mathbf{x}) : \mathbf{x} \in X\} \end{array}$$

Then for all $\mathbf{u} \geq \mathbf{0}$ it is $z(\mathbf{u}) \geq z$.

For both combinatorial and Lagrangian relaxations the constraints that is worth trying to relax are the ones that

- do not worsen too much the quality of bound (tightness of relaxation)
- leave a remaining problem that can be solved efficiently
- have multipliers that can be found efficiently
- are difficult to formulate mathematically
- are too expensive to write up

Duality the concept of duality works only for linear programming. We can adapt the definitions of dual problems to integer programming as follows:

Definition 6.1. Two problems:

$$z = \max\{c(\mathbf{x}) : \mathbf{x} \in X\} \qquad w = \min\{w(\mathbf{u}) : \mathbf{u} \in U\}$$

where X and U are two arbitrary sets from $\mathbb{R}^n$ form a **weak-dual pair** if $c(\mathbf{x}) \leq w(\mathbf{u})$ for all $\mathbf{x} \in X$ and all $\mathbf{u} \in U$.
When $z = w$ they form a **strong-dual pair**.

Proposition 6.3. *$z = \max\{\mathbf{c}^T\mathbf{x} : A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^n_+\}$ and $w^{LP} = \min\{\mathbf{u}\mathbf{b}^T : \mathbf{u}A \geq \mathbf{c}, \mathbf{u} \in \mathbb{R}^m_+\}$ (ie, the dual of the linear relaxation) form a weak-dual pair.*

Proposition 6.4. *Let IP and D be a weak-dual pair, $\max\{c(\mathbf{x}) : \mathbf{x} \in X\}$ and $\min\{w(\mathbf{u}) : \mathbf{u} \in U\}$, respectively. Then:*

(i) If D is unbounded, then IP is infeasible

(ii) If $\mathbf{x}^* \in X$ and $\mathbf{u}^* \in U$ satisfying $c(\mathbf{x}^*) = w(\mathbf{u}^*)$ then $\mathbf{x}^*$ is optimal for IP and $\mathbf{u}^*$ is optimal for D.

An advantage of ILP with respect to LP is that once we have a dual problem we do not need to solve an LP like in the LP relaxation to have a bound, any feasible dual solution gives a dual bound for the primal problem.

Here are some examples of dual pairs in ILP:

Example 6.1.
Matching: $z = \max\{\mathbf{1}^T\mathbf{x} : A\mathbf{x} \leq \mathbf{1}, \mathbf{x} \in \mathbb{Z}^m_+\}$
Vertex cover: $w = \min\{\mathbf{1}^T\mathbf{y} : \mathbf{y}^T A \geq 1, \mathbf{y} \in \mathbb{Z}^n_+\}$

are weak dual pairs. Indeed, it is easy to see that LP relaxations of these two problems are dual of each other, then

$$z \leq z^{LP} = w^{LP} \leq w.$$

The two problems are strong-dual pairs when the graphs are bipartite.

Example 6.2.
Packing: $z = \max\{\mathbf{1}^T\mathbf{x} : A\mathbf{x} \leq \mathbf{1}, \mathbf{x} \in \mathbb{Z}^n_+\}$
Set covering: $w = \min\{\mathbf{1}^T\mathbf{y} : A^T\mathbf{y} \geq \mathbf{1}, \mathbf{y} \in \mathbb{Z}^m_+\}$
are weak-dual pairs, which is provable again via the duality of the linear relaxation.

6.2 Well Solved Problems

6.2.1 Separation problem

We have seen that

$$\max\{\mathbf{c}^T\mathbf{x} : \mathbf{x} \in X\} \equiv \max\{\mathbf{c}^T\mathbf{x} : \mathbf{x} \in \text{conv}(X)\}$$

where $X \subseteq \mathbb{Z}^n$ and $X = P \cap \mathbb{Z}^n$, P a polyhedron $P \subseteq \mathbb{R}^n$.

Definition 6.2 (Separation problem for a COP)**.** Given $\mathbf{x}^* \in P$, we want to determine whether $\mathbf{x}^* \in \text{conv}(X)$ and if not find an inequality $\mathbf{a}\mathbf{x} \leq \mathbf{b}$ satisfied by all points in X but violated by the point $\mathbf{x}^*$.

Farkas' lemma states the existence of such an inequality.

The following four properties often go together:

(i) **Efficient optimization property**: there exists a polynomial time algorithm for $\max\{\mathbf{c}\mathbf{x} : \mathbf{x} \in X \subseteq \mathbb{R}^n\}$

(ii) **Strong duality property**: there exists a strong dual D $\min\{w(\mathbf{u}) : \mathbf{u} \in U\}$ that allows to quickly verify optimality

(iii) **Efficient separation property**: there exists an efficient algorithm for the separation problem

(iv) **Efficient convex hull property**: there is a compact description of the convex hull available.

Note that if the explicit convex hull property is true, then the strong duality property and the efficient separation property hold (just description of $\text{conv}(X)$).

Problems that are easy have typically all four properties satisfied.

Polyhedral analysis is the field of theoretical analysis to prove results about: the strength of certain inequalities that are facet defining and the descriptions of convex hull of some discrete $X \subseteq \mathbb{Z}^*$ (we see one way to do this in the next section).

Example 6.3. Let

$$X = \{(x, y) \in \mathbb{R}^m_+ \times \mathbb{B}^1 : \sum_{i=1}^{m} x_i \leq my, x_i \leq 1 \text{ for } i = 1, \ldots, m\}$$

$$P = \{(x, y) \in \mathbb{R}^n_+ \times \mathbb{R}^1 : x_i \leq y \text{ for } i = 1, \ldots, m, y \leq 1\}.$$

The polytope P describes $\text{conv}(X)$.

6.3 Totally Unimodular Matrices

When will the LP solution to this problem

$$IP: \quad \max\{\mathbf{c}^T\mathbf{x} : A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^n_+\}$$

with all data (the parameters $\mathbf{c}, \mathbf{x}, \mathbf{b}$), integer, be integer?

Let's look back at the simplex in matrix notation.

$$\left[\begin{array}{c|c|c|c} A_N & A_B & \mathbf{0} & \mathbf{b} \\ \hline \mathbf{c}_N^T & \mathbf{c}_B^T & 1 & 0 \end{array}\right]$$

$A_B x_B + A_N x_N = b$

$\mathbf{x}_N = \mathbf{0} \rightsquigarrow A_B\mathbf{x}_B = \mathbf{b}$,
A_B $m \times m$ non singular matrix
$\mathbf{x}_B \geq 0$
Cramer's rule for solving systems of linear equations:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} e \\ f \end{bmatrix}$$

$$x = \frac{\begin{vmatrix} e & b \\ f & d \end{vmatrix}}{\begin{vmatrix} a & b \\ c & d \end{vmatrix}} \qquad y = \frac{\begin{vmatrix} a & e \\ c & f \end{vmatrix}}{\begin{vmatrix} a & b \\ c & d \end{vmatrix}}$$

$$\mathbf{x} = A_B^{-1}\mathbf{b} = \frac{A_B^{adj}\mathbf{b}}{\det(A_B)}$$

Definition 6.3.

- A square integer matrix B is called **unimodular** (UM) if $\det(B) = \pm 1$
- An integer matrix A is called **totally unimodular** (TUM) if every square, nonsingular submatrix of A is UM

Proposition 6.5.
- *If A is TUM then all vertices of $R_1(A) = \{x : Ax = b, x \geq 0\}$ are integer if b is integer*
- *If A is TUM then all vertices of $R_2(A) = \{x : Ax \leq b, x \geq 0\}$ are integer if b is integer.*

Proof. if A is TUM then $\left[\begin{array}{c|c} A & I \end{array}\right]$ is TUM

Any square, nonsingular submatrix C of $\left[\begin{array}{c|c} A & I \end{array}\right]$ can be written as

$$C = \left[\begin{array}{c|c} B & 0 \\ \hline D & I_k \end{array}\right]$$

where B is square submatrix of A. Hence $\det(C) = \det(B) = \pm 1$ □

Proposition 6.6. *The transpose matrix A^T of a TUM matrix A is also TUM.*

Theorem 6.7 (Sufficient condition)**.** *An integer matrix A is TUM if*

1. *$a_{ij} \in \{0, -1, +1\}$ for all i, j*
2. *each column contains at most two non-zero coefficients ($\sum_{i=1}^{m} |a_{ij}| \leq 2$)*
3. *if the rows can be partitioned into two sets I_1, I_2 such that:*
   - *if a column has 2 entries of same sign, their rows are in different sets*
   - *if a column has 2 entries of different signs, their rows are in the same set*

$$\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & -1 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Proof: by induction

Basis: one matrix of one element $\{+1, -1\}$ is TUM

Induction: let C be of size k.

If C has column with all 0s then it is singular.

If a column with only one 1 then expand on that by induction

If 2 non-zero in each column then

$$\forall j : \sum_{i \in I_1} a_{ij} = \sum_{i \in I_2} a_{ij}$$

but then linear combination of rows and $\det(C) = 0$

Other matrices with integrality property:

- TUM
- Balanced matrices
- Perfect matrices
- Integer vertices

Defined in terms of forbidden substructures that represent fractionating possibilities.

Proposition 6.8. *A is always TUM if it comes from*

- *node-edge incidence matrix of undirected bipartite graphs (ie, no odd cycles) ($I_1 = U, I_2 = V, B = (U, V, E)$)*
- *node-arc incidence matrix of directed graphs ($I_2 = \emptyset$)*

Eg: Shortest path, max flow, min cost flow, bipartite weighted matching

Chapter 7

Network Flows

models arise in broader problem contexts and how the algorithms that we have developed for the core models can be used in conjunction with other methods to solve more complex problems that arise frequently in practice. In particular, this discussion permits us to introduce and describe the basic ideas of decomposition methods for several important network optimization models—constrained shortest paths, the traveling salesman problem, vehicle routing problem, multicommodity flows, and network design.

Since the proof of the pudding is in the eating, we have also included a chapter on some aspects of computational testing of algorithms. We devote much of our discussion to devising the best possible algorithms for solving network flow problems, in the theoretical sense of computational complexity theory. Although the theoretical model of computation that we are using has proven to be a valuable guide for modeling and predicting the performance of algorithms in practice, it is not a perfect model, and therefore algorithms that are not theoretically superior often perform best in practice. Although empirical testing of algorithms has traditionally been a valuable means for investigating algorithmic ideas, the applied mathematics, computer science, and operations research communities have not yet reached a consensus on how to measure algorithmic performance empirically. So in this chapter we not only report on computational experience with an algorithm we have presented, but also offer some thoughts on how to measure computational performance and compare algorithms.

1.2 NETWORK FLOW PROBLEMS

In this section we introduce the network flow models we study in this book, and in the next section we present several applications that illustrate the practical importance of these models. In both the text and exercises throughout the remaining chapters, we introduce many other applications. In particular, Chapter 19 contains a more comprehensive summary of applications with illustrations drawn from several specialties in applied mathematics, engineering, logistics, manufacturing, and the physical sciences.

Minimum Cost Flow Problem

The minimum cost flow model is the most fundamental of all network flow problems. Indeed, we devote most of this book to the minimum cost flow problem, special cases of it, and several of its generalizations. The problem is easy to state: We wish to determine a least cost shipment of a commodity through a network in order to satisfy demands at certain nodes from available supplies at other nodes. This model has a number of familiar applications: the distribution of a product from manufacturing plants to warehouses, or from warehouses to retailers; the flow of raw material and intermediate goods through the various machining stations in a production line; the routing of automobiles through an urban street network; and the routing of calls through the telephone system. As we will see later in this chapter and in Chapters 9 and 19, the minimum cost flow model also has many less transparent applications.

In this section we present a mathematical programming formulation of the minimum cost flow problem and then describe several of its specializations and

variants as well as other basic models that we consider in later chapters. We assume our readers are familiar with the basic notation and definitions of graph theory; those readers without this background might consult Section 2.2 for a brief account of this material.

Let $G = (N, A)$ be a directed network defined by a set N of n *nodes* and a set A of m *directed arcs*. Each arc $(i, j) \in A$ has an associated *cost* c_{ij} that denotes the cost per unit flow on that arc. We assume that the flow cost varies linearly with the amount of flow. We also associate with each arc $(i, j) \in A$ a *capacity* u_{ij} that denotes the maximum amount that can flow on the arc and a *lower bound* l_{ij} that denotes the minimum amount that must flow on the arc. We associate with each node $i \in N$ an integer number $b(i)$ representing its supply/demand. If $b(i) > 0$, node i is a *supply node*; if $b(i) < 0$, node i is a *demand node* with a demand of $-b(i)$; and if $b(i) = 0$, node i is a *transshipment node*. The decision variables in the minimum cost flow problem are arc flows and we represent the flow on an arc $(i, j) \in A$ by x_{ij}. The minimum cost flow problem is an optimization model formulated as follows:

$$\text{Minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1.1a}$$

subject to

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i) \qquad \text{for all } i \in N, \tag{1.1b}$$

$$l_{ij} \le x_{ij} \le u_{ij} \qquad \text{for all } (i,j) \in A, \tag{1.1c}$$

where $\sum_{i=1}^{n} b(i) = 0$. In matrix form, we represent the minimum cost flow problem as follows:

$$\text{Minimize } cx \tag{1.2a}$$

subject to

$$\mathcal{N}x = b, \tag{1.2b}$$

$$l \le x \le u. \tag{1.2c}$$

In this formulation, $\mathcal{N}$ is an $n \times m$ matrix, called the *node–arc incidence matrix* of the minimum cost flow problem. Each column $\mathcal{N}_{ij}$ in the matrix corresponds to the variable x_{ij}. The column $\mathcal{N}_{ij}$ has a $+1$ in the ith row, a -1 in the jth row; the rest of its entries are zero.

We refer to the constraints in (1.1b) as *mass balance constraints*. The first term in this constraint for a node represents the total *outflow* of the node (i.e., the flow emanating from the node) and the second term represents the total *inflow* of the node (i.e., the flow entering the node). The mass balance constraint states that the outflow minus inflow must equal the supply/demand of the node. If the node is a supply node, its outflow exceeds its inflow; if the node is a demand node, its inflow exceeds its outflow; and if the node is a transshipment node, its outflow equals its inflow. The flow must also satisfy the lower bound and capacity constraints (1.1c), which we refer to as *flow bound constraints*. The flow bounds typically model physical capacities or restrictions imposed on the flows' operating ranges. In most applications, the lower bounds on arc flows are zero; therefore, if we do not state lower bounds for any problem, we assume that they have value zero.

In most parts of the book we assume that the data are integral (i.e., all arc capacities, arc costs, and supplies/demands of nodes are integral). We refer to this assumption as the *integrality assumption*. The integrality assumption is not restrictive for most applications because we can always transform rational data to integer data by multiplying them by a suitably large number. Moreover, we necessarily need to convert irrational numbers to rational numbers to represent them on a computer.

The following special versions of the minimum cost flow problem play a central role in the theory and applications of network flows.

Shortest path problem. The shortest path problem is perhaps the simplest of all network flow problems. For this problem we wish to find a path of minimum cost (or length) from a specified *source node* s to another specified *sink node* t, assuming that each arc $(i, j) \in A$ has an associated cost (or length) c_{ij}. Some of the simplest applications of the shortest path problem are to determine a path between two specified nodes of a network that has minimum length, or a path that takes least time to traverse, or a path that has the maximum reliability. As we will see in our later discussions, this basic model has applications in many different problem domains, such as equipment replacement, project scheduling, cash flow management, message routing in communication systems, and traffic flow through congested cities. If we set $b(s) = 1$, $b(t) = -1$, and $b(\text{i}) = 0$ for all other nodes in the minimum cost flow problem, the solution to the problem will send 1 unit of flow from node s to node t along the shortest path. The shortest path problem also models situations in which we wish to send flow from a single-source node to a single-sink node in an uncapacitated network. That is, if we wish to send v units of flow from node s to node t and the capacity of each arc of the network is at least v, we would send the flow along a shortest path from node s to node t. If we want to determine shortest paths from the source node s to every other node in the network, then in the minimum cost flow problem we set $b(s) = (n - 1)$ and $b(i) = -1$ for all other nodes. [We can set each arc capacity u_{ij} to any number larger than $(n - 1)$.] The minimum cost flow solution would then send unit flow from node s to every other node i along a shortest path.

Maximum flow problem. The maximum flow problem is in a sense a complementary model to the shortest path problem. The shortest path problem models situations in which flow incurs a cost but is not restricted by any capacities; in contrast, in the maximum flow problem flow incurs no costs but is restricted by flow bounds. The maximum flow problem seeks a feasible solution that sends the maximum amount of flow from a specified source node s to another specified sink node t. If we interpret u_{ij} as the maximum flow rate of arc (i, j), the maximum flow problem identifies the maximum steady-state flow that the network can send from node s to node t per unit time. Examples of the maximum flow problem include determining the maximum steady-state flow of (1) petroleum products in a pipeline network, (2) cars in a road network, (3) messages in a telecommunication network, and (4) electricity in an electrical network. We can formulate this problem as a minimum cost flow problem in the following manner. We set $b(i) = 0$ for all $i \in N$, $c_{ij} = 0$ for all $(i, j) \in A$, and introduce an additional arc (t, s) with cost $c_{ts} = -1$ and flow bound $u_{ts} = \infty$. Then the minimum cost flow solution maximizes the flow on arc (t, s); but

since any flow on arc (t, s) must travel from node s to node t through the arcs in A [since each $b(i) = 0$], the solution to the minimum cost flow problem will maximize the flow from node s to node t in the original network.

Assignment problem. The data of the assignment problem consist of two equally sized sets N_1 and N_2 (i.e., $| N_1 | = | N_2 |$), a collection of pairs $A \subseteq N_1 \times N_2$ representing possible assignments, and a cost c_{ij} associated with each element $(i, j) \in A$. In the assignment problem we wish to pair, at minimum possible cost, each object in N_1 with exactly one object in N_2. Examples of the assignment problem include assigning people to projects, jobs to machines, tenants to apartments, swimmers to events in a swimming meet, and medical school graduates to available internships. The assignment problem is a minimum cost flow problem in a network $G = (N_1 \cup N_2, A)$ with $b(i) = 1$ for all $i \in N_1$, $b(i) = -1$ for all $i \in N_2$, and $u_{ij} = 1$ for all $(i, j) \in A$.

Transportation problem. The transportation problem is a special case of the minimum cost flow problem with the property that the node set N is partitioned into two subsets N_1 and N_2 (of possibly unequal cardinality) so that (1) each node in N_1 is a supply node, (2) each node N_2 is a demand node, and (3) for each arc (i, j) in A, $i \in N_1$ and $j \in N_2$. The classical example of this problem is the distribution of goods from warehouses to customers. In this context the nodes in N_1 represent the warehouses, the nodes in N_2 represent customers (or, more typically, customer zones), and an arc (i, j) in A represents a distribution channel from warehouse i to customer j.

Circulation problem. The circulation problem is a minimum cost flow problem with only transshipment nodes; that is, $b(i) = 0$ for all $i \in N$. In this instance we wish to find a feasible flow that honors the lower and upper bounds l_{ij} and u_{ij} imposed on the arc flows x_{ij}. Since we never introduce any exogenous flow into the network or extract any flow from it, all the flow circulates around the network. We wish to find the circulation that has the minimum cost. The design of a routing schedule of a commercial airline provides one example of a circulation problem. In this setting, any airplane circulates among the airports of various cities; the lower bound l_{ij} imposed on an arc (i, j) is 1 if the airline needs to provide service between cities i and j, and so must dispatch an airplane on this arc (actually, the nodes will represent a combination of both a physical location and a time of day so that an arc connects, for example, New York City at 8 A.M. with Boston at 9 A.M.).

In this book, we also study the following generalizations of the minimum cost flow problem.

Convex cost flow problems. In the minimum cost flow problem, we assume that the cost of the flow on any arc varies linearly with the amount of flow. Convex cost flow problems have a more general cost structure: The cost is a convex function of the amount of flow. Flow costs vary in a convex manner in numerous problem settings, including (1) power losses in an electrical network due to resistance, (2) congestion costs in a city transportation network, and (3) expansion costs of a communication network.

Generalized flow problems. In the minimum cost flow problem, arcs conserve flows (i.e., the flow entering an arc equals the flow leaving the arc). In generalized flow problems, arcs might "consume" or "generate" flow. If x_{ij} units of flow enter an arc (i, j), then $\mu_{ij}x_{ij}$ units arrive at node j; μ_{ij} is a positive *multiplier* associated with the arc. If $0 < \mu_{ij} < 1$, the arc is *lossy*, and if $1 < \mu_{ij} < \infty$, the arc is *gainy*. Generalized network flow problems arise in several application contexts: for example, (1) power transmission through electric lines, with power lost with distance traveled, (2) flow of water through pipelines or canals that lose water due to seepage or evaporation, (3) transportation of a perishable commodity, and (4) cash management scenarios in which arcs represent investment opportunities and multipliers represent appreciation or depreciation of an investment's value.

Multicommodity flow problems. The minimum cost flow problem models the flow of a single commodity over a network. Multicommodity flow problems arise when several commodities use the same underlying network. The commodities may either be differentiated by their physical characteristics or simply by their origin–destination pairs. Different commodities have different origins and destinations, and commodities have separate mass balance constraints at each node. However, the sharing of the common arc capacities binds the different commodities together. In fact, the essential issue addressed by the multicommodity flow problem is the allocation of the capacity of each arc to the individual commodities in a way that minimizes overall flow costs. Multicommodity flow problems arise in many practical situations, including (1) the transportation of passengers from different origins to different destinations within a city; (2) the routing of nonhomogeneous tankers (nonhomogeneous in terms of speed, carrying capability, and operating costs); (3) the worldwide shipment of different varieties of grains (such as corn, wheat, rice, and soybeans) from countries that produce grains to those that consume it; and (4) the transmission of messages in a communication network between different origin–destination pairs.

Other Models

In this book we also study two other important network models: the *minimum spanning tree problem* and the *matching problem*. Although these two models are not flow problems per se, because of their practical and mathematical significance and because of their close connection with several flow problems, we have included them as part of our treatment of network flows.

Minimum spanning tree problem. A spanning tree is a tree (i.e., a connected acyclic graph) that spans (touches) all the nodes of an undirected network. The cost of a spanning tree is the sum of the costs (or lengths) of its arcs. In the minimum spanning tree problem, we wish to identify a spanning tree of minimum cost (or length). The applications of the minimum spanning tree problem are varied and include (1) constructing highways or railroads spanning several cities; (2) laying pipelines connecting offshore drilling sites, refineries, and consumer markets; (3) designing local access networks; and (4) making electric wire connections on a control panel.

Matching problems. A *matching* in a graph $G = (N, A)$ is a set of arcs with the property that every node is incident to at most one arc in the set; thus a matching induces a pairing of (some of) the nodes in the graph using the arcs in A. In a matching, each node is matched with at most one other node, and some nodes might not be matched with any other node. The *matching problem* seeks a matching that optimizes some criteria. Matching problems on a bipartite graphs (i.e., those with two sets of nodes and with arcs that join only nodes between the two sets, as in the assignment and transportation problems) are called *bipartite matching problems*, and those on nonbipartite graphs are called *nonbipartite matching problems*. There are two additional ways of categorizing matching problems: *cardinality matching problems*, which maximize the number of pairs of nodes matched, and *weighted matching problems*, which maximize or minimize the weight of the matching. The weighted matching problem on a bipartite graph is also known as the *assignment problem*. Applications of matching problems arise in matching roommates to hostels, matching pilots to compatible airplanes, scheduling airline crews for available flight legs, and assigning duties to bus drivers.

1.3 APPLICATIONS

Networks are pervasive. They arise in numerous application settings and in many forms. Physical networks are perhaps the most common and the most readily identifiable classes of networks; and among physical networks, transportation networks are perhaps the most visible in our everyday lives. Often, these networks model homogeneous facilities such as railbeds or highways. But on other occasions, they correspond to composite entities that model, for example, complex distribution and logistics decisions. The traditional operations research "transportation problem" is illustrative. In the transportation problem, a shipper with inventory of goods at its warehouses must ship these goods to geographically dispersed retail centers, each with a given customer demand, and the shipper would like to meet these demands incurring the minimum possible transportation costs. In this setting, a transportation link in the underlying network might correspond to a complex distribution channel with, for example, a trucking shipment from the warehouse to a railhead, a rail shipment, and another trucking leg from the destination rail yard to the customer's site.

Physical networks are not limited to transportation settings; they also arise in several other disciplines of applied science and engineering, such as mathematics, chemistry, and electrical, communications, mechanical, and civil engineering. When physical networks occur in these different disciplines, their nodes, arcs, and flows model many different types of physical entities. For example, in a typical communication network, nodes will represent telephone exchanges and transmission facilities, arcs will denote copper cables or fiber optic links, and flow would signify the transmission of voice messages or of data. Figure 1.1 shows some typical associations for the nodes, arcs, and flows in a variety of physical networks.

Network flow problems also arise in surprising ways for problems that on the surface might not appear to involve networks at all. Sometimes these applications are linked to a physical entity, and at other times they are not. Sometimes the nodes and arcs have a temporal dimension that models activities that take place over time.

has non-negative cost by Theorem 4.10.1. Using that P is a minimum cost (s,t)-path in $\mathcal{N}(x)$, we conclude that each of R, Q has cost at least $c(P)$ implying that $c(P') \geq c(P)$. Hence (4.24) holds. □

4.10.3 The Assignment and the Transportation Problem

In this section we briefly discuss two special cases of the minimum cost flow problem, both of which occur frequently in practical applications. For a more detailed discussion see, e.g., [91, Section 3.12].

In the ASSIGNMENT PROBLEM, the input consists of a set of persons $P_1, P_2, \ldots, P_n$, a set of jobs $J_1, J_2, \ldots, J_n$ and an $n \times n$ matrix $M = [M_{ij}]$ whose entries are non-negative integers. Here M_{ij} is a measure for the skill of person P_i in performing job J_j (the lower the number the better P_i performs job J_j). The goal is to find an assignment π of persons to jobs so that each person gets exactly one job and the sum $\sum_{i=1}^{n} M_{i\pi(i)}$ is minimized. Given any instance of the assignment problem, we may form a complete bipartite graph $B = (U, V; E)$ where $U = \{P_1, P_2, \ldots, P_n\}$, $V = \{J_1, J_2, \ldots, J_n\}$ and E contains the edge P_iJ_j with the weight M_{ij} for each $i \in [m]$, $j \in [n]$. Now the assignment problem is equivalent to finding a minimum weight perfect matching in B. Clearly we can also go the other way and hence the assignment problem is equivalent to the WEIGHTED BIPARTITE MATCHING PROBLEM. It is also easy to see from this observation that the assignment problem is a (very) special case of the minimum cost flow problem. In fact, if we think of M_{ij} as a cost and orient all edges from U to V in the bipartite graph above, then what we are seeking is an integer-valued flow of minimum cost so that the value of the balance vector equals 1 for each P_i, $i = 1, 2, \ldots, m$, and equals -1 for each J_j, $j = 1, 2, \ldots, n$.

Inspecting the description of the buildup algorithm above, it is not hard to see that the following holds (Exercise 4.53).

Theorem 4.10.8 *The buildup algorithm solves the assignment problem for a bipartite graph on n vertices in time $\mathcal{O}(n^3)$.* □

In the TRANSPORTATION PROBLEM we are given a set of production plants $S_1, S_2, \ldots, S_m$ that produce a certain product to be shipped to a set of retailers $T_1, T_2, \ldots, T_n$. For each pair (S_i, T_j) there is a real-valued cost c_{ij} of transporting one unit of the product from S_i to T_j. Each plant produces a_i, $i = 1, 2, \ldots, m$, units per time unit and each retailer needs b_j, $j = 1, 2, \ldots, n$, units of the product per time unit. We assume below that $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$ (this is no restriction of the model as shown in Exercise 4.54). The goal is to find a transportation schedule for the whole production (i.e., how many units to send from S_i to T_j for $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$) in order to minimize the total transportation cost.

Again the transportation problem is easily seen to be a special case of the minimum cost flow problem. Consider a bipartite network $\mathcal{N}$ with partite sets

$S = \{S_1, S_2, \ldots, S_m\}$ and $T = \{T_1, T_2, \ldots, T_n\}$ and all possible arcs from S to T, where the capacity of the arc S_iT_j is ∞ and the cost of sending one unit of flow along S_iT_j is c_{ij}. Now it is easy to see that an optimal transportation schedule corresponds to a minimum cost flow in $\mathcal{N}$ with respect to the balance vectors

$$b(S_i) = a_i, i = 1, 2, \ldots, m, \text{ and } b(T_j) = -b_j, j = 1, 2, \ldots, n.$$

Again we could solve the transportation problem by the buildup algorithm but in this case we would not be guaranteed a polynomial running time since the running time would depend on the required balance values. Applying Theorem 4.10.4, we obtain a strongly polynomial algorithm for the problem. Clearly one would expect the existence of an algorithm of better complexity for the transportation problem (being a restricted version of the general minimum cost flow problem). Such an algorithm was given by Kleinschmidt and Schannath.

Theorem 4.10.9 *[600] The transportation problem with m suppliers and n consumers can be solved in time $\mathcal{O}(\min\{n, m\}(n + m)^2 \log(n + m))$.* □

For much more material on the assignment and transportation problems, including a survey of various complexities, the reader may consult Chapters 17 and 21 of Schrijver's book [803].

4.11 Applications of Flows

In this section we illustrate the applicability of flows to a large spectrum of problems both of theoretical and practical nature. For further applications see, e.g., Chapters 5, 13 and 17. Since we will need these results in later chapters the main focus is on finding certain substructures in digraphs.

4.11.1 Maximum Matchings in Bipartite Graphs

Let $G = (V, E)$ be an undirected graph. Recall that a matching in G is a set of edges from E, no two of which share a vertex, and a **maximum matching** of G is a matching of maximum cardinality among all matchings of G. Matching problems occur in many practical applications such as the following scheduling problem. We are given a set $T = \{t_1, t_2, \ldots, t_r\}$ of tasks (such as handling a certain machine) to be performed and a set $P = \{p_1, p_2, \ldots, p_s\}$ of persons, each of which is capable of performing some of the tasks from T. The goal is to find a maximum number of tasks such that each task can be performed by some person who does not at the same time perform any other task and no task is performed by more than one person. This can be formulated as a matching problem as follows. Let $B = (P, T; E)$ be the bipartite graph whose vertex set is $P \cup T$ and such that for each i, j such that

Chapter 8

Cutting Plane Algorithms

8.1 Valid Inequalities

Given an interger programming problem:

$$(IP): \quad z = \max\{\mathbf{c}^T\mathbf{x} : \mathbf{x} \in X\}, X = \{\mathbf{x} : A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^n_+\}$$

We have argued that if we knew the convex hull description of X then we could solve (IP) by linear programming. As we said, it can be proved that: $\text{conv}(X) = \{\mathbf{x} : \tilde{A}\mathbf{x} \leq \tilde{\mathbf{b}}, \mathbf{x} \geq 0\}$. That is, the convex hull can be described by the intersection of halfplanes and it is therefore a polyhedron. Hence the best formulation for (IP) would be:

$$LP: \quad z = \max\{\mathbf{c}^T\mathbf{x} : \tilde{A}\mathbf{x} \leq \tilde{\mathbf{b}}, \mathbf{x} \geq \mathbf{0}.\}$$

If this formulation is not knwon then we can try to approximate it.

Definition 8.1 (Valid inequalities)**.** An inequality $\mathbf{ax} \leq \mathbf{b}$ is a **valid inequality** for $X \subseteq \mathbb{R}^n$ if $\mathbf{ax} \leq \mathbf{b}$, for all $\mathbf{x} \in X$

Which are useful inequalities? How can we find them? How can we use them?
The following examples show how inequalities can be deduced in a pre-processing stage.

Example 8.1.

$$X = \{(x, y) : x \leq 999y; 0 \leq x \leq 5, y \in \mathbb{B}^1\}$$

The constraint

$$x \leq 5y$$

is implied by the formulation above. Indeed if y is 0 then $x \leq 0$, if y is 1 then $x \leq 5$. Hence x will never be larger than 5.

Example 8.2. $X = \{\mathbf{x} \in \mathbb{Z}^n_+ : 13x_1 + 20x_2 + 11x_3 + 6x_4 \geq 72\}$
Dividing left and right hand side by 11 we get:

$$2x_1 + 2x_2 + x_3 + x_4 \geq \frac{13}{11}x_1 + \frac{20}{11}x_2 + x_3 + \frac{6}{11}x_4 \geq \frac{72}{11} = 6 + \frac{6}{11}$$

Since x_1, x_2, x_3, x_4 must be integer then the constraint above implies:

$$2x_1 + 2x_2 + x_3 + x_4 \geq 7.$$

Example 8.3. Capacitated facility location:

$$\sum_{i \in M} x_{ij} \leq b_j y_j \quad \forall j \in N \qquad\qquad x_{ij} \leq b_j y_j$$

$$\sum_{j \in N} x_{ij} = a_i \quad \forall i \in M \qquad\qquad x_{ij} \leq a_i$$

$$x_{ij} \geq 0, y_j \in B^n \qquad\qquad x_{ij} \leq \min\{a_i, b_j\} y_j$$

8.2 Cutting Plane Algorithms for Integer Programming

8.2.1 Chvátal-Gomory cuts

Let $X \in P \cap \mathbb{Z}^n_+$ be a feasibility region with $P = \{\mathbf{x} \in \mathbb{R}^n_+ : A\mathbf{x} \leq \mathbf{b}\}, A \in \mathbb{R}^{m\times n}$. Let also $\mathbf{u} \in \mathbb{R}^m_+$ be a vector of multipliers. Further, recall the notation $\{\mathbf{a}_1, \mathbf{a}_2, \dots \mathbf{a}_n\}$ to represent the columns of A.

The following procedure for cut generation due to Chvátal and Gomory generates valid inequalities:

1)	$\sum_{j=1}^{n} \mathbf{u}\mathbf{a}_j x_j \leq \mathbf{u}\mathbf{b}$	valid: $\mathbf{u} \geq \mathbf{0}$
2)	$\sum_{j=1}^{n} \lfloor \mathbf{u}\mathbf{a}_j \rfloor x_j \leq \mathbf{u}\mathbf{b}$	valid: $\mathbf{x} \geq \mathbf{0}$ and $\sum \lfloor \mathbf{u}\mathbf{a}_j \rfloor x_j \leq \sum \mathbf{u}\mathbf{a}_j x_j$
3)	$\sum_{j=1}^{n} \lfloor \mathbf{u}\mathbf{a}_j \rfloor x_j \leq \lfloor \mathbf{u}\mathbf{b} \rfloor$	valid for X since $\mathbf{x} \in \mathbb{Z}^n$

Theorem 8.1. *Applying this CG procedure a finite number of times every valid inequality for X can be obtained.*

8.2.2 Cutting Plane Algorithms

Let $X \in P \cap \mathbb{Z}^n_+$ be a feasible region for which we are given a family of valid inequalities $\mathcal{F} : \mathbf{a}^T\mathbf{x} \leq b, (\mathbf{a}, b) \in \mathcal{F}$. We do not find and use them all a priori. We are only interested in those close to optimum. We achieve this with the following procedure:

Init.: $t = 0, P^0 = P$

Iter. t**:** Solve $\bar{z}^t = \max\{\mathbf{c}^T\mathbf{x} : \mathbf{x} \in P^t\}$
let $\mathbf{x}^t$ be an optimal solution
if $\mathbf{x}^t \in \mathbb{Z}^n$ stop, $\mathbf{x}^t$ is opt to the IP
if $\mathbf{x}^t \notin \mathbb{Z}^n$ solve separation problem for $\mathbf{x}^t$ and $\mathcal{F}$
if $(\mathbf{a}^t, b^t)$ is found with $\mathbf{a}^t\mathbf{x}^t > b^t$ that cuts off x^t

$$P^{t+1} = P \cap \{\mathbf{x} : \mathbf{a}^i\mathbf{x} \leq b^i, i = 1, \dots, t\}$$

else stop (P^t is in any case an improved formulation)

8.2.3 Gomory's fractional cutting plane algorithm

Cutting plane algorithm + Chvátal-Gomory cuts

- $\max\{\mathbf{c}^T\mathbf{x} : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0, \mathbf{x} \in \mathbb{Z}^n\}$
- Solve LPR to optimality

$$\left[\begin{array}{c|c|c|c} I & \bar{A}_N = A_B^{-1}A_N & 0 & \bar{b} \\ \hline \bar{c}_B & \bar{c}_N (\leq 0) & 1 & -\bar{d} \end{array}\right]$$

$$x_u = \bar{b}_u - \sum_{j\in N} \bar{a}_{uj} x_j, \quad u \in B$$
$$z = \bar{d} + \sum_{j\in N} \bar{c}_j x_j$$

- If basic optimal solution to LPR is not integer then $\exists$ some row u: $\bar{b}_u \notin \mathbb{Z}^1$.
  The Chvatál-Gomory cut applied to this row is:

$$x_{B_u} + \sum_{j \in N} \lfloor \bar{a}_{uj} \rfloor x_j \leq \lfloor \bar{b}_u \rfloor$$

  (B_u is the index in the basis B corresponding to the row u)

- Eliminating $x_{B_u} = \bar{b}_u - \sum_{j \in N} \bar{a}_{uj} x_j$ in the CG cut we obtain:

$$\sum_{j \in N} (\underbrace{\bar{a}_{uj} - \lfloor \bar{a}_{uj} \rfloor}_{0 \leq f_{uj} < 1}) x_j \geq \underbrace{\bar{b}_u - \lfloor \bar{b}_u \rfloor}_{0 < f_u < 1}$$

$$\sum_{j \in N} f_{uj} x_j \geq f_u$$

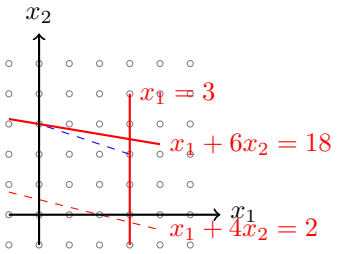
  $f_u > 0$ or else u would not be the row of a fractional solution. It implies that x^* in which $x^*_N = 0$ is cut out!

- Moreover: when x is integer, since all coefficient in the CG cut are integer the slack variable of the cut is also integer:

$$s = -f_u + \sum_{j \in N} f_{uj} x_j$$

(theoretically it terminates after a finite number of iterations, but in practice it is not successful.)

$$\begin{array}{rrcl} \max & x_1 + 4x_2 & & \\ & x_1 + 6x_2 & \leq & 18 \\ & x_1 & \leq & 3 \\ & x_1, x_2 & \geq & 0 \\ & x_1, x_2 & \text{integer} & \end{array}$$




```
|   | x1 | x2 | x3 | x4 | -z |  b |
|---+----+----+----+----+----+----|
|   |  1 |  6 |  1 |  0 |  0 | 18 |
|   |  1 |  0 |  0 |  1 |  0 |  3 |
|---+----+----+----+----+----+----|
|   |  1 |  4 |  0 |  0 |  1 |  0 |

|   | x1 | x2 | x3 | x4 | -z |  b |
|---+----+----+----+----+----+----|
|   |  0 |  6 |  1 | -1 |  0 | 15 |
|   |  1 |  0 |  0 |  1 |  0 |  3 |
|---+----+----+----+----+----+----|
|   |  0 |  4 |  0 | -1 |  1 | -3 |

|   | x1 | x2 | x3    | x4    | -z |    b |
|---+----+----+-------+-------+----+------|
|   |  0 |  1 | 1/6   | -1/6  |  0 | 15/6 |
|   |  1 |  0 | 0     | 1     |  0 |    3 |
|---+----+----+-------+-------+----+------|
|   |  0 |  0 | -2/3  | -1/3  |  1 |  -13 |
```

$x_2 = 5/2, x_1 = 3$
Optimum, not integer

- We take the first row: `|  | 0 | 1 | 1/6 | -1/6 | 0 | 15/6 |`
- CG cut $\sum_{j \in N} f_{uj} x_j \geq f_u \rightsquigarrow \frac{1}{6}x_3 + \frac{5}{6}x_4 \geq \frac{1}{2}$
- Let's see that it leaves out x^*: from the CG proof:

$$\begin{array}{rrcl} 1/6 & (x_1 + 6x_2 & \leq & 18) \\ 5/6 & (x_1 & \leq & 3) \\ \hline & x_1 + x_2 & \leq & 3 + 5/2 = 5.5 \end{array}$$

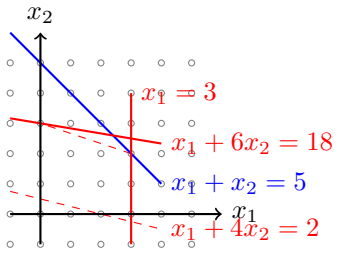
  since x_1, x_2 are integers $x_1 + x_2 \leq 5$

- Let's see how it looks in the space of the original variables: from the first tableau:

$$\begin{aligned} x_3 &= 18 - 6x_2 - x_1 \\ x_4 &= 3 - x_1 \end{aligned}$$

$$\frac{1}{6}(18 - 6x_2 - x_1) + \frac{5}{6}(3 - x_1) \geq \frac{1}{2} \qquad \rightsquigarrow \qquad x_1 + x_2 \leq 5$$

- Graphically:



- Let's continue:

```
|   | x1 | x2 | x3   | x4   | x5 | -z | b    |
|---+----+----+------+------+----+----+------|
|   |  0 |  0 | -1/6 | -5/6 |  1 |  0 | -1/2 |
|   |  0 |  1 | 1/6  | -1/6 |  0 |  0 | 5/2  |
|   |  1 |  0 | 0    | 1    |  0 |  0 | 3    |
|---+----+----+------+------+----+----+------|
|   |  0 |  0 | -2/3 | -1/3 |  0 |  1 | -13  |
```

  We need to apply dual-simplex
  (will always be the case, why?)

  ratio rule: $\min |\frac{c_j}{a_{ij}}|$

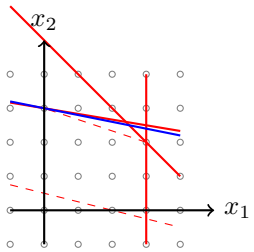
- After the dual simplex iteration:

```
|   | x1 | x2 | x3   | x4 | x5   | -z | b     |
|---+----+----+------+----+------+----+-------|
|   |  0 |  0 | 1/5  |  1 | -6/5 |  0 | 3/5   |
|   |  0 |  1 | 1/5  |  0 | -1/5 |  0 | 13/5  |
|   |  1 |  0 | -1/5 |  0 | 6/5  |  0 | 12/5  |
|---+----+----+------+----+------+----+-------|
|   |  0 |  0 | -3/5 |  0 | -2/5 |  1 | -64/5 |
```

We can choose any of the three rows.
Let's take the third: CG cut: $\frac{4}{5}x_3 + \frac{1}{5}x_5 \geq \frac{2}{5}$

- In the space of the original variables:

$$4(18 - x_1 - 6x_2) + (5 - x_1 - x_2) \geq 2$$
$$x_1 + 5x_2 \leq 15$$



- ...

8

Cutting Plane Algorithms

8.1 INTRODUCTION

Here we consider the general integer program:

$$(IP) \qquad \max\{cx : x \in X\}$$

where $X = \{x : Ax \leq b, x \in Z_+^n\}$.

Proposition 8.1 *$conv(X) = \{x : \tilde{A}x \leq \tilde{b}, x \geq 0\}$ is a polyhedron.*

This result, already presented in Chapter 1, tells us that we can, in theory, reformulate problem IP as the linear program:

$$(LP) \qquad \max\{cx : \tilde{A}x \leq \tilde{b}, x \geq 0\}$$

and then for any value of c, an optimal extreme point solution of LP is an optimal solution of IP. The same result holds for mixed integer programs with $X = \{(x, y) \in R_+^n \times Z_+^p : Ax + Gy \leq b\}$ provided the data A, G, b are rational.

In Chapter 3 we have seen several problems, including the assignment problem and the spanning tree problem, for which we have given an explicit description of conv(X). However, unfortunately for $\mathcal{NP}$-hard problems, there is almost no hope of finding a "good" description. Given an instance of an $\mathcal{NP}$-hard problem, the goal in this chapter is to find effective ways to try and approximate conv(X) for the given instance.

The fundamental concept that we have already used informally is that of a valid ine ality.

Definition 8.1 An inequality $\pi x \leq \pi_0$ is a *valid inequality* for $X \subseteq R^n$ if $\pi x \leq \pi_0$ for all $x \in X$.

If $X = \{x \in Z^n : Ax \leq b\}$ and $\text{conv}(X) = \{x \in R^n : \tilde{A}x \leq \tilde{b}\}$, the constraints $a^i x \leq b_i$ and $\tilde{a}^i x \leq \tilde{b}_i$ are clearly valid inequalities for X.

The two questions that immediately spring to mind are

(i) Which are the "good" or useful valid inequalities? and

(ii) If we know a set or family of valid inequalities for a problem, how can we use them in trying to solve a particular instance?

8.2 SOME SIMPLE VALID INEQUALITIES

First we present some examples of valid inequalities. The first type, logical inequalities, have already been seen in Example 7.5 in looking at preprocessing.

Example 8.1 A Pure 0–1 Set. Consider the 0–1 knapsack set:

$$X = \{x \in B^5 : 3x_1 - 4x_2 + 2x_3 - 3x_4 + x_5 \leq -2\}.$$

If $x_2 = x_4 = 0$, the lhs (left-hand side) $= 3x_1 + 2x_3 + x_5 \geq 0$ and the rhs (right-hand side) $= -2$, which is impossible. So all feasible solutions satisfy the valid inequality $x_2 + x_4 \geq 1$.

If $x_1 = 1$ and $x_2 = 0$, the lhs $= 3 + 2x_3 - 3x_4 + x_5 \geq 3 - 3 = 0$ and the rhs $= -2$, which is impossible, so $x_1 \leq x_2$ is also a valid inequality. ∎

Example 8.2 A Mixed 0–1 Set. Consider the set:

$$X = \{(x, y) : x \leq 9999y, 0 \leq x \leq 5, y \in B^1\}.$$

It is easily checked that the inequality

$$x \leq 5y$$

is valid because $X = \{(0,0), (x,1)$ with $0 \leq x \leq 5\}$. As X only involves two variables, it is possible to represent X graphically, so it is easy to check that the addition of the inequality $x \leq 5y$ gives us the convex hull of X.

Such constraints arise often. For instance, in the capacitated facility location problem one has the feasible region:

$$\sum_{i \in M} x_{ij} \leq b_j y_j \text{ for } j \in N$$
$$\sum_{j \in N} x_{ij} = a_i \text{ for } i \in M$$
$$x_{ij} \geq 0 \text{ for } i \in M, j \in N, y_j \in \{0,1\} \text{ for } j \in N.$$

All feasible solutions satisfy $x_{ij} \leq b_j y_j$ and $x_{ij} \leq a_i$ with $y_j \in B^1$. This is precisely the situation above leading to the family of valid inequalities $x_{ij} \leq \min\{a_i, b_j\} y_j$. ∎

Example 8.3 A Mixed Integer Set. Consider the set

$$X = \{(x, y) : x \leq 10y, 0 \leq x \leq 14, y \in Z^1_+\}.$$

It is not difficult to verify the validity of the inequality $x \leq 6 + 4y$, or written another way, $x \leq 14 - 4(2 - y)$. In Figure 8.1 we represent X graphically, and see that the addition of the inequality $x \leq 6 + 4y$ gives the convex hull of X.

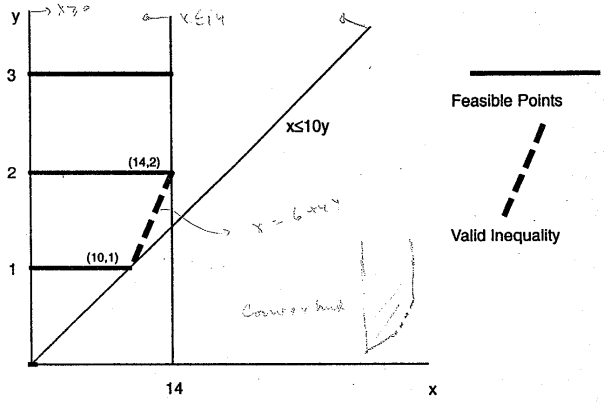


Fig. 8.1 Mixed integer inequality

For the general case, when C does not divide b, and

$$X = \{(x, y) : x \leq Cy, 0 \leq x \leq b, y \in Z^1_+\},$$

one obtains the valid inequality $x \leq b - \gamma(K - y)$ where $K = \lceil \frac{b}{C} \rceil$ and $\gamma = b - (\lceil \frac{b}{C} \rceil - 1)C$. ∎

Example 8.4 A Combinatorial Set: Matchings. Consider the X of incidence vectors of matchings:

$$\sum_{e \in \delta(i)} x_e \leq 1 \text{ for } i \in V \tag{8.1}$$
$$x \in Z^{|E|}_+ \tag{8.2}$$

where $\delta(i) = \{e \in E : e = (i, j) \text{ for some } j \in V\}$.

Take a set $T \subseteq V$ of nodes of odd cardinality. As the edges of a matching are disjoint, the number of edges of a matching having both endpoints in T is at most $\frac{|T|-1}{2}$. Therefore

$$\sum_{e \in E(T)} x_e \leq \frac{|T|-1}{2} \tag{8.3}$$

is a valid inequality for X if $|T| \geq 3$ and $|T|$ is odd. ∎

Example 8.5 Integer Rounding. Consider the integer region $X = P \cap Z^4$ where

$$P = \{x \in R^4_+ : 13x_1 + 20x_2 + 11x_3 + 6x_4 \geq 72\}.$$

Dividing by 11 gives the valid inequality for P:

$$\tfrac{13}{11}x_1 + \tfrac{20}{11}x_2 + x_3 + \tfrac{6}{11}x_4 \geq 6\tfrac{6}{11}.$$

As $x \geq 0$, rounding up the coefficients on the left to the nearest integer gives $2x_1 + 2x_2 + x_3 + x_4 \geq \frac{13}{11}x_1 + \frac{20}{11}x_2 + x_3 + \frac{6}{11}x_4 \geq 6\frac{6}{11}$, and so we get a weaker valid inequality for P:

$$2x_1 + 2x_2 + x_3 + x_4 \geq 6\tfrac{6}{11}.$$

As x is integer and all the coefficients are integer, the lhs must be integer. An integer that is greater than or equal to $6\frac{6}{11}$ must be at least 7, and so we can round the rhs up to the nearest integer giving the valid inequality for X:

$$2x_1 + 2x_2 + x_3 + x_4 \geq 7.$$ ∎

Such regions arise in many problems. Consider, for instance, a *Generalized Transportation Problem* where the problem is to satisfy the demand d_j of client j using trucks of different types. A truck of type i has capacity C_i, there are a_i of them available, and the cost if a truck of type i is sent to client j is c_{ij}. The resulting integer program is:

$$\begin{aligned} \min \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & \\ \sum_{i=1}^{n} C_i x_{ij} &\geq d_j \text{ for } j = 1,\ldots,n \\ \sum_{j=1}^{n} x_{ij} &\leq a_i \text{ for } i = 1,\ldots,m \\ x &\in Z^{mn}_+, \end{aligned}$$

where each demand constraint gives rise to a cut of the form X.

Example 8.6 Mixed Integer Rounding. Consider the same example as with the addition of a continuous variable. Let $X = P \cap (Z^4 \times R^1)$ wh

$$P = \{(y, s) \in R^4_+ \times R^1_+ : 13y_1 + 20y_2 + 11y_3 + 6y_4 + s \geq 72\}.$$

In terms of the generalized transportation model, there are four types o available to satisfy demand, but it is also possible to satisfy demand fr alternative source. Dividing by 11 gives

$$\tfrac{13}{11}y_1 + \tfrac{20}{11}y_2 + y_3 + \tfrac{6}{11}y_4 \geq \tfrac{72-s}{11},$$

suggesting that there is a valid inequality

$$2y_1 + 2y_2 + y_3 + y_4 + \alpha s \geq 7 \text{ for some } \alpha.$$

Looking at the rhs term $\frac{72-s}{11}$, we see that the rhs $\lceil \frac{72-s}{11} \rceil$ decreases to 6 at the critical value $s = 6$, indicating the value $\alpha = \frac{1}{6}$. Inequalit turns out to be valid for values of $\alpha \geq \frac{1}{6}$, and later we will see that even be strengthened a little, giving:

$$\tfrac{3}{2}y_1 + 2y_2 + y_3 + y_4 + \tfrac{1}{6}s \geq 7.$$

8.3 VALID INEQUALITIES

To understand how to generate valid inequalities for integer program first necessary to understand valid inequalities for polyhedra (or line grams).

8.3.1 Valid Inequalities for Linear Programs

So the first question is: When is the inequality $\pi x \leq \pi_0$ valid for P $Ax \leq b, x \geq 0\}$?

Proposition 8.2 *$\pi x \leq \pi_0$ is valid for $P = \{x : Ax \leq b, x \geq 0\} \neq \emptyset$ only if*
there exist $u \geq 0, v \geq 0$ such that $uA - v = \pi$ and $ub \leq \pi_0$, or alterna
there exists $u \geq 0$ such that $uA \geq \pi$ and $ub \leq \pi_0$.

Proof. By linear programming duality, $\max\{\pi x : x \in P\} \leq \pi_0$ if and $\min\{ub : uA - v = \pi, u \geq 0, v \geq 0\} \leq \pi_0$.

8.3.2 Valid Inequalities for Integer Programs

Now we consider the feasible region of an integer program:

$$\{x : Ax \leq b, x \in Z_+^n\}$$

and ask the same question.

Surprisingly, the complete answer is in some sense given in the following very simple observation.

Proposition 8.3 *Let $X = \{y \in Z^1 : y \leq b\}$, then the inequality $y \leq \lfloor b \rfloor$ is valid for X.*

We have already used this idea in Example 8.5. We now give two more examples.

Example 8.4 (cont) Valid Inequalities for Matching. Here we give an alternative algebraic justification for the validity of inequality (8.3) that can be broken up into three steps.
(i) Take a nonnegative linear combination of the constraints (8.1) with weights $u_i = \frac{1}{2}$ for $i \in T$ and $u_i = 0$ for $i \in V \setminus T$. This gives the valid inequality:

$$\sum_{e \in E(T)} x_e + \frac{1}{2} \sum_{e \in \delta(T, V \setminus T)} x_e \leq \frac{|T|}{2}.$$

(ii) Because $x_e \geq 0$, $\sum_{e \in E(T)} x_e \leq \sum_{e \in E(T)} x_e + \frac{1}{2} \sum_{e \in \delta(T, V \setminus T)} x_e$, and so

$$\sum_{e \in E(T)} x_e \leq \frac{|T|}{2}$$

is a valid inequality.
(iii) Because $x \in Z^n$, the lhs $\sum_{e \in E(T)} x_e$ must be an integer, and so one can replace the rhs value by the largest integer less than or equal to the rhs value. So

$$\sum_{e \in E(T)} x_e \leq \lfloor \frac{|T|}{2} \rfloor$$

is a valid inequality. ∎

Example 8.7 Valid Inequalities for an Integer Program. An identical approach can be used to derive valid inequalities for any integer programming region. Let $X = P \cap Z^n$ be the set of integer points in P where P is given by:

$$\begin{aligned} 7x_1 - 2x_2 &\leq 14 \\ x_2 &\leq 3 \\ 2x_1 - 2x_2 &\leq 3 \\ x &\geq 0. \end{aligned}$$

(i) First combining the constraints with nonnegative weights $u = (\frac{2}{7}, \frac{37}{63}, 0)$ we obtain the valid inequality for P

$$2x_1 + \frac{1}{63}x_2 \leq \frac{121}{21}.$$

(ii) Reducing the coefficients on the left-hand side to the nearest integer give the valid inequality for P:

$$2x_1 + 0x_2 \leq \frac{121}{21}.$$

(iii) Now as the left-hand side is integral for all points of X, we can reduc the rhs to the nearest integer, and we obtain the valid inequality for X:

$$2x_1 \leq \lfloor \frac{121}{21} \rfloor = 5.$$

Observe that if we repeat the procedure, and use a weight of $\frac{1}{2}$ on this las constraint, we obtain the tighter inequality $x_1 \leq \lfloor \frac{5}{2} \rfloor = 2$.

Now it is easy to describe the general procedure that we have been using.

Chvátal-Gomory procedure to construct a valid inequality for the se $X = P \cap Z^n$, where $P = \{x \in R_+^n : Ax \leq b\}$, A is an $m \times n$ matrix wit columns $\{a_1, a_2, \ldots, a_n\}$, and $u \in R_+^m$:

(i) the inequality

$$\sum_{j=1}^{n} u a_j x_j \leq ub$$

is valid for P as $u \geq 0$ and $\sum_{j=1}^{n} a_j x_j \leq b$,

(ii) the inequality

$$\sum_{j=1}^{n} \lfloor u a_j \rfloor x_j \leq ub$$

is valid for P as $x \geq 0$,

(iii) the inequality

$$\sum_{j=1}^{n} \lfloor u a_j \rfloor x_j \leq \lfloor ub \rfloor$$

is valid for X as x is integer, and thus $\sum_{j=1}^{n} \lfloor u a_j \rfloor x_j$ is integer.

The surprising fact is that this simple procedure is sufficient to generat all valid inequalities for an integer program.

Theorem 8.4 *Every valid inequality for X can be obtained by applying the Chvátal-Gomory procedure a finite number of times.*

Proof.* We present a proof for the 0–1 case. Thus let $P = \{x \in R^n : Ax \leq b, \mathbf{0} \leq x \leq \mathbf{1}\} \neq \emptyset, X = P \cap Z^n$, and suppose that $\pi x \leq \pi_0$ with π, π_0 integral is a valid inequality for X. We will show that this inequality is a *C-G inequality*, or in other words that it can be obtained by applying the Chvátal-Gomory procedure a finite number of times.

Claim 1 The inequality $\pi x \leq \pi_0 + t$ is a valid inequality for P for some $t \in Z^1_+$.
Proof. $z_{LP} = \max\{cx : x \in P\}$ is bounded as P is bounded. Take $t = \lceil z_{LP} \rceil - \pi_0$.

Claim 2 There exists a sufficiently large integer M that the inequality

$$\pi x \leq \pi_0 + M \sum_{j \in N^0} x_j + M \sum_{j \in N^1} (1 - x_j) \tag{8.5}$$

is valid for P for every partition (N^0, N^1) of N.
Proof. It suffices to show that the inequality is satisfied at all vertices x^* of P. If $x^* \in Z^n$, then the inequality $\pi x \leq \pi_0$ is satisfied, and so (8.5) is satisfied. Otherwise there exists $\alpha > 0$ such that $\sum_{j \in N^0} x^*_j + \sum_{j \in N^1} (1 - x^*_j) \geq \alpha$ for all partitions (N^0, N^1) of N and all non-integral extreme points x^* of P. Taking $M \geq \frac{t}{\alpha}$, it follows that for all extreme points of P,

$$\pi x^* \leq \pi_0 + t \leq \pi_0 + M \sum_{j \in N^0} x^*_j + M \sum_{j \in N^1} (1 - x^*_j).$$

Claim 3 If $\pi x \leq \pi_0 + \tau + 1$ is a C-G inequality for X with $\tau \in Z^1_+$, then

$$\pi x \leq \pi_0 + \tau + \sum_{j \in N^0} x_j + \sum_{j \in N^1} (1 - x_j) \tag{8.6}$$

is a C-G inequality for X.
Proof. Take the inequality $\pi x \leq \pi_0 + \tau + 1$ with weight $(M-1)/M$ and the inequality (8.5) with weight $1/M$. The resulting C-G inequality is (8.6).

Claim 4 If

$$\pi x \leq \pi_0 + \tau + \sum_{j \in T^0 \cup \{p\}} x_j + \sum_{j \in T^1} (1 - x_j) \tag{8.7}$$

and

$$\pi x \leq \pi_0 + \tau + \sum_{j \in T^0} x_j + \sum_{j \in T^1 \cup \{p\}} (1 \quad x_j) \tag{8.8}$$

are C-G inequalities for X where (T^0, T^1) is any partition of $\{1, \ldots, p-1\}$ then

$$\pi x \leq \pi_0 + \tau + \sum_{j \in T^0} x_j + \sum_{j \in T^1} (1 - x_j) \tag{8.9}$$

is a C-G inequality for X.
Proof. Take the inequalities (8.7) and (8.8) with weights 1/2 and the resulting C-G inequality is (8.9).

Claim 5 If

$$\pi x \leq \pi_0 + \tau + 1$$

is a C-G inequality for X, then

$$\pi x \leq \pi_0 + \tau$$

is a C-G inequality for X.
Proof. Apply Claim 4 successively for $p = n, n-1, \ldots, 1$ with all partition (T^0, T^1) of $\{1, \ldots, p-1\}$.

Finally starting with $\tau = t-1$ and using Claim 5 for $\tau = t-1, \cdots, 0$ establishe that $\pi x \leq \pi_0$ is a C-G inequality.

For inequalities generated by other arguments, it is sometimes interesting t see how easily they are generated as C-G inequalities, see Exercise 8.15.

Now that we have seen a variety of both ad hoc and general ways to deriv valid inequalities, we turn to the important practical question of how to us them.

8.4 A PRIORI ADDITION OF CONSTRAINTS

In discussing branch-and-bound we saw that preprocessing was a first ste in tightening a formulation. Here we go a step further. The idea here is t examine the initial formulation $P = \{x : Ax \leq b, x \geq 0\}$ with $X = P \cap Z^n$ find a set of valid inequalities $Qx \leq q$ for X, add these to the formulatio immediately giving a new formulation $P' = \{x : Ax \leq b, Qx \leq q, x \geq 0\}$ wit $X = P' \cap Z^n$. Then one can apply one's favorite algorithm, Branch-and-Boun or whatever, to formulation P'.

Advantages. One can use standard branch-and-bound software. If the vali inequalities are well chosen so that formulation P' is significantly smaller tha P, the bounds should be improved and hence the branch-and-bound algorith should be more effective. In addition the chances of finding feasible integ solution in the course of the algorithm should increase.

Disadvantages. Often the family of valid inequalities one would like to add is enormous. In such cases either the linear programs become very big and take a long time to solve, or it becomes impossible to use standard branch-and-bound software because there are too many constraints.

How can one start looking for valid inequalities a priori? In many instances the feasible region X can be written naturally as the intersection of two or more sets with structure, that is, $X = X^1 \cap X^2$. *Decomposing* or concentrating on one of the sets at a time may be a good idea.

For instance, we may know that the optimization problem over the set $X^2 = P^2 \cup Z^n$ is easy. Then we may be able to find an explicit description of $P'^2 = \text{conv}(P^2 \cap Z^n)$. In this case we can replace the initial formulation $P^1 \cap P^2$ by an improved formulation $P' = P^1 \cap P'^2$.

Whether the optimization problem over X^2 is easy or not, one may be able to take advantage of the structure to find valid inequalities for X^2, which allow us to improve its formulation from P^2 to $P'^2 \subset P^2$, and thereby again provide an improved formulation $P' = P^1 \cap P'^2$ for X.

Example 8.8 Uncapacitated Facility Location. Take the "weak" formulation used in the 1950s and 1960s:

$$\begin{aligned} \sum_{j=1}^{n} x_{ij} &= 1 \text{ for } i = 1, \dots, m \\ \sum_{i=1}^{m} x_{ij} &\le m y_j \text{ for } j = 1, \dots, n \\ x_{ij} &\ge 0 \text{ for } i = 1, \dots, m,\ j = 1, \dots, n \\ 0 &\le y_j \le 1 \text{ for } j = 1, \dots, n. \end{aligned}$$

Let X_j be the set of points in the polyhedron P_j :

$$\begin{aligned} \sum_{i=1}^{m} x_{ij} &\le m y_j \\ x_{ij} &\ge 0 \text{ for } i = 1, \dots, m \\ 0 &\le y_j \le 1, \end{aligned}$$

with y_j integer. The convex hull P'_j of the set X_j is given by

$$\begin{aligned} x_{ij} &\le y_j \text{ for } i = 1, \dots, m \\ x_{ij} &\ge 0 \text{ for } i = 1, \dots, m \\ 0 &\le y_j \le 1. \end{aligned}$$

Now [illegible] reformulation obtained by replacing P_j by P'_j for $j = 1, \dots, n$ is th "strong" formulation P':

$$\begin{aligned} \sum_{j=1}^{n} x_{ij} &= 1 \text{ for } i = 1, \dots, m \\ x_{ij} &\le y_j \text{ for } i = 1, \dots, m\ j = 1, \dots, n \\ x_{ij} &\ge 0 \text{ for } i = 1, \dots, m,\ j = 1, \dots, n \\ 0 \le y_j &\le 1 \text{ for } j = 1, \dots, n. \end{aligned}$$

The strong formulation is now commonly used for this problem because t bound it provides is much stronger than that given by the weak formulatio and the linear programming relaxation has solutions that are close to bei integral.

Example 8.9 Constant Capacity Lot-Sizing. Using the same notation as *ULS* introduced in Section 1.4, a basic formulation of the feasible region X

$$\begin{aligned} s_{t-1} + x_t &= d_t + s_t \text{ for } t = 1, \dots, n \\ x_t &\le C y_t \text{ for } t = 1, \dots, n \\ s_0 = s_n = 0, s &\in R^{n+1}_+, x \in R^n, y \in B^n. \end{aligned}$$

We derive two families of inequalities for X. First consider any point $(x, s, y$ X. First as $s_{t-1} \ge 0$, the inequality $x_t \le d_t + s_t$ clearly holds. Along w $x_t \le C y_t$ and $y_t \in \{0, 1\}$, it is then not difficult to show that $x_t \le d_t y_t +$ is valid for X. (Note that without the variable s_t, this is precisely the mi 0–1 inequality of Example 8.2).

Second, summing the flow conservation constraints, and using $s_t \ge 0$, get the inequality $\sum_{i=1}^{t} x_i \ge \sum_{i=1}^{t} d_i$. Then using $x_i \le C y_i$ gives $C \sum_{i=1}^{t} y$ $\sum_{i=1}^{t} d_i$, or $\sum_{i=1}^{t} y_i \ge (\sum_{i=1}^{t} d_i)/C$. Now as $\sum_{i=1}^{t} y_i$ is integral, we can Chvátal-Gomory integer rounding to obtain the valid inequality

$$\sum_{i=1}^{t} y_i \ge \lceil \frac{\sum_{i=1}^{t} d_i}{C} \rceil.$$

Adding just these $2n$ inequalities significantly strengthens the formulat of this problem.

8.5 AUTOMATIC REFORMULATION OR CUTTING PLANE ALGORITHMS

Suppose that $X = P \cap Z^n$ and that we know a family $\mathcal{F}$ of valid inequali $\pi x \le \pi_0, (\pi, \pi_0) \in \mathcal{F}$ for X. In many cases $\mathcal{F}$ contains too many inequali

(2^n or more) for them to be added a priori. Also given a specific objective function, one is not really interested in finding the complete convex hull, but one wants a good approximation to it in the neighborhood of the optimal solution.

We now describe a basic cutting plane algorithm for (IP), $\max\{cx : x \in X\}$, that generates "useful" inequalities from $\mathcal{F}$.

Cutting Plane Algorithm

Initialization. Set $t = 0$ and $P^0 = P$.
Iteration t. Solve the linear program:

$$\overline{z}^t = \max\{cx : x \in P^t\}.$$

Let x^t be an optimal solution.
If $x^t \in Z^n$, stop. x^t is an optimal solution for IP.
If $x^t \notin Z^n$, solve the separation problem for x^t and the family $\mathcal{F}$.
If an inequality $(\pi^t, \pi_0^t) \in \mathcal{F}$ is found with $\pi^t x^t > \pi_0^t$ so that it cuts off x^t, set $P^{t+1} = P^t \cap \{x : \pi^t x \leq \pi_0^t\}$, and augment t. Otherwise stop.

If the algorithm terminates without finding an integral solution for IP,

$$P^t = P \cap \{x : \pi^i x \leq \pi_0^i \quad i = 1, \ldots, t\}$$

is an improved formulation that can be input to a branch-and-bound algorithm. It should also be noted that in practice it is often better to add several violated cuts at each iteration, and not just one at a time.

In the next section we look at a specific implementation of this algorithm.

8.6 GOMORY'S FRACTIONAL CUTTING PLANE ALGORITHM

Here we consider the integer program:

$$\max\{cx : Ax = b, x \geq 0 \text{ and integer}\}.$$

The idea is to first solve the associated linear programming relaxation and find an optimal basis, choose a basic variable that is not integer, and then generate a Chvátal-Gomory inequality on the constraint associated with this basic variable so as to cut off the linear programming solution. We suppose, given an optimal basis, that the problem is rewritten in the form:

$$\begin{gathered} \max \overline{a}_{00} + \textstyle\sum_{j \in NB} \overline{a}_{0j} x_j \\ x_{B_u} + \textstyle\sum_{j \in NB} \overline{a}_{uj} x_j = \overline{a}_{u0} \text{ for } u = 1, \ldots, m \\ x \geq 0 \text{ and integer} \end{gathered}$$

with $\overline{a}_{0j} \leq 0$ for $j \in NB$ and $\overline{a}_{u0} \geq 0$ for $u = 1, \ldots, m$, where NB is the se of nonbasic variables.

If the basic optimal solution x^* is not integer, there exists some row u wit $\overline{a}_{u0} \notin Z^1$. Choosing such a row, the Chvátal-Gomory cut for row u is

$$x_{B_u} + \sum_{j \in NB} \lfloor \overline{a}_{uj} \rfloor x_j \leq \lfloor \overline{a}_{u0} \rfloor. \tag{8.10}$$

Rewriting this inequality by eliminating x_{B_u} gives

$$\sum_{j \in NB} (\overline{a}_{uj} - \lfloor \overline{a}_{uj} \rfloor) x_j \geq \overline{a}_{u0} - \lfloor \overline{a}_{u0} \rfloor$$

or

$$\sum_{j \in NB} f_{uj} x_j \geq f_{u0} \tag{8.11}$$

where $f_{uj} = \overline{a}_{uj} - \lfloor \overline{a}_{uj} \rfloor$ for $j \in NB$, and $f_{u0} = \overline{a}_{u0} - \lfloor \overline{a}_{u0} \rfloor$.

By the definitions and the choice of row u, $0 \leq f_{uj} < 1$ and $0 < f_{u0} <$ As x_j^*=0 for all nonbasic variables $j \in NB$ in the optimal LP solution, this i equality cuts off x^*. It is also important to observe that the difference betwee the left- and right-hand sides of the Chvátal-Gomory inequality (8.10), an hence also of (8.11), is integral when x is integral, so that when (8.11) rewritten as an equation:

$$s = -f_{u0} + \sum_{j \in NB} f_{uj} x_j,$$

the slack variable s is a nonnegative integer variable.

Example 8.10 Consider the integer program

$$\begin{array}{rrrrrrl} z = \max & 4x_1 & - & x_2 & & & \\ & 7x_1 & - & 2x_2 & \leq & 14 & \\ & & & x_2 & \leq & 3 & \\ & 2x_1 & - & 2x_2 & \leq & 3 & \\ & x_1, & & x_2 & \geq & 0 \text{ and integer.} & \end{array}$$

Adding slack variables x_3, x_4, x_5, observe that as the constraint data is intege the slack variables must also take integer values. Now solving as a linea program gives:

$$\begin{array}{rrrrrrrrrl} z = \max \frac{59}{7} & & & -\frac{4}{7}x_3 & & -\frac{1}{7}x_4 & & & & \\ & x_1 & & +\frac{1}{7}x_3 & & +\frac{2}{7}x_4 & & & = & \frac{20}{7} \\ & & x_2 & & & +x_4 & & & = & 3 \\ & & & -\frac{2}{7}x_3 & & +\frac{10}{7}x_4 & +x_5 & & = & \frac{23}{7} \\ & x_1, & x_2, & x_3, & & x_4, & & x_5 & \geq & 0 \text{ and integer.} \end{array}$$

The optimal linear programming solution is $x = (\frac{20}{7}, 3, 0, 0, \frac{23}{7}) \notin Z^5_+$, so we use the first row, in which the basic variable x_1 is fractional, to generate the cut:

$$\tfrac{1}{7}x_3 + \tfrac{2}{7}x_4 \geq \tfrac{6}{7}$$

or

$$s = -\tfrac{6}{7} + \tfrac{1}{7}x_3 + \tfrac{2}{7}x_4$$

with $s, x_3, x_4 \geq 0$ and integer.

Adding this cut, and reoptimizing leads to the new optimal tableau

$$\begin{array}{lccccccl} z = \max \frac{15}{2} & & & & -\frac{1}{2}x_5 & -3s & & \\ & x_1 & & & & +s & = 2 & \\ & & x_2 & & -\frac{1}{2}x_5 & +s & = \frac{1}{2} & \\ & & & x_3 & -x_5 & -5s & = 1 & \\ & & & & x_4 + \frac{1}{2}x_5 & +6s & = \frac{5}{2} & \\ & x_1, & x_2, & x_3, & x_4, & x_5, s & \geq & 0 \text{ and integer.} \end{array}$$

Now the new optimal linear programming solution $x = (2, \frac{1}{2}, 1, \frac{5}{2}, 0)$ is still not integer, as the original variable x_2, and the slack variable x_4 are fractional. The Gomory fractional cut on row 2, in which x_2 is basic, is $\frac{1}{2}x_5 \geq \frac{1}{2}$ or $-\frac{1}{2}x_5 + t = -\frac{1}{2}$ with $t \geq 0$ and integer. Adding this constraint and reoptimizing, we obtain

$$\begin{array}{lccccccl} z = \max 7 & & & & & -3s & -t & \\ & x_1 & & & & +s & & = 2 \\ & & x_2 & & & +s & -t & = 1 \\ & & & x_3 & & -5s & -2t & = 2 \\ & & & & x_4 & +6s & +t & = 2 \\ & & & & x_5 & & -t & = 1 \\ & x_1, & x_2, & x_3, & x_4, & x_5, s, & t & \geq 0 \text{ and integer.} \end{array}$$

Now the linear programing solution is integral, and optimal, and thus $(x_1, x_2) = (2, 1)$ solves the original integer program. ∎

It is natural to also look at the cuts in the space of the original variables.

Example 8.10 (cont) Considering the first cut, and substituting for x_3 and x_4 gives:

$$\tfrac{1}{7}(14 - 7x_1 + 2x_2) + \tfrac{2}{7}(3 - x_2) \geq \tfrac{6}{7}$$

or $x_1 \leq 2$.

In Figure 8.2 we can verify that this inequality is valid and cuts off the fractional solution $(\frac{20}{7}, 3)$. Similarly, substituting for x_5 in the second cut $\frac{1}{2}x_5 \geq \frac{1}{2}$ gives the valid inequality $x_1 - x_2 \leq 1$ in the original variables. ∎

To find a general formula that gives us the cut in terms of the original variables, one can show:

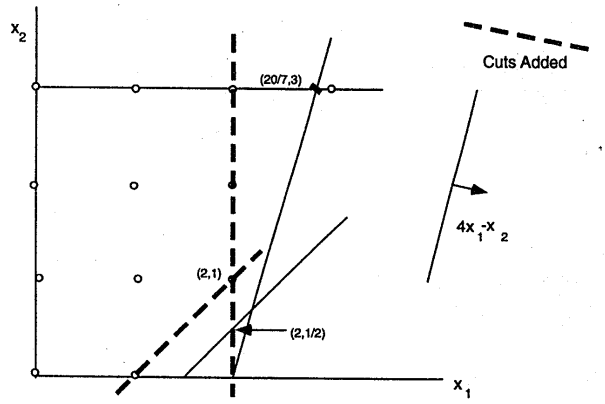


Fig. 8.2 Gomory cutting planes

Proposition 8.5 *Let β be row u of B^{-1}, and $q_i = \beta_i - \lfloor \beta_i \rfloor$ for $i = 1, \cdots, m$. The Gomory cut $\sum_{j \in NB} f_{uj} x_j \geq f_{u0}$, when written in terms of the original variables, is the Chvátal-Gomory inequality*

$$\sum_{j=1}^{n} \lfloor q a_j \rfloor x_j \leq \lfloor qb \rfloor.$$

Looking at the first Gomory cut generated in Example 8.10, β is given by the coefficients of the slack variables in row $u = 1$, so $\beta = (\frac{1}{7}, \frac{2}{7}, 0)$. Thus $q = (\frac{1}{7}, \frac{2}{7}, 0)$ and we obtain $1x_1 + 0x_2 \leq \lfloor \frac{20}{7} \rfloor = 2$.

8.7 MIXED INTEGER CUTS

8.7.1 The Basic Mixed Integer Inequality

We saw above that when $y \leq b, y \in Z^1$, the rounding inequality $y \leq \lfloor b \rfloor$ suffices to generate all the inequalities for a pure integer program. Here we examine if there is a similar *basic* inequality for mixed integer programs.

Proposition 8.6 *Let $X^{\geq} = \{(x, y) \in R^1_+ \times Z^1 : x + y \geq b\}$, and $f = b - \lfloor b \rfloor > 0$. The inequality*

$$x \geq f(\lceil b \rceil - y) \text{ or } \frac{x}{f} + y \geq \lceil b \rceil$$

is valid for $X^{\geq}$.

Chapter 9

Branch and Bound

$\sum_{j\in N\setminus S} a_j$, or prove that it is impossible. Show that 2-*PARTITION* is polynomially reducible to 0–1 *KNAPSACK*. Does this imply that 2-*PARTITION* is $\mathcal{NP}$-complete?

2. Show that *SATISFIABILITY* is polynomially reducible to *STABLE SET* (Node Packing), and thus that *STABLE SET* is $\mathcal{NP}$-complete, where *STABLE SET* is the problem of finding a maximum weight set of nonadjacent nodes in a graph.

3. Show that *STABLE SET* is polynomially reducible to *SET PACKING*, where *SET PACKING* is the problem of finding a maximum weight set of disjoint columns in a 0–1 matrix.

4. Show that *SET COVERING* is polynomially reducible to *UFL*.

5. Show that *SET COVERING* is polynomially reducible to *DIRECTED STEINER TREE*.

6. Given $D = (V, A)$, c_e for $e \in A$, a subset $F \subseteq A$, and a node $r \in V$, *ARC ROUTING* is the problem of finding a minimum length directed subtour that contains the arcs in F and starts and ends at node r. Show that *TSP* is polynomially reducible to *ARC ROUTING*.

7.* Show that the decision problem associated to *IP* is an integer programming feasibility problem, and is in $\mathcal{NP}$.

8. Consider a 0–1 knapsack set $X = \{x \in B^n : \sum_{j\in N} a_j x_j \leq b\}$ with $0 \leq a_j \leq b$ for $j \in N$ and let $\{x^t\}_{t=1}^T$ be the points of X. With it, associate the bounded polyhedron $\Pi^1 = \{\pi \in R_+^n : x^t\pi \leq 1 \text{ for } t = 1,\dots,T\}$ with extreme points $\{\pi^s\}_{s=1}^S$. Consider a point x^* with $0 \leq x_j^* \leq 1$ for $j \in N$.

(i) Show that $x^* \in \text{conv}(X)$ if and only if $\min\{\sum_{t=1}^T \lambda_t : x^* \leq \sum_{t=1}^T x^t\lambda_t, \lambda \in R_+^T\} = \max\{x^*\pi : \pi \in \Pi^1\} \leq 1$.
(ii) Deduce that if $x^* \notin \text{conv}(X)$, then for some $s = 1,\dots,S$, $\pi^s x^* > 1$.

7

Branch and Bound

7.1 DIVIDE AND CONQUER

Consider the problem:

$$z = \max\{cx : x \in S\}.$$

How can we break the problem into a series of smaller problems that are easier, solve the smaller problems, and then put the information together again to solve the original problem?

Proposition 7.1 *Let $S = S_1 \cup \dots \cup S_K$ be a decomposition of S into smaller sets, and let $z^k = \max\{cx : x \in S_k\}$ for $k = 1,\dots,K$. Then $z = \max_k z^k$.*

A typical way to represent such a divide and conquer approach is via an enumeration tree. For instance, if $S \subseteq \{0,1\}^3$, we might construct the enumeration tree shown in Figure 7.1.

Here we first divide S into $S_0 = \{x \in S : x_1 = 0\}$ and $S_1 = \{x \in S : x_1 = 1\}$, then $S_{00} = \{x \in S_0 : x_2 = 0\} = \{x \in S : x_1 = x_2 = 0\}$, $S_{01} = \{x \in S_0 : x_2 = 1\}$, and so on. Note that a leaf of the tree $S_{i_1 i_2 i_3}$ is nonempty if and only if $x = (i_1, i_2, i_3)$ is in S. Thus the leaves of the tree correspond precisely to the points of B^3 that one would examine if one carried out complete enumeration. Note that by convention the tree is drawn upside down with its root at the top.

Another example is the enumeration of all the tours of the traveling salesman problem. First we divide S the set of all tours on 4 cities into $S_{(12)}, S_{(13)}, S_{(14)}$ where $S_{(ij)}$ is the set of all tours containing arc (ij). Then $S_{(12)}$ is divided again into $S_{(12)(23)}$ and $S_{(12)(24)}$, and so on. Note that at the first level we have arbitrarily chosen to branch on the arcs leaving node 1, and at the

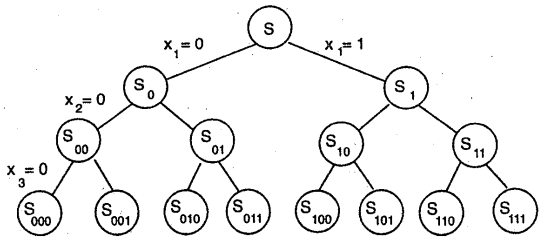


Fig. 7.1 Binary enumeration tree

second level on the arcs leaving node 2 that do not immediately create a subtour with the previous branching arc. The resulting tree is shown in Figure 7.2. Here the six leaves of the tree correspond to the $(n-1)!$ tours shown, where $i_1 i_2 i_3 i_4$ means that the cities are visited in the order i_1, i_2, i_3, i_4, i_1 respectively. Note that this is an example of multiway as opposed to binary branching, where a set can be divided into more than two parts.

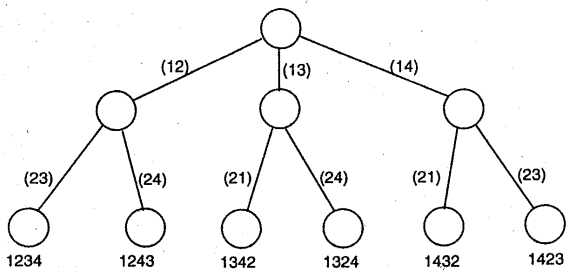


Fig. 7.2 *TSP* enumeration tree

7.2 IMPLICIT ENUMERATION

We saw in Chapter 1 that complete enumeration is totally impossible for most problems as soon as the number of variables in an integer program, or nodes in a graph exceeds 20 or 30. So we need to do more than just divide indefinitely. How can we use some bounds on the values of $\{z^k\}$ intelligently? First, how can we put together bound information?

Proposition 7.2 *Let $S = S_1 \cup \ldots \cup S_K$ be a decomposition of S into smaller sets, and let $z^k = \max\{cx : x \in S_k\}$ for $k = 1, \ldots, K$, $\overline{z}^k$ be an upper bound on z^k and $\underline{z}^k$ be a lower bound on z^k. Then $\overline{z} = \max_k \overline{z}^k$ is an upper bound on z and $\underline{z} = \max_k \underline{z}^k$ is a lower bound on z.*

Now we examine three hypothetical examples to see how bound information, or partial information about a subproblem can be put to use. What can be deduced about lower and upper bounds on the optimal value z and which sets need further examination in order to find the optimal value?

Example 7.1 In Figure 7.3 we show a decomposition of S into two sets S_1 and S_2 as well as upper and lower bounds on the corresponding problems.

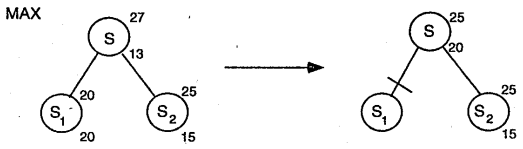


Fig. 7.3 Pruned by optimality

We note first that $\overline{z} = \max_k \overline{z}^k = \max\{20, 25\} = 25$ and $\underline{z} = \max_k \underline{z}^k = \max\{20, 15\} = 20$.

Second, we observe that as the lower and upper bounds on z_1 are equal, $z_1 = 20$, and there is no further reason to examine the set S_1. Therefore the branch S_1 of the enumeration tree can be pruned *by optimality.* ∎

Example 7.2 In Figure 7.4 we again decompose S into two sets S_1 and S_2 and show upper and lower bounds on the corresponding problems.

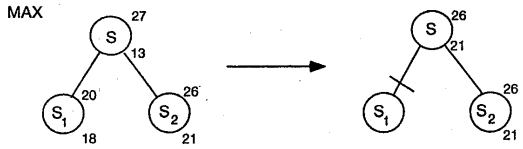


Fig. 7.4 Pruned by bound

We note first that $\overline{z} = \max_k \overline{z}^k = \max\{20, 26\} = 26$ and $\underline{z} = \max_k \underline{z}^k = \max\{18, 21\} = 21$.

Second, we observe that as the optimal value has value at least 21, and the upper bound $\overline{z}^1 = 20$, no optimal solution can lie in the set S_1. Therefore the branch S_1 of the enumeration tree can be pruned *by bound.* ∎

Example 7.3 In Figure 7.5 we again decompose S into two sets S_1 and S_2 with different upper and lower bounds.

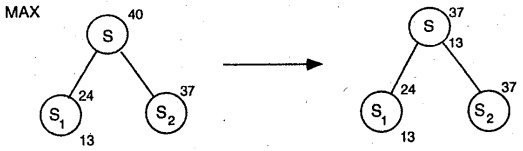


Fig. 7.5 No pruning possible

We note first that $\overline{z} = \max_k \overline{z}^k = \max\{24, 37\} = 37$ and $\underline{z} = \max_k \underline{z}^k = \max\{13, -\} = 13$. Here no other conclusion can be drawn and we need to explore both sets S_1 and S_2 further. ∎

Based on these examples, we can list at least three reasons that allow us to prune the tree and thus enumerate a large number of solutions implicitly.

(i) Pruning by optimality: $z_t = \{\max cx : x \in S_t\}$ has been solved.

(ii) Pruning by bound: $\overline{z_t} \leq \underline{z}$.

(iii) Pruning by infeasiblity: $S_t = \phi$.

If we now ask how the bounds are to be obtained, the reply is no different from in Chapter 2. The primal (lower) bounds are provided by feasible solutions and the dual (upper) bounds by relaxation or duality.

Building an implicit enumeration algorithm based on the above ideas is now in principle a fairly straightforward task. There are, however, many questions that must be addressed before such an algorithm is well-defined. Some of the most important questions are:

What relaxation or dual problem should be used to provide upper bounds? How should one choose between a fairly weak bound that can be calculated very rapidly and a stronger bound whose calculation takes a considerable time?

How should the feasible region be separated into smaller regions $S = S_1 \cup \ldots \cup S_K$? Should one separate into two or more parts? Should one use a fixed a priori rule for dividing up the set, or should the divisions evolve as a function of the bounds and solutions obtained en route?

In what order should the subproblems be examined? Typically there is a list of active problems that have not yet been pruned. Should the next one be chosen on a the basis of last-in first-out, of best/largest upper bound first, or of some totally different criterion?

These and other questions will be discussed further once we have seen an example.

7.3 BRANCH AND BOUND: AN EXAMPLE

The most common way to solve integer programs is to use implicit enumeration, or *branch and bound*, in which linear programming relaxations provide the bounds. We first demonstrate the approach by an example:

$$z = \max 4x_1 - x_2 \tag{7.1}$$
$$7x_1 - 2x_2 \leq 14 \tag{7.2}$$
$$x_2 \leq 3 \tag{7.3}$$
$$2x_1 - 2x_2 \leq 3 \tag{7.4}$$
$$x \in Z^2_+. \tag{7.5}$$

Bounding. To obtain a first upper bound, we add slack variables x_3, x_4, x_5 and solve the linear programming relaxation in which the integrality constraints are dropped. The resulting optimal basis representation is:

$$\begin{array}{rrrrrrr} \overline{z} = \max \frac{59}{7} & & -\frac{4}{7}x_3 & -\frac{1}{7}x_4 & & & \\ x_1 & & +\frac{1}{7}x_3 & +\frac{2}{7}x_4 & & = & \frac{20}{7} \\ & x_2 & & +x_4 & & = & 3 \\ & & -\frac{2}{7}x_3 & +\frac{10}{7}x_4 & +x_5 & = & \frac{23}{7} \\ x_1, & x_2, & x_3, & x_4, & x_5 & \geq & 0. \end{array}$$

Thus we obtain an upper bound $\overline{z} = \frac{59}{7}$, and a nonintegral solution $(\overline{x}_1, \overline{x}_2) = (\frac{20}{7}, 3)$. Is there any straightforward way to find a feasible solution? Apparently not. By convention, as no feasible solution is yet available, we take as lower bound $\underline{z} = -\infty$.

Branching. Now because $\underline{z} < \overline{z}$, we need to divide or branch. How should we split up the feasible region? One simple idea is to choose an integer variable that is basic and fractional in the linear programming solution, and split the problem into two about this fractional value. If $x_j = \overline{x}_j \notin Z^1$, one can take:

$$S_1 = S \cap \{x : x_j \leq \lfloor \overline{x}_j \rfloor\}$$
$$S_2 = S \cap \{x : x_j \geq \lceil \overline{x}_j \rceil\}.$$

It is clear that $S = S_1 \cup S_2$ and $S_1 \cap S_2 = \phi$. Another reason for this choice is that the solution $\bar{x}$ of $LP(S)$ is not feasible in either $LP(S_1)$ or $LP(S_2)$. This implies that if there is no degeneracy (i.e., multiple optimal LP solutions), then $\max\{\overline{z}_1, \overline{z}_2\} < \overline{z}$, so the upper bound will strictly decrease.

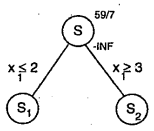


Fig. 7.6 Partial branch-and-bound tree 1

Following this idea, as $\bar{x}_1 = 20/7 \notin Z^1$, we take $S_1 = S \cap \{x : x_1 \leq 2\}$ and $S_2 = S \cap \{x : x_1 \geq 3\}$. We now have the tree shown in Figure 7.6. The subproblems (nodes) that must still be examined are called *active*.

Choosing a Node. The list of active problems (nodes) to be examined now contains S_1, S_2. We arbitrarily choose S_1.

Reoptimizing. How should we solve the new modified linear programs $LP(S_i)$ for $i = 1, 2$ without starting again from scratch?

As we have just added one single upper or lower bound constraint to the linear program, our previous optimal basis remains dual feasible, and it is therefore natural to reoptimize from this basis using the dual simplex algorithm. Typically, only a few pivots will be needed to find the new optimal linear programming solution.

Applying this to the linear program $LP(S_1)$, we can write the new constraint $x_1 \leq 2$ as $x_1 + s = 2, s \geq 0$, which can be rewritten in terms of the nonbasic variables as

$$-\tfrac{1}{7}x_3 - \tfrac{2}{7}x_4 + s = -\tfrac{6}{7}.$$

Thus we have the dual feasible representation:

$$\begin{array}{rlllllll}
\bar{z}_1 = \max \tfrac{59}{7} & & -\tfrac{4}{7}x_3 & -\tfrac{1}{7}x_4 & & & & \\
x_1 & & +\tfrac{1}{7}x_3 & +\tfrac{2}{7}x_4 & & & = & \tfrac{20}{7} \\
x_2 & & & +x_4 & & & = & 3 \\
& & -\tfrac{2}{7}x_3 & +\tfrac{10}{7}x_4 & +x_5 & & = & \tfrac{23}{7} \\
& & -\tfrac{1}{7}x_3 & -\tfrac{2}{7}x_4 & & +s & = & -\tfrac{6}{7} \\
x_1, & x_2, & x_3, & x_4, & x_5, & s & \geq & 0.
\end{array}$$

After two simplex pivots, the linear program is reoptimized, giving:

$$\begin{array}{rllllll}
\bar{z}_1 = \max \tfrac{15}{2} & & & -\tfrac{1}{2}x_5 & -3s & & \\
x_1 & & & & +s & = & 2 \\
x_2 & & & -\tfrac{1}{2}x_5 & +s & = & \tfrac{1}{2} \\
x_3 & & & -x_5 & -5s & = & 1 \\
x_4 & & & +\tfrac{1}{2}x_5 & +6s & = & \tfrac{5}{2} \\
x_1, & x_2, & x_3, & x_4, & x_5, & s & \geq 0
\end{array}$$

with $\bar{z}_1 = \frac{15}{2}$, and $(\bar{x}_1^1, \bar{x}_2^1) = (2, \frac{1}{2})$.

Branching. S_1 cannot be pruned, so using the same branching rule as before, we create two new nodes $S_{11} = S_1 \cap \{x : x_2 \leq 0\}$ and $S_{12} = S_1 \cap \{x : x_2 \geq 1\}$, and add them to the node list. The tree is now as shown in Figure 7.7.

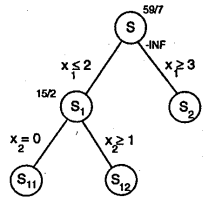


Fig. 7.7 Partial branch-and-bound tree 2

Choosing a Node. The active node list now contains S_2, S_{11}, S_{12}. Arbitrarily choosing S_2, we remove it from the node list and examine it in more detail.

Reoptimizing. To solve $LP(S_2)$, we use the dual simplex algorithm in the same way as above. The constraint $x_1 \geq 3$ is first written as $x_1 - t = 3, t \geq 0$, which expressed in terms of the nonbasic variables becomes:

$$\tfrac{1}{7}x_3 + \tfrac{2}{7}x_4 + t = -\tfrac{1}{7}.$$

From inspection of this constraint, we see that the resulting linear program

$$\begin{array}{rlllllll}
\bar{z}_2 = \max \tfrac{59}{7} & & -\tfrac{4}{7}x_3 & -\tfrac{1}{7}x_4 & & & & \\
x_1 & & +\tfrac{1}{7}x_3 & +\tfrac{2}{7}x_4 & & & = & \tfrac{20}{7} \\
x_2 & & & +x_4 & & & = & 3 \\
& & -\tfrac{2}{7}x_3 & +\tfrac{10}{7}x_4 & +x_5 & & = & \tfrac{23}{7} \\
& & \tfrac{1}{7}x_3 & +\tfrac{2}{7}x_4 & & +t & = & -\tfrac{1}{7} \\
x_1, & x_2, & x_3, & x_4, & x_5, & t & \geq & 0
\end{array}$$

is infeasible, $\bar{z}_2 = -\infty$, and hence node S_2 is *pruned by infeasibility.*

Choosing a Node. The node list now contains S_{11}, S_{12}. Arbitrarily choosing S_{12}, we remove it from the list.

Reoptimizing. $S_{12} = S \cap \{x : x_1 \leq 2, x_2 \geq 1\}$. The resulting linear program has optimal solut $\bar{x}^{12} = (2, 1)$ with value 7. As $\bar{x}^{12}$ is integer, $z^{12} = 7$.

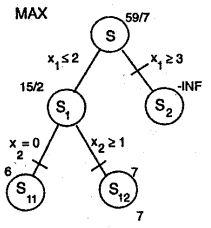


Fig. 7.8 Complete branch and bound tree

Updating the Incumbent. As the solution of $LP(S_{12})$ is integer, we update the value of the best feasible solution found $\underline{z} \leftarrow \max\{\underline{z}, 7\}$, and store the corresponding solution $(2,1)$. S_{12} is now *pruned by optimality.*

Choosing a Node. The node list now contains only S_{11}.

Reoptimizing. $S_{11} = S \cap \{x : x_1 \leq 2, x_2 \leq 0\}$. The resulting linear program has optimal solution $\overline{x}^{11} = (\frac{3}{2}, 0)$ with value 6. As $\underline{z} = 7 > \overline{z}_{11} = 6$, the node is *pruned by bound.*

Choosing a Node. As the node list is empty, the algorithm terminates. The incumbent solution $x = (2,1)$ with value $z = 7$ is optimal.

The complete branch-and-bound tree is shown in Figure 7.8. In Figure 7.9 we show graphically the feasible node sets S_i, the branching, the relaxations $LP(S_i)$, and the solutions encountered in the example.

7.4 LP-BASED BRANCH AND BOUND

In Figure 7.10 we present a flowchart of a simple branch and bound algorithm, and then discuss in more detail some of the practical aspects of developing and using such an algorithm.

Storing the Tree. In practice one does not store a tree, but just the list of *active* nodes or subproblems that have not been pruned and that still need to be explored further. Here the question arises of how much information one should keep. Should one keep a minimum of information and be prepared to repeat certain calculations, or should one keep all the information available? At a minimum, the best known dual bound and the variable lower and upper

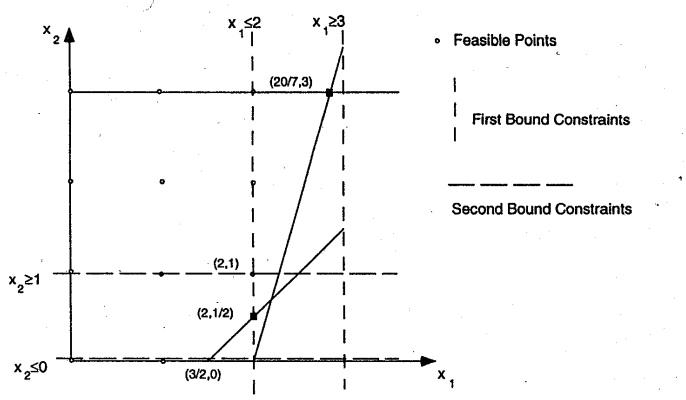


Fig. 7.9 Division of the feasible region

bounds needed to restore the subproblem are stored. Usually one also keeps an optimal or near-optimal basis, so that the linear programming relaxation can be reoptimized rapidly.

Returning to the questions raised earlier, there is no single answer that is best for all instances. One needs to use rules based on a combination of theory, common sense, and practical experimentation. In our example, the question of **how to bound** was solved by using an LP relaxation; **how to branch** was solved by choosing an integer variable that is fractional in the LP solution. However, as there is typically a choice of a set C of several candidates, we need a rule to choose between them. One common choice is *the most fractional variable*:

$$\arg\max_{j \in C} \min[f_j, 1 - f_j]$$

where $f_j = x_j^* - \lfloor x_j^* \rfloor$, so that a variable with fractional value $f_j = \frac{1}{2}$ is best. Other rules are based on the idea of *estimating* the cost of forcing the variable x_j to become integer.

How to choose a node was avoided by making an arbitrary choice. In practice there are several contradictory arguments that can be invoked:

(i) It is only possible to prune the tree significantly with a (primal) feasible solution, giving a hopefully good lower bound. Therefore one should descend as quickly as possible in the enumeration tree to find a first feasible solution. This suggests the use of a *Depth-First Search* strategy. Another argument for

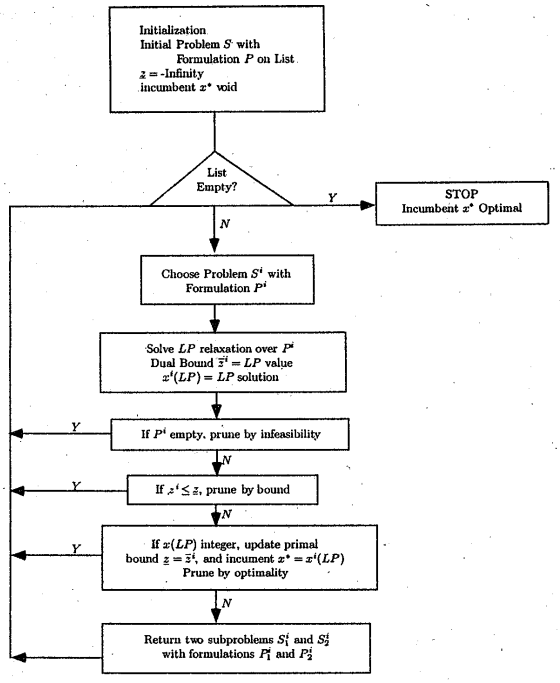


Fig. 7.10 Branch-and-bound flow chart

such a strategy is the observation that it is always easy to resolve the linear programming relaxation when a simple constraint is added, and the optimal basis is available. Therefore passing from a node to one of its immediate descendants is to be encouraged. In the example this would imply that after treating node S_1, the next node treated would be S_{11} or S_{12} rather than S_2.

(ii) To minimize the total number of nodes evaluated in the tree, the optimal strategy is to always choose the active node with the best (largest upper) bound (i.e., choose node s where $\overline{z}_s = \max_t \overline{z}_t$). With such a rule, one will never divide any node whose upper bound $\overline{z}_t$ is less than the optimal value z. This leads to a *Best-Node First* strategy. In the example of the previous section, this would imply that after treating node S_1, the next node chosen

would be S_2 with bound $\frac{59}{7}$ from its predecessor, rather than S_{11} or S_{12} with bound $\frac{15}{2}$.

In practice a compromise between these ideas is often adopted, involving an initial depth-first strategy until at least one feasible solution has been found, followed by a strategy mixing best node and depth first so as to try to prove optimality and also find better feasible solutions.

7.5 USING A BRANCH-AND-BOUND SYSTEM

Commercial branch-and-bound systems for integer and mixed integer programming are essentially as described in the previous section, and the default strategies have been chosen by tuning over hundreds of different problem instances. The basic philosophy is to solve and resolve the linear programming relaxations as rapidly as possible, and if possible to branch intelligently. Given this philosophy, all recent systems contain, or offer,

1. A powerful (automatic) preprocessor, which simplifies the model by reducing the number of constraints and variables, so that the linear programs are easier
2. The simplex algorithm with a choice of pivoting strategies, and an interior point option for solving the linear programs
3. Limited choice of branching and node selection options
4. Use of priorities

and some offer

5. GUB/SOS branching
6. Strong branching
7. Reduced cost fixing
8. Primal heuristics

In this section we briefly discuss those topics requiring user intervention. Preprocessing, which is very important, but automatic, is presented in the (optional) next section. Reduced cost fixing is treated in Exercise 7.7, and primal heuristics are discussed in Chapter 12.

Priorities. Priorities allow the user to tell the system the relative importance of the integer variables. The user provides a file specifying a value (importance) of each integer variable. When it has to decide on a branching variable, the system will choose the highest priority integer variable whose current linear programming value is fractional. At the same time the user can specify a preferred branching direction telling the system which of the two branches to

explore first.

GUB Branching. Many models contain generalized upper bound (GUB) or special ordered sets (SOS) of the form

$$\sum_{j=1}^{k} x_j = 1$$

with $x_j \in \{0,1\}$ for $j = 1,\ldots,k$. If the linear programming solution x^* has some of the variables $x_1^*,\ldots,x_k^*$ fractional, then the standard branching rule is to impose $S_1 = S \cap \{x : x_j = 0\}$ and $S_2 = S \cap \{x : x_j = 1\}$ for some $j \in \{1,\ldots,k\}$. However, because of the GUB constraint, $\{x : x_j = 0\}$ leaves $k-1$ possibilities $\{x : x_i = 1\}_{i \neq j}$ whereas $\{x : x_j = 1\}$ leaves only one possibility. So S_1 is typically a much larger set than S_2, and the tree is *unbalanced.*

GUB branching is designed to provide a more balanced division of S into S_1 and S_2. Specifically the user specifies an ordering of the variables in the GUB set $j_1,\ldots,j_k$, and the branching scheme is then to set

$$S_1 = S \cap \{x : x_{j_i} = 0 \; i = 1,\ldots r\} \text{ and}$$
$$S_2 = S \cap \{x : x_{j_i} = 0 \; i = r+1,\ldots k\},$$

where $r = \min\{t : \sum_{i=1}^{t} x_{j_i}^* \geq \frac{1}{2}\}$. In many cases such a branching scheme is much more effective than the standard scheme, and the number of nodes in the tree is significantly reduced.

User Options (a) **Cutoffs.** If the user knows or can construct a good feasible solution to his or her problem, it is very important that its value is passed to the system as the incumbent value to serve as a *cutoff* in the branch and bound.

(b) **Simplex Strategies.** Though the linear programming algorithms are finely tuned, the default strategy will not be best for all classes of problems. Different *simplex pricing* strategies may make a huge difference in running times for a given class of models, so if similar models are resolved repeatedly or the linear programs seem very slow, some experimentation by the user with pricing strategies is permitted. In addition, on very large models, *interior point methods* may be best for the solution of the first linear program. Unfortunately, up to now such methods are still not good for reoptimizing quickly at each node of the tree.

(c) **Strong Branching.** The idea behind strong branching is that on difficult problems it should be worthwhile to do more work to try to choose a better branching variable. The system chooses a set C of basic integer variables that are fractional in the LP solution, branches up and down on each of them in turn, and reoptimizes on each branch either to optimality, or for a specified number of dual simplex pivots. Now for each variable $j \in C$, it has upper bounds z_j^D for the down branch and z_j^U for the up branch. The variable having the largest effect (decrease of the dual bound)

$$j^* = \arg\min_{j \in C} \max[z_j^D, z_j^U]$$

is then chosen, and branching really takes place on this variable. Obviously, solving two LPs for each variable in C is costly, so such branching should only be used when the other criteria have been found to be ineffective.

7.5.1 If All Else Fails

What can one do if a particular problem instance turns out to be difficult, meaning that after a certain time

(i) no feasible solution has been found, or

(ii) the gap between the value of the best feasible solution and the value of the dual upper bound is unsatisfactorily large, or

(iii) the system runs out of space because there are too many active nodes in the node list?

Finding Feasible Solutions. This is in general $\mathcal{NP}$-hard. Some systems have simple primal heuristics embedded in them. Also as discussed earlier, using priorities and directions for branching can help. How to find feasible solutions, starting from the LP solution or using explicit problem structure, is the topic of Chapter 12.

Finding Better Dual Bounds. Branch-and-bound algorithms fail very often because the bounds obtained from the linear programming relaxations are too weak. This means that tightening up the formulation of the problem is of crucial importance. Systematic ways to do this are the subject of Chapters 8–11. Specifically the addition of constraints or cuts to improve the formulation is treated in Chapers 8 and 9, leading to the development of a potentially more powerful branch-and-cut algorithm. The Lagrangian relaxation and column generation approaches of Chapters 10 and 11 provide alternative ways to strengthen the formulation by convexifying part of the feasible region.

7.6 PREPROCESSING*

Before solving a linear or integer program, it is natural to check that the formulation is "sensible", and as strong as possible given the information available.

All the commercial branch-and-bound systems carry out such a check, called *preprocessing*. The basic idea is to try to quickly detect and eliminate redundant constraints and variables, and tighten bounds where possible. Then if the resulting linear/integer program is smaller/tighter, it will typically be solved much more quickly. This is especially important in the case of branch-and-bound because tens or hundreds of thousands of linear programs may need to be solved.

First we demonstrate linear programming preprocessing by an example.

Example 7.4 Consider the linear program

$$\begin{array}{lrcrcrcl} \max & 2x_1 & + & x_2 & - & x_3 & & \\ & 5x_1 & - & 2x_2 & + & 8x_3 & \leq & 15 \\ & 8x_1 & + & 3x_2 & - & x_3 & \geq & 9 \\ & x_1 & + & x_2 & + & x_3 & \leq & 6 \\ & 0 & \leq & x_1 & \leq & 3 & & \\ & 0 & \leq & x_2 & \leq & 1 & & \\ & 1 & \leq & x_3. & & & & \end{array}$$

Tightening Bounds. Isolating variable x_1 in the first constraint we obtain

$$5x_1 \leq 15 + 2x_2 - 8x_3 \leq 15 + 2 \times 1 - 8 \times 1 = 9$$

where we use the bound inequalities $x_2 \leq 1$ and $-x_3 \leq -1$. Thus we obtain the tightened bound $x_1 \leq \frac{9}{5}$.

Similarly isolating variable x_3, we obtain

$$8x_3 \leq 15 + 2x_2 - 5x_1 \leq 15 + 2 \times 1 - 5 \times 0 = 17,$$

and the tightened bound $x_3 \leq \frac{17}{8}$.

Isolating variable x_2, we obtain

$$2x_2 \geq 5x_1 + 8x_3 - 15 \geq 5 \times 0 + 8 \times 1 - 15 = -7.$$

Here the existing bound $x_2 \geq 0$ is not changed.

Turning to the second constraint, isolating x_1 and using the same approach, we obtain $8x_1 \geq 9 - 3x_2 + x_3 \geq 9 - 3 + 1 = 7$, and an improved lower bound $x_1 \geq \frac{7}{8}$.

No more bounds are changed based on the second or third constraints. However, as certain bounds have been tightened, it is worth passing through the constraints again.

Constraint 1 for x_3 now gives $8x_3 \leq 15 + 2x_2 - 5x_1 \leq 15 + 2 - 5 \times \frac{7}{8} = \frac{101}{8}$. Thus we have the new bound $x_3 \leq \frac{101}{64}$.

Redundant Constraints. Using the latest upper bounds in constraint 3, we see that

$$x_1 + x_2 + x_3 \leq \frac{9}{5} + 1 + \frac{101}{64} < 6,$$

and so this constraint is redundant and can be discarded. The problem is now reduced to

$$\begin{array}{lrrrl} \max & 2x_1 & +x_2 & -x_3 & \\ & 5x_1 & -2x_2 & +8x_3 & \leq 15 \\ & 8x_1 & +3x_2 & -x_3 & \geq 9 \\ & \frac{7}{8} \leq x_1 & \leq \frac{9}{5}, & 0 \leq x_2 \leq 1, & 1 \leq x_3 \leq \frac{101}{64}. \end{array}$$

Variable Fixing (by Duality). Considering variable x_2, observe that increasing its value makes all the constraints (other than its bound constraints) less tight. As the variable has a positive objective coefficent, it is advantageous to make the variable as large as possible, and thus set it to its upper bound of 1. (Another way to arrive at a similar conclusion is to write out the LP dual. For the dual constraint corresponding to the primal variable x_2 to be feasible, the dual variable associated with the constraint $x_2 \leq 1$ must be positive. This implies by complememtary slackness that $x_2 = 1$ in any optimal solution.)

Similarly, decreasing x_3 makes the constraints less tight. As the variable has a negative objective coefficient, it is best to make it as small as possible, and thus set it to its lower bound $x_3 = 1$. Finally the LP is reduced to the trivial problem

$$\max\{2x_1 : \tfrac{7}{8} \leq x_1 \leq \tfrac{9}{5}\}.$$

■

Formalizing the above ideas is straightforward.

Proposition 7.3 *Consider the set* $S = \{x : a_0x_0 + \sum_{j=1}^n a_jx_j \leq b, l_j \leq x_j \leq u_j \text{ for } j = 0, 1, \ldots, n\}$.

(i) Bounds on Variables. If $a_0 > 0$, *then*

$$x_0 \leq (b - \sum_{j:a_j>0} a_jl_j - \sum_{j:a_j<0} a_ju_j)/a_0,$$

and if $a_0 < 0$, *then*

$$x_0 \geq (b - \sum_{j:a_j>0} a_jl_j - \sum_{j:a_j<0} a_ju_j)/a_0.$$

(ii) Redundancy. The constraint $a_0x_0 + \sum_{j=1}^n a_jx_j \leq b$ *is redundant if*

$$\sum_{j:a_j>0} a_ju_j + \sum_{j:a_j<0} a_jl_j \leq b.$$

(iii) Infeasibility. $S = \emptyset$ *if*

$$\sum_{j:a_j>0} a_jl_j + \sum_{j:a_j<0} a_ju_j > b.$$

(iv) Variable Fixing. For a maximization problem in the form: $\max\{cx : Ax \leq b, l \leq x \leq u\}$, *if* $a_{ij} \geq 0$ *for all* $i = 1, \ldots, m$ *and* $c_j < 0$, *then* $x_j = l_j$. *Conversely if* $a_{ij} \leq 0$ *for all* $i = 1, \ldots, m$ *and* $c_j > 0$, *then* $x_j = u_j$.

Turning now to integer programming problems, preprocessing can sometimes be taken a step further. Obviously, if $x_j \in Z^1$ and the bounds l_j or u_j are not integer, we can tighten to

$$\lceil l_j \rceil \leq x_j \leq \lfloor u_j \rfloor.$$

For mixed integer programs with variable upper and lower bound constraints $l_j y_j \leq x_j \leq u_j y_j$ with $y_j \in \{0,1\}$, it is also important to use the tightest bound information.

For *BIPs* it is common to look for simple "logical" or "boolean" constraints involving only one or two variables, and then either add them to the problem or use them to fix some variables. Again we demonstrate by example.

Example 7.5 Consider the set of constraints involving four 0–1 variables:

$$\begin{array}{rrrrcl} 7x_1 & +3x_2 & -4x_3 & -2x_4 & \leq & 1 \\ -2x_1 & +7x_2 & +3x_3 & +x_4 & \leq & 6 \\ & -2x_2 & -3x_3 & -6x_4 & \leq & -5 \\ 3x_1 & & -2x_3 & & \geq & -1 \\ & & x & \in & B^4. & \end{array}$$

Generating Logical Inequalities. Examining row 1, we see that if $x_1 = 1$, then necessarily $x_3 = 1$, and similarly $x_1 = 1$ implies $x_4 = 1$. This can be formulated with the linear inequalities $x_1 \leq x_3$ and $x_1 \leq x_4$. We see also that the constraint is infeasible if both $x_1 = x_2 = 1$ leading to the constraint $x_1 + x_2 \leq 1$.

Row 2 gives the inequalities $x_2 \leq x_1$ and $x_2 + x_3 \leq 1$.

Row 3 gives $x_2 + x_4 \geq 1$ and $x_3 + x_4 \geq 1$.

Row 4 gives $x_1 \geq x_3$.

Combining Pairs of Logical Inequalities. We consider pairs involving the same variables.

From rows 1 and 4, we have $x_1 \leq x_3$ and $x_1 \geq x_3$, which together give $x_1 = x_3$.

From rows 1 and 2, we have $x_1 + x_2 \leq 1$ and $x_2 \leq x_1$ which together give $x_2 = 0$. Now from $x_2 + x_4 \geq 1$ and $x_2 = 0$, we obtain $x_4 = 1$.

Simplifying. Making the substitutions $x_2 = 0, x_3 = x_1, x_4 = 1$, all four constraints of the feasible region are redundant, and we are left with $x_1 \in \{0,1\}$, so the only feasible solutions are $(1,0,1,1)$ and $(0,0,0,1)$. ∎

In Exercise 7.10, the reader is asked to formalize the approach taken in this example. The logical inequalities can also be viewed as providing a foretaste of the valid inequalities to be developed in the next chapter.

7.7 NOTES

7.2 The first paper presenting a branch-and-bound algorithm for integer programming is [LanDoi60]. [Litetal63] presents a computationally successful application to the *TSP* problem using an assignment relaxation. [Balas65] developed an algorithm for 0–1 problems using simple tests to obtain dual bounds and check primal feasibility.

7.4 Almost all commercial codes since the 1960s have been linear programming based branch-and-bound codes. The two–way branching scheme commonly used is from [Dak65].

7.5 A discussion of important elements of commercial codes can be found in [Beal79]. GUB/SOS branching is from [BealTom70], probing from [GuiSpi81], and strong branching from [Appetal95]. One important new idea is constraint branching, used for *TSP* problems in [CloNad93], and by [CooRutetal93] in their implementation of basis reduction for integer programming based upon the fundamental paper [Len83]. Recent experiments with various branch-and-bound strategies are reported in [LinSav97].

As solving linear programs forms such an important part of an integer programming algorithm, improvements in solving linear programs are crucial. All recent commercial codes include an interior point algorithm, as for many large linear programs, the latter algorithm is faster than the simplex method. However, because reoptimization with the simplex method is easier than with interior point codes, the simplex method is still used in branch-and-bound. Improving reoptimization with interior point codes is a major challenge for the next few years. See [RooTerVia97] and [Wri97] for recent texts on interior point algorithms. Work on solving integer programs with interior point algorithms is a wide open area [MitTod92],[Mit96].

Knapsack problems, in which the linear programming relaxations can be solved by inspection, have always been treated by specialized codes; see the book [MarTot90].

7.6 Preprocessing is crucially important for the rapid solution of linear programs. Its importance for integer programs is recognized in [BreMitWil73], and discussed more recently in [HofPad91],[Sav94].

7.8 EXERCISES

1. Consider the enumeration tree (minimization problem) in Figure 7.11:

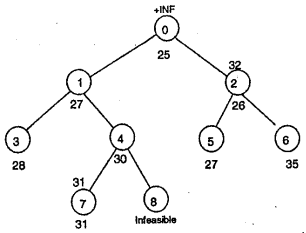


Fig. 7.11 Enumeration tree (min)

(i) Give tightest possible lower and upper bounds on the optimal value z.
(ii) Which nodes can be pruned and which must be explored further?

2. Consider the two-variable integer program:

$$\begin{array}{lrcrcl} \max & 9x_1 & + & 5x_2 & & \\ & 4x_1 & + & 9x_2 & \leq & 35 \\ & x_1 & & & \leq & 6 \\ & x_1 & - & 3x_2 & \geq & 1 \\ & 3x_1 & + & 2x_2 & \leq & 19 \\ & & x & \in & Z^2_+. & \end{array}$$

Solve by branch-and-bound graphically and algebraically.

3. Consider the 0–1 knapsack problem:

$$\max\{\sum_{j=1}^{n} c_j x_j : \sum_{j=1}^{n} a_j x_j \leq b, x \in B^n\}$$

with $a_j, c_j > 0$ for $j = 1, \ldots, n$.

(i) Show that if $\frac{c_1}{a_1} \geq \ldots \geq \frac{c_n}{a_n} > 0$, $\sum_{j=1}^{r-1} a_j \leq b$ and $\sum_{j=1}^{r} a_j > b$, the solution of the LP relaxation is $x_j = 1$ for $j = 1, \ldots, r-1$, $x_r = (b - \sum_{j=1}^{r-1} a_j)/a_r$, and $x_j = 0$ for $j > r$.
(ii) Solve the instance

$$\begin{gathered} \max 17x_1 + 10x_2 + 25x_3 + 17x_4 \\ 5x_1 + 3x_2 + 8x_3 + 7x_4 \leq 12 \\ x \in B^4 \end{gathered}$$

by branch-and-bound.

4. Solve the integer knapsack problem:

$$\begin{gathered} \max 10x_1 + 12x_2 + 7x_3 + \tfrac{3}{2}x_4 \\ 4x_1 + 5x_2 + 3x_3 + 1x_4 \leq 10 \\ x_1, x_2 \in Z^1_+, x_3, x_4 \in \{0, 1\} \end{gathered}$$

by branch-and-bound.

5. (i) Solve the $STSP$ instance with $n = 5$ and distance matrix

$$(c_e) = \begin{pmatrix} - & 10 & 2 & 4 & 6 \\ - & - & 9 & 3 & 1 \\ - & - & - & 5 & 6 \\ - & - & - & - & 2 \end{pmatrix}$$

by branch-and-bound using a 1-tree relaxation (see Definition 2.3) to obtain bounds.
(ii) Solve the TSP instance with $n = 4$ and distance matrix

$$(c_{ij}) = \begin{pmatrix} - & 7 & 6 & 3 \\ 3 & - & 6 & 9 \\ 2 & 3 & - & 1 \\ 7 & 9 & 4 & \end{pmatrix}$$

by branch-and-bound using an assignment relaxation to obtain bounds.
(iii) Describe clearly the branching rules you use in (i) and (ii), and motivate your choice.

6. Using a branch-and-bound system, solve your favorite integer program with different choices of branching and node selection rules, and report on the differences in the running time and the number of nodes in the branch-and-bound tree.

7. *Reduced cost fixing.* Suppose that the linear programming relaxation of an integer program has been solved to optimality, and the objective function is then represented in the form

$$z = \max cx, cx = \bar{a}_{00} + \sum_{j \in NB_1} \bar{a}_{0j} x_j + \sum_{j \in NB_2} \bar{a}_{0j}(x_j - u_j)$$

where NB_1 are the nonbasic variables at zero, and NB_2 are the nonbasic variables at their upper bounds u_j, $\bar{a}_{0j} \leq 0$ for $j \in NB_1$, and $\bar{a}_{0j} \geq 0$ for $j \in NB_2$. In addition suppose that a primal feasible solution of value $\underline{z}$ is known. Prove the following: In any optimal solution,

$$x_j \le \lfloor \tfrac{\bar{a}_{00}-\underline{z}}{-\bar{a}_{0j}} \rfloor \text{ for } j \in N_1, \text{ and}$$
$$x_j \ge u_j - \lceil \tfrac{\bar{a}_{00}-\underline{z}}{\bar{a}_{0j}} \rceil \text{ for } j \in N_2.$$

8. Consider a fixed charge network problem:

$$\min\{cx + fy : Nx = b, x \le uy, x \in R^n_+, y \in Z^n_+\}$$

where N is the node-arc incidence matrix of the network, and b the demand vector. In using priorities, suggest a preferred direction for the variables.

9. Consider the 0–1 problem:

$$\begin{array}{lrcrcrcrcl} \max & 5x_1 & - & 7x_2 & - & 10x_3 & + & 3x_4 & - & 5x_5 & \\ & x_1 & + & 3x_2 & - & 5x_3 & + & x_4 & + & 4x_5 & \le & 0 \\ & -2x_1 & - & 6x_2 & + & 3x_3 & - & 2x_4 & - & 2x_5 & \le & -4 \\ & & & 2x_2 & - & 2x_3 & - & x_4 & + & x_5 & \le & -2 \\ & & & & & & x & \in & B^5. & & & \end{array}$$

Simplify using logical inequalties.

10. *Logical.* Given a set in 0–1 variables

$$X = \{x \in B^n : \sum_{j=1}^{n} a_j x_j \le b\}$$

with $a_j \ge 0$ for $j = 1, \ldots, n$, under what conditions is

(i) the set X empty?
(ii) the constraint $\sum_{j=1}^{n} a_j x_j \le b$ redundant?
(iii) the constraint $x_j = 0$ valid?
(iv) the constraint $x_i + x_j \le 1$ valid?

Apply these rules to the first constraint in Exercise 9.

11. Prove Proposition 7.3 concerning preprocessing.

12. Let

$$X = \{x \in B^n : \sum_{j=1}^{n} a_j x_j \le b\}$$

with $a_1 \ge a_2 \ge \ldots \ge a_n \ge 0$ and $b \ge 0$. The idea is to write each such set in some simple canonical form. For example, for $x \in B^3$, $12x_1 + 8x_2 + 3x_1 \le 14$ is equivalent to $2x_1 + 1x_2 + 1x_3 \le 2$.

(i) When $n = 2$, how many distinct knapsack sets are there? Write them out in a canonical form with integral coefficients and $1 = a_1 \ge a_2$.

(ii) Repeat for $n = 3$ with $a_1 \le 2$.

13*. Some small integer programs are very difficult for mixed integer programming systems. Try to find a feasible solution to the integer equality knapsack: $\{x \in Z^n_+ : \sum_{j=1}^{n} a_j x_j = b\}$ with $a = (12228, 36679, 36682, 48908, 61139, 73365)$ and $b = 89716837$.

14. Suppose that $P^i = \{x \in R^n : A^i x \le b^i\}$ for $i = 1, \ldots, m$ and that $C_k \subseteq \{1, \ldots, m\}$ for $k = 1, \ldots, K$. A *disjunctive program* is a problem of the form

$$\max\{cx : x \in \cup_{i \in C_k} P^i \text{ for } k = 1, \ldots, K\}.$$

Show how the following can be modeled as disjunctive programs:

(i) a 0–1 integer program.
(ii) a linear complementarity problem: $w = q + Mz, w, z \in R^m_+, w_j z_j = 0$ for $j = 1, \ldots, m$.
(iii) a single machine sequencing problem with job processing times p_j, and variables t_j representing the start time of job j for $j = 1, \ldots, n$.
(iv) the nonlinear expression $z = \max\{3x_1 + 2x_2 - 3, 9x_1 - 4x_2 + 6\}$.
(v) the constraint: if $x \in R^1_+$ is positive, then x lies between 20 and 50, and is a multiple of 5.

15. Devise a branch-and-bound algorithm for a disjunctive program.

References

T. Cormen, C. Leiserson, R. Rivest, and Stein. *Introduction to algorithms*. MIT press, Cambridge, Massachusetts, USA, third edition, 2009.

G. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In T. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 339–347. John Wiley & Sons, New York, 1951.

J. Matoušek and B. Gärtner. *Understanding and Using Linear Programming*. Springer Berlin Heidelberg, 2007. doi: 10.1007/978-3-540-30717-4.

R. Vanderbei. *Linear Programming: Foundations and Extensions*, volume 114 of *International Series in Operations Research & Management Science*. Springer US, third edition, 2008. doi: 10.1007/978-0-387-74388-2. URL http://www.princeton.edu/~rvdb/LPbook/index.html.

H. P. Williams. *Model building in mathematical programming*. John Wiley & Sons, Chichester, fifth edition, 2013. ISBN 9780471997887. URL http://site.ebrary.com/lib/sdub/docDetail.action?docID=10657847.