

Operations Research and Networks

Operations Research and Networks

Edited by
Gerd Finke

Series Editor
Pierre Dumolard

ISTE

 WILEY

First published in France in 2002 by Hermes Science/Lavoisier entitled: "Recherche opérationnelle et réseaux: Méthodes d'analyse spatiale"

First published in Great Britain and the United States in 2008 by ISTE Ltd and John Wiley & Sons, Inc.

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms and licenses issued by the CLA. Enquiries concerning reproduction outside these terms should be sent to the publishers at the undermentioned address:

ISTE Ltd
27-37 St George's Road
London SW19 4EU
UK

www.iste.co.uk

John Wiley & Sons, Inc.
111 River Street
Hoboken, NJ 07030
USA

www.wiley.com

© ISTE Ltd, 2008

© LAVOISIER, 2002

The rights of Gerd Finke to be identified as the author of this work have been asserted by him in accordance with the Copyright, Designs and Patents Act 1988.

Library of Congress Cataloging-in-Publication Data

Operations research and networks / edited by Gerd Finke.

p. cm.

Includes bibliographical references and index.

ISBN 978-1-84821-092-9 (alk. paper)

1. Network analysis (Planning) 2. Operations research. I. Finke, Gerd.

T57.85.O64 2008

658.4'032--dc22

2008030338

British Library Cataloguing-in-Publication Data

A CIP record for this book is available from the British Library

ISBN: 978-1-84821-092-9

Printed and bound in Great Britain by Antony Rowe Ltd, Chippenham, Wiltshire.



Mixed Sources

Product group from well-managed
forests and other controlled sources

Cert no. SGS-COC-2953
www.fsc.org
© 1996 Forest Stewardship Council

Table of Contents

Introduction	ix
Gerd FINKE	
Chapter 1. Linear Programming	1
Gerd FINKE and Nadia BRAUNER	
1.1. Fundamental concepts	1
1.2. Software.	9
1.3. Indivisible units	14
1.4. Modeling with integer variables.	22
1.5. Conclusion	27
1.6. Bibliography	28
Chapter 2. Graphs and Networks	29
Wojciech BIENIA	
2.1. The concept of a graph	29
2.2. Sub-structures and exploration	31
2.3. Edge- and vertex-connectivity.	34
2.4. Directed graphs	36
2.5. Valued graphs and networks.	37
2.5.1. The shortest spanning tree problem in a connected graph	37
2.5.2. The shortest path	41
2.6. Assignment and coloring.	46
2.6.1. Matchings	46
2.6.2. Vertex colorings.	48
2.7. Flow in networks	53
2.8. Conclusion	68
2.9. Bibliography	68

Chapter 3. Classical Combinatorial Problems and Solution Techniques . . . 71 Clarisse DHAENENS, Marie-Laure ESPINOUSE and Bernard PENZ

3.1. Introduction.	71
3.2. Classical optimization problems	72
3.2.1. Introduction	72
3.2.2. Combinatorial optimization problems in graph theory	72
3.2.3. Assignment Problems	73
3.2.4. Transportation problems	74
3.2.5. Location problems	75
3.2.6. Scheduling problems	76
3.3. Complexity	77
3.4. Solution of hard problems	80
3.4.1. Introduction	80
3.4.2. Relaxations.	81
3.4.3. Heuristic methods of construction	81
3.4.4. Improvement methods	83
3.4.5. Exact methods	90
3.5. Conclusion	101
3.6. Bibliography	102

Chapter 4. Project Scheduling 105 Lionel DUPONT

4.1. Presentation.	105
4.1.1. Conducting a project	105
4.1.2. Definitions	106
4.1.3. Scheduling methods	108
4.2. Scheduling and graphs without cycles	108
4.3. Fundamental problem.	113
4.3.1. Calculation of the earliest dates.	113
4.3.2. Calculation of the latest dates	114
4.3.3. Margins	116
4.4. Visualizations	117
4.4.1. Representation on the graph PERT/CPM	117
4.4.2. Gantt chart	118
4.5. Probabilistic PERT	119
4.5.1. Analytic solution	120
4.5.2. Solution by simulation	122
4.6. Sequencing with disjunctive constraints	123
4.7. Sequencing with cumulative constraints: serial methods	126
4.8. Time-cost trade-off problem	131
4.9. Conclusion	135
4.10. Bibliography	135

Chapter 5. Operations Management in Transportation Networks. 137

Jacques DESROSIERS

5.1. Introduction.	137
5.1.1. A bit of history	137
5.1.2. University–industry: a winning partnership	138
5.2. Fundamental notions	139
5.2.1. A common structure	139
5.2.2. The shortest path problem with time windows	140
5.2.3. Some mathematics	141
5.2.4. A generic algorithm.	142
5.3. A mechanism of decomposition.	145
5.3.1. Local restrictions and global constraints	145
5.3.2. The column generation method.	148
5.3.3. Solutions satisfying the integrality constraints	150
5.4. Diversity of the local restrictions	151
5.4.1. A few words on the extension functions	151
5.4.2. Modeling examples	154
5.5. Applications in large transportation networks	156
5.5.1. Urban transportation	156
5.5.2. Air transportation	157
5.5.3. Rail transportation	158
5.6. What does the future look like?	159
5.7. Bibliography	160

Chapter 6. Pickup and Delivery Problems with Services on Nodes or Arcs of a Network 165

Alain HERTZ and Michel MITTAS

6.1. Introduction.	165
6.2. Node routing problems	166
6.2.1. The traveling salesman problem	166
6.2.2. Vehicle tours with capacity constraints	170
6.3. Arc routing problems	174
6.3.1. The Chinese postman problem	174
6.3.2. The rural postman problem	177
6.3.3. Arc routing problems with capacity constraints	183
6.4. Conclusion	185
6.5. Bibliography	186

Chapter 7. Telecommunication Networks 189

Alexandre CAMINADA, Jin-Kao HAO, Jean-Luc LUTTON and Vincent MARTIN

7.1. Introduction.	189
7.2. Optimal synthesis of backbone networks	190
7.3. Topology and dimensioning of the nominal network	193
7.3.1. Descent algorithm.	194

7.3.2. Simulated annealing	197
7.4. Dimensioning of spare capacities	200
7.4.1. Non-simultaneous multiflow model and linear programming	200
7.4.2. Solution of non-simultaneous multiflows by decomposition.	203
7.4.3. Extension of the decomposition model for modular capacities	206
7.5. Conception of cellular networks	208
7.6. Antenna positioning and configuration	210
7.6.1. Multicriteria model	212
7.6.2. A heuristic for positioning antennas	215
7.7. Frequency assignment	220
7.7.1. Modeling by set T-coloring	223
7.7.2. Solution by evolutionary algorithms	224
7.8. Conclusion	228
7.9. Bibliography	229
Chapter 8. Mission Planning for Observation Satellites	235
Virginie GABREL, Cécile MURAT and Vangelis PASCHOS	
8.1. Introduction.	235
8.2. Description of the problem of planning satellite shots.	237
8.2.1. Context: image acquisitions – constraints of the satellite used	237
8.2.2. Problems studied	239
8.3. Models and formulations of induced combinatorial problems	241
8.4. Algorithms for MS and MSO	246
8.4.1. Solving MS	246
8.4.2. Solving MSO in the discrete case	251
8.4.3. Solving MSO in the continuous case	251
8.5. Experiments and numerical results	257
8.5.1. Performances of approximate algorithms proposed to solve MS	257
8.5.2. Performances of approximate algorithms proposed to solve MSO	259
8.6. Conclusion	261
8.7. Bibliography	261
List of Authors	263
Index	265

Introduction

This volume presents the principal operations research (OR) tools that help in the planning and management of all sorts of networks. The term “network” is to be understood in a very broad sense. In effect, this term also designates physical networks, such as road or railway networks, as well as logical networks, used for complex project planning for example. In this case, the network elements correspond to activities, and the interconnections describe temporal relations.

OR considers real problems in two phases: a modeling phase that corresponds to the development of a mathematical model is followed by a solution procedure, exact or approximate, and in general algorithmic. These two phases are closely connected and OR proposes collections of models, adapted to numerous applications, as well as the corresponding resolution techniques.

The models can be classified in two large categories, those based on an algebraic formulation and those founded on the concept of graphs. These two models are presented in Chapters 1 and 2 and the general solution approaches are in Chapter 3. Then, in Chapters 4 to 8, some chosen applications are detailed. They concern project management, collection and distribution networks, telecommunication networks and graph models for mission planning of observation satellites.

Chapter 1 presents the main ideas of linear programming. This model is based on basic notions of linear algebra. The goal is to optimize a linear function (maximize profit, minimize distance covered, minimize transportation cost, etc.) by satisfying

certain conditions (constraints). These constraints are linear (equations or inequations) and model the use of finite capacity resources, distance limits, budget restrictions, etc.

The fundamental concepts of linear programming are described: the type of solution (basic solution), duality and the economic interpretation. However, the goal of this chapter is situated more at the level of modeling than developing theory. Software is easily accessible, mostly in standard form. The objective of this chapter is to describe the use of a type of software and to show how the results can be interpreted. Modeling by linear programming is very flexible and enables an efficient solution to the problems, thanks to the simplex algorithm. But there is an essential restriction: fractional value solutions must be accepted. Unfortunately, for many of the problems, discrete variables (of integer values) must be considered and they often take values 0 or 1 indicating a decision (a site, chosen or not, or a resource, used or not). For these integer programming problems, an efficient universal algorithm does not exist. The task of creating a “good” model becomes much more difficult and requires us to exploit the particular structure of the problem.

Graphs are extremely useful structures to describe and understand numerous problems. These impressive modeling tools are detailed in Chapter 2. The graphs are made up of vertices (depots, cities, clients) and of connections between these vertices (routes, cables). Often other information is given, for example the quantity of available merchandise in a vertex and the length or the cost of a connection (edge, arc) connecting two vertices. These graphs, called valued graphs, constitute what are called abstract networks. In this chapter, a large number of examples illustrate the richness of this model. The usual concepts are introduced – paths, trees and flows – with sometimes unexpected applications. Graph theory has created its own algorithmic solution. These procedures can be an alternative to linear programming with integer variables, or even in certain cases give the only possible access to an efficient solution.

Chapter 3 presents the spectrum of diverse solution techniques. Some elements of the complexity theory are given, distinguishing the easy problems, such as linear programming in fractional variables or minimal length paths in graphs, and difficult problems, such as integer linear programming or the traveling salesman problem. Solution techniques are presented, including exact methods (branch and bound, dynamic programming) and, in particular, approximate methods: heuristics and metaheuristics (simulated annealing, tabu method and genetic algorithms).

Chapter 4 addresses the management and scheduling of large projects and, in particular, describes the Program Evaluation and Review Technique (PERT). It is a quite general tool, applicable to complex projects, such as construction of an

apartment building, a ship or a dam, as well as large agricultural projects. This method is based on a network of events whose arcs indicate the succession of the different work phases necessary to realize the project. Without giving an exhaustive list, some large spatial problems are described in the following chapters.

In Chapter 5, the management of large urban transportation networks, both air and rail, is considered. Real applications are numerous: school transportation, transport of the handicapped, assignment and determination of the hours of bus drivers and locomotive engineers and crew rotation for an airline. It is remarkable that for this multitude of applications a sole structure, described by paths that respect a time interval in a space-time network, is applicable. The number of possible paths, and thus the number of variables of choice in these models, can reach several million. Consequently, even the linear relaxations of the problem can no longer be solved directly by the simplex method. Other more elaborate procedures must be used; for example the method of column generation, the object of recent research.

Chapter 6 presents other aspects in transportation networks that are connected to the distribution and collection of goods to the clients. Two types of models emerge, depending on the position of the clients. If they are found on the vertices of the network (cities for example), the problem studied is a problem of vehicle tours, related to the traveling salesman problem. On the other hand, if the clients are situated along the roads, the problems lead to so-called Chinese postman problems.

In Chapter 7, large telecommunication networks are studied. Many actors intervene in these networks and the quality of service becomes dominant for the users. There are two large categories of telecommunication networks: fixed networks with cable connections and mobile networks where communications are made by radioelectric waves for clients who are moving in a certain territory. The methods are different for these two types of networks. A whole range of OR techniques is applied here: linear programming or algorithms in graphs, for example coloring for the frequency assignment in a mobile radio network.

Finally (Chapter 8), the last problem presented concerns mission planning for observation satellites. Satellite users demand images of terrestrial strips to be made while the satellite turns around the earth on its orbit. A uniform graph model is developed to describe a sequence of images without conflict and respecting all the constraints.

Thus, based on the two types of OR formulations, linear programming and graphs, numerous applications, especially in networks, have been described and modeled. By then calling upon OR solution techniques it was possible to obtain efficient algorithmic solution methods, exact or approximate, in order to help in the decision-making process.

Chapter 1

Linear Programming

1.1. Fundamental concepts

This chapter concerns the problems whose objective is to optimize a function by respecting constraints. Linear programming, more precisely, studies the cases where the function to optimize (or objective function) and the constraints are expressed linearly. This model, which can seem rather specialized, is in fact very flexible. We will show in this chapter that it allows us to model a great many important applications.

Numerous studies describe linear programming techniques. We cite, not wanting to give a long list, some reference studies in French [CHA 96, MIN 83] and in English [BAZ 90, CHV 83, DAN 63, MUR 85, SAK 83, NEM 88].

We first describe an example that illustrates the allocation of resources: the company Biereco produces two varieties of beer, B_1 and B_2 , from three ingredients: corn, hops and malt. These cereals are available in quantities of 80, 30 and 40 parts. Each barrel of beer B_1 uses one part corn, one part hops and two parts malt, while each barrel of B_2 must have two parts corn, one part hops and one part malt. A barrel of B_1 is sold at 50 monetary units and a barrel of B_2 is sold at 40 monetary units. The objective of the producer is to maximize the profit. Table 1.1 gives the data for this problem.

Beer	Composition/barrel			Selling price
	corn	hops	malt	
B_1	1	1	2	50
B_2	2	1	1	40
Available	80	30	40	

Table 1.1. Data for the Biereco example

We model this problem with the help of a linear program. The decision variables x_1 and x_2 represent the number of barrels of beer B_1 and B_2 to be produced. In order for the solution obtained to be realistic, these variables must be non-negative. The limits on the quantity of available resources and the composition of the beers involve the following constraints:

$$(\text{corn}) \quad 1 \cdot x_1 + 2 \cdot x_2 \leq 80$$

$$(\text{hops}) \quad 1 \cdot x_1 + 1 \cdot x_2 \leq 30$$

$$(\text{malt}) \quad 2 \cdot x_1 + 1 \cdot x_2 \leq 40$$

The profit to maximize is given by the function $z = 50x_1 + 40x_2$. We thus obtain the linear program below:

$$(\text{objective function}) \quad \text{maximize } z = 50x_1 + 40x_2$$

$$(\text{constraints}) \quad 1x_1 + 2x_2 \leq 80$$

$$1x_1 + 1x_2 \leq 30$$

$$2x_1 + 1x_2 \leq 40$$

$$x_1 \geq 0, x_2 \geq 0$$

The linearity of the objective function and the constraints is natural for this type of problem. In effect, the profits and the consumption of the resources are additives.

Figure 1.1 describes the set of feasible solutions R : the elements of R satisfy all the constraints. The constraints are inequations. They are thus represented by half-planes (in the figure, the arrows indicate which half-plane is to be considered). By definition, R is the intersection of these half-planes.

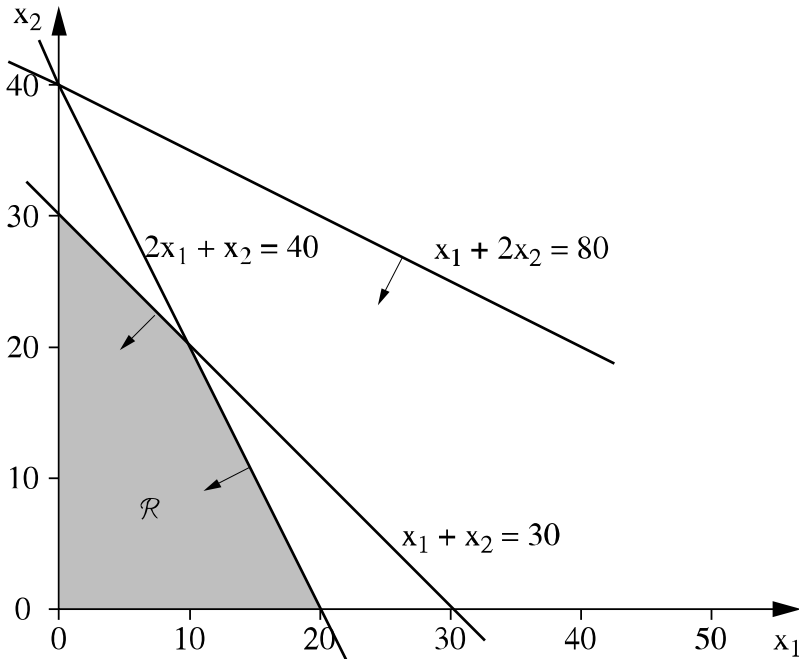


Figure 1.1. *Admissible region R*

The region R represents the candidates (x_1, x_2) in competition to maximize the objective function. Let us consider the straight lines $50x_1 + 40x_2 = z$ where z is a constant. On each of these lines, the profit remains the same (Figure 1.2).

The optimal solution to the Biereco problem is $x_1 = 10$ and $x_2 = 20$ and the maximum profit is $z = 1,300$. This example enables us to come to several conclusions:

- The constraint of corn is oversized. In effect, no feasible solution attains the limit of 80 available units.

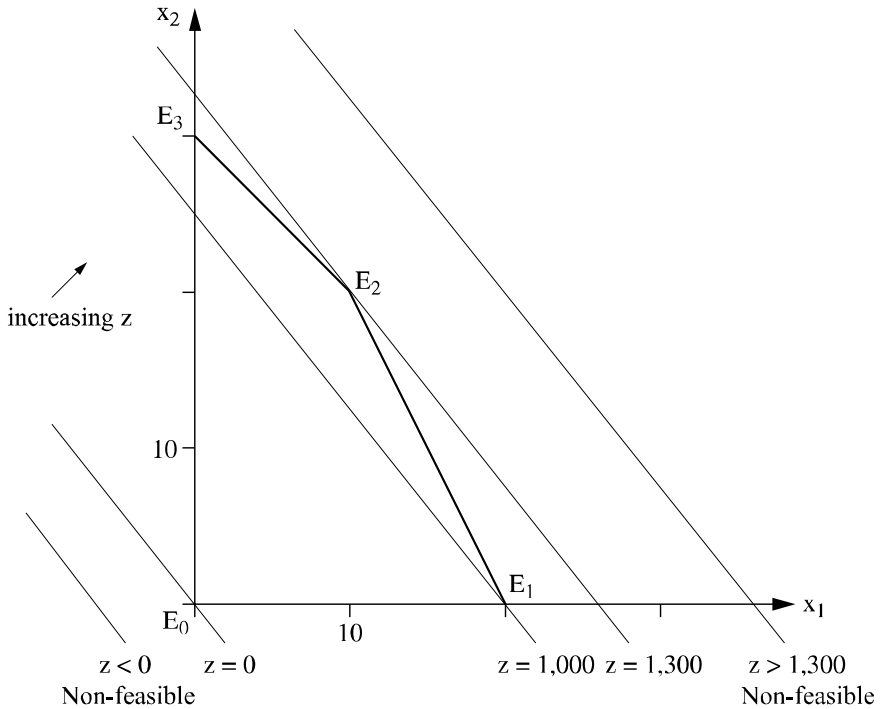


Figure 1.2. *Different values of the objective function*

– The extreme points of R , $E_0 = (0, 0)$, $E_1 = (20, 0)$, $E_2 = (10, 20)$ and $E_3 = (0, 30)$ are particularly interesting. In each of these points, at least two constraints are saturated. In this example, E_2 is optimal and it saturates the constraints of hops and malt. The construction of Figure 1.2 shows that each linear objective function admits an optimal solution in an extreme point. This optimal extreme point is not necessarily unique. For example, let us modify the objective: $z' = 60x_1 + 30x_2$. In this case, the extreme points E_1 and E_2 are both optimal.

– A point that saturates several constraints simultaneously is not necessarily an extreme point. For example, the point (0.40) saturates the three constraints $x_1 \geq 0$, $x_1 + 2x_2 \leq 80$ and $2x_1 + x_2 \leq 40$, but is non-feasible. However, when the number of constraints is finite, the admissible region can only contain a finite number of extreme points.

The celebrated simplex method that allows us to solve a linear program in fact determines the best extreme point. The basic idea is simple. The algorithm first

constructs an initial extreme point, then it passes successively to neighboring extreme points until an optimal point is attained. For the preceding example, the method begins with the origin E_0 ; then two paths toward the optimal point are possible: $E_0 \rightarrow E_1 \rightarrow E_2$ and $E_0 \rightarrow E_3 \rightarrow E_2$.

In the Biereco example, the constraints are in the form “ \leq ”. This reflects the fact that the resources are in limited quantities and that we cannot use more than the available quantity. In general, we distinguish three classes of constraint:

$$\sum_j a_{ij} x_j \leq b_i \text{ or } \geq b_i \text{ or } = b_i, i = 1, 2, \dots, m; j = 1, 2, \dots, n$$

In the preceding example, the number of variables is $n = 2$, the number of constraints is $m = 3$ and the three constraints are of the type “ \leq ”. The non-negativity constraints of the variables are treated separately and are directly integrated in the simplex algorithm. The constraints of type “ \geq ” indicate that a minimal resource quantity must be used and the constraints of type “ $=$ ” impose the exact consumption of a resource.

An equation can always be written as two inequations “ \leq ” and “ \geq ” and we transform a constraint of type “ \geq ” into a constraint of type “ \leq ” by multiplying it by (-1) . We also note that $\max cx = -\min(-cx)$. Consequently, each maximization problem is equivalent to a minimization problem. Every linear program can thus be written in a normalized form, also called a *canonical form*:

$\max z = cx$ under the constraints $Ax \leq b, x \geq 0$ where

– A is the matrix of size $m \times n$ of technical coefficients;

– $x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ is the column of decision variables;

– $b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$ is the column of limited resource quantities;

– $c = (c_1, c_2, \dots, c_n)$ is the vector of coefficients (unit profits in our example) of the objective function.

The canonical form of the Biereco example is:

$$\max z = (50, \ 40) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

subject to constraints (s.t.)

$$\begin{pmatrix} 1 & 2 \\ 1 & 1 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 80 \\ 30 \\ 40 \end{pmatrix}; \quad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Another important form of writing a linear program is *the standard form* for which the constraints are equations:

$$\max z = cx$$

$$\text{s.t. } Ax = b, x \geq 0.$$

For example, for the constraint of corn, we introduce a new variable, s_I , called the slack variable, which measures the amount of unused corn:

$$x_I + 2x_2 + s_I = 80; s_I \geq 0.$$

The standard form of the Biereco example is written as follows:

$$\max z = (50, \ 40, \ 0, \ 0, \ 0) \begin{pmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

s.t.

$$\begin{pmatrix} 1 & 2 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} 80 \\ 30 \\ 40 \end{pmatrix}; \quad \begin{pmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The simplex method uses the standard form. It is interesting for us to know what the intuitive concept of extreme points has become in this linear system.

To characterize the extreme points, we give some classical definitions of linear programming. In general, the system $Ax = b$ has more variables than constraints, $m \leq n$, and the matrix A is of full rank, $r(A) = m$. A submatrix B of A of dimension $m \times m$ is a basis if its determinant is different from zero, $\det(B) \neq 0$. More precisely, B is a collection of m columns of A taken in any order. With each basis B , we associate a solution of the linear system: let x^B be the vector of variables associated with the columns of A that are part of B , and x^N , the vector containing the other variables. The elements of x^B are the basic variables and those of x^N are the non-basic variables.

If $x^N = 0$, then the system $Ax = b$ is reduced to $Bx^B = b$ whose unique solution is $x^B = B^{-1}b$. The solution $x^N = 0$, $x^B = B^{-1}b$ is called the basic solution associated with B . If this solution is feasible, that is $x^B = B^{-1}b \geq 0$, we have an admissible basic solution. The admissible basic solutions correspond to the extreme points (Table 1.2).

The idea of the simplex method is to search for an optimal basis. At each iteration, we generate an admissible neighboring basis of the current basis by pivoting. For that, a new variable (column) is introduced into the basis and a variable (column) is eliminated from the basis.

It is not our intention to give more details about this solution procedure. However, the concept of admissible bases is essential for comprehension of the files of results supplied by classic linear programming software.

Base B	Inverse B^{-1}	Basic solution	Extreme points
$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$x^N = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ $x^B = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} 80 \\ 30 \\ 40 \end{pmatrix}$	E_0 $(x_1, x_2) = (0, 0)$
$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & -\frac{1}{2} \\ 0 & 1 & -\frac{1}{2} \\ 0 & 0 & \frac{1}{2} \end{pmatrix}$	$x^N = \begin{pmatrix} x_1 \\ s_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ $x^B = \begin{pmatrix} s_1 \\ s_2 \\ x_1 \end{pmatrix} = \begin{pmatrix} 60 \\ 10 \\ 20 \end{pmatrix}$	E_1 $(x_1, x_2) = (20, 0)$
$\begin{pmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 & -3 & 1 \\ 0 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}$	$x^N = \begin{pmatrix} s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ $x^B = \begin{pmatrix} s_1 \\ x_2 \\ x_1 \end{pmatrix} = \begin{pmatrix} 30 \\ 20 \\ 10 \end{pmatrix}$	E_2 $(x_1, x_2) = (10, 20)$
$\begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & -2 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix}$	$x^N = \begin{pmatrix} x_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ $x^B = \begin{pmatrix} s_1 \\ x_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} 20 \\ 30 \\ 10 \end{pmatrix}$	E_3 $(x_1, x_2) = (0, 30)$

Table 1.2. Admissible bases and extreme points

1.2. Software

Very powerful linear programming software exists: CPLEX, LINDO, OSL, XPRESS, AIMMS, to mention only a few. Browsing in [SAL 94] gives the description of about thirty commercial software programs. Without publicizing any particular product, we use a system whose reduced version is available free on Internet: the solver is CPLEX and the modeler is MPL (see www.maximal-usa.com). We detail the writing of linear programs and the interpretation of the results with the help of some examples.

In our first example, we want to model a problem of a canned fruit factory. For the fruit, we use peaches in halves or in pieces. The peaches are of quality Q_1 or Q_2 . For the peaches in halves, only the first quality fruit, Q_1 , is chosen, while for the peaches in pieces, we take a mixture of fruit of qualities Q_1 and Q_2 . However, in order to maintain a minimal product quality we want to have, for the peaches in pieces, at least 3 peaches of quality Q_1 for any 2 peaches of Q_2 . The decision variables of the model are:

- x_1 , the number of kg of quality Q_1 peaches cut in halves;
- x_2 , the number of kg of quality Q_1 peaches cut in pieces;
- x_3 , the number of kg of quality Q_2 peaches cut in pieces.

With a net profit of 20, 10 and 12 for each kg of x_1 , x_2 and x_3 , we obtain the linear program below:

$$\max z = 20x_1 + 10x_2 + 12x_3 \quad [\text{maximize the profit}]$$

s.t.

$$x_1 \leq 80 \quad [\text{limit of peaches in halves}]$$

$$x_2 + x_3 \leq 51 \quad [\text{limit of peaches in pieces}]$$

$$x_1 + x_2 \leq 150 \quad [\text{quality } Q_1 \text{ peaches available}]$$

$$x_3 \leq 50 \quad [\text{quality } Q_2 \text{ peaches available}]$$

$$-2x_2 + 3x_3 \leq 0 \quad [\text{quality of peaches in pieces}]$$

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

The modeler MPL enables the direct writing of small size models. The entry file is presented as follows (the non-negativity of the variables is imposed by default):

```
TITLE "Canning factory";

MODEL MAX z = 20 x1 + 10 x2 + 12 x3;

SUBJECT TO

    x1      <= 80;

    x2 + x3 <= 51;

    x1 + x2 <= 150;

    x3 <= 50;

    -2x2 + 3x3 <= 0;

END
```

The solution is found in the output file obtained:

```
SOLUTION RESULT

    optimal solution found

    MAX z = 2150.8000
```

DECISION VARIABLES

PLAIN VARIABLES

Variable Name	Activity	Reduced Costs
x1	80.0000	0.0000
x2	30.6000	0.0000
x3	20.4000	0.0000

CONSTRAINTS

PLAIN CONSTRAINTS

Constraint Name	Slack	Shadow Price
c1	0.0000	20.0000
c2	0.0000	10.8000
c3	39.4000	0.0000
c4	29.6000	0.0000
c5	0.0000	0.4000

The solution given by the software is thus $x_1 = 80$, $x_2 = 30.6$ and $x_3 = 20.4$ with a maximal profit of $z = 2150.8$. The “Slack” column indicates the values of the slack variables s_j of the constraints c_j ($j = 1, 2, \dots, 5$). For example, the slack variable of the third constraint has a value of $s_3 = 150 - (x_1 + x_2) = 150 - 110.6 = 39.4$. This means that, for the given solution, there remain 39.4 kg of quality Q_1 peaches.

The second columns “Reduced Costs” and “Shadow Price” are more subtle. The simplex method not only solves the problem posed (the primal problem $[P]$), but also a second linear program (the dual problem $[D]$) that is solved simultaneously. The primal program is written in the canonical form as follows:

$$\begin{aligned}
 [P] \quad & \max \quad z = cx \\
 & \text{s.t.} \quad Ax \leq b \\
 & \quad \quad x \geq 0
 \end{aligned}$$

We associate with $[P]$ the following program:

$$\begin{aligned}
 [D] \quad & \min \quad v = yb \\
 & \text{s.t.} \quad yA \geq c \\
 & \quad \quad y \geq 0
 \end{aligned}$$

For the example of canned peaches, we have:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & -2 & 3 \end{pmatrix}; \quad b = \begin{pmatrix} 80 \\ 51 \\ 150 \\ 50 \\ 0 \end{pmatrix}; \quad c = (20, \ 10, \ 12)$$

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}; \quad y = (y_1, \ y_2, \ y_3, \ y_4, \ y_5)$$

The slack variables are denoted $s \geq 0$ for $[P]$ and $t \geq 0$ for $[D]$. We note that t is a surplus, and a constraint of the form $y_1 + y_3 \geq 20$ is transformed to $y_1 + y_3 - t_1 = 20$ and $t_1 \geq 0$. The two programs can be compared in the following manner:

$\begin{aligned} \max \quad & z = 20x_1 + 10x_2 + 12x_3 \\ \text{s.t.} \quad & \\ & x_1 \qquad \qquad \qquad + \quad s_1 = \quad 80 \\ & \qquad x_2 \quad + \quad x_3 \quad + \quad s_2 = \quad 51 \\ & x_1 \quad + \quad x_2 \qquad \qquad + \quad s_3 = 150 \\ & \qquad \qquad x_3 \quad + \quad s_4 = \quad 50 \\ & -2x_2 \quad + \quad 3x_3 \quad + \quad s_5 = \quad 0 \\ & \qquad \qquad \qquad x_1 \geq \quad 0 \quad y_1 \qquad + y_3 \qquad \qquad - \quad t_1 = \quad 20 \\ & \qquad \qquad \qquad x_2 \geq \quad 0 \qquad y_2 \quad + y_3 \qquad \qquad -2y_5 \quad - \quad t_2 = \quad 10 \\ & \qquad \qquad \qquad x_3 \geq \quad 0 \qquad y_2 \qquad \qquad + y_4 \quad + 3y_5 \quad - \quad t_3 = \quad 12 \end{aligned}$	$\begin{aligned} \min \quad & v = 80y_1 + 51y_2 + 150y_3 + 50y_4 \\ \text{s.t.} \quad & \\ & y_1 \geq \quad 0 \\ & y_2 \geq \quad 0 \\ & y_3 \geq \quad 0 \\ & y_4 \geq \quad 0 \\ & y_5 \geq \quad 0 \end{aligned}$
--	--

In the solution file, the solution of the dual program is found in the column “Shadow Price”, $y_1 = 20$, $y_2 = 10.8$, $y_3 = 0$, $y_4 = 0$ and $y_5 = 0.4$ and the slack variables of the dual program in the column “Reduced Costs”, $t_1 = t_2 = t_3 = 0$. The optimal value of the dual program is the same as the optimal value of the primal program $v = 2150.8 = z$.

We can give an economic interpretation of the dual variables. In effect, the variable y_i gives the value that we should be ready to pay for each supplementary

unit (kg) of the resource i . For example, this price is $y_2 = 10.8$ for each supplementary kg of peaches in pieces.

As suggested by the example indicated above, we can couple the variables s_i with y_i , and x_j with t_j . In effect, the conditions of the complementary slackness indicate that $s_i \times y_i = 0$ and $x_j \times t_j = 0$. For example, $s_3 = 39.4$ implies $y_3 = 0$. This reflects the fact that the available quantity of quality Q_I peaches is not used up. Thus, an augmentation of these peaches is without profit, that is, the value of peaches of quality Q_I in this system is zero.

The output file also gives other interesting information:

OBJECTIVE RANGES

PLAIN VARIABLES

Variable Name	Coefficient	Lower Range	Upper Range
x1	20.0000	0.0000	1E+020
x2	10.0000	-8.0000	12.0000
x3	12.0000	10.0000	1E+020

RANGES RHS

PLAIN CONSTRAINTS

Constraint Name	RHS Value	Lower Bound	Upper Bound
c1	80.0000	0.0000	119.4000
c2	51.0000	0.0000	116.6667
c3	150.0000	110.6000	1E+020
c4	50.0000	20.4000	1E+020
c5	0.0000	-102.0000	148.0000

These two tables describe the sensitivity intervals of the problem data. The interval (Lower Range, Upper Range) in the first table gives the admissible variation for the coefficients c_j ($j = 1, 2, 3$) of the objective function for which the basis (and consequently the solution x_j) remains optimal. This is a local property that describes the stability of the solution when we modify only one given number. In general, these effects are not cumulative.

Similarly, the intervals (Lower Bound, Upper Bound) give the variations of the quantities b_i of the resources so that the basis remains optimal. However, the solution associated with this basis changes. In effect, when B^{-1} is fixed, $x^B = B^{-1}b$ is a function of the resource vector b . In addition, the prices y_i are applicable exactly in their sensitivity interval. Consequently, a supplement of 5 kg of peaches in pieces gives a profit of $5y_2 = 5 \times 10.8 = 54$ monetary units.

Linear programming software is very efficient. It can solve problems containing thousands of variables and constraints. It has become an indispensable tool for planning and logistics in an industrial environment.

1.3. Indivisible units

In the preceding example, the variables can take fractional values. Thus, the variable x_2 is valued at 30.6 kg or 30600 g. But let us consider the following problem: in a maritime region, empty containers are found at ports P_1 (5 containers), P_2 (7 container) and P_3 (8 containers). Containers are demanded in the cities V_1 (6 containers), V_2 (3 containers), V_3 (4 containers) and V_4 (5 containers). Transfer of empty containers can be done in the network described in Figure 1.3. The unit cost of transportation, in thousands of monetary units, is indicated on the connections. The objective is to find a container distribution plan at minimum cost and to decide in which ports the two surplus containers must be stored. The transfers are made on the best routes. For example, an empty container goes from P_2 to V_4 on the path $P_2 \rightarrow V_1 \rightarrow P_3 \rightarrow V_4$ with a total cost of 11.

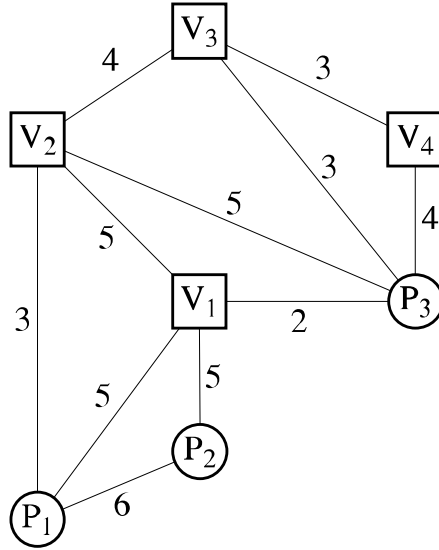


Figure 1.3. Network of transfer costs

Generally, the data of a transportation problem are:

- the vector $stock(i)$ for the depots $i = 1, 2 \dots m$;
- the vector $demand(j)$ for the destinations $j = 1, 2 \dots n$;
- the matrix $cost(i, j)$ of the unit costs of sending one unit from the depot i to the destination j .

As Figure 1.3 suggests, transportation problems can be solved by using another fundamental operational research tool, namely graph theory. This approach has its own algorithmic (see Chapter 2). We continue to develop the preceding example with the algebraic version of linear programs. A transportation problem can be formulated in the following manner:

$$\min cost = \sum_{i=1}^m \sum_{j=1}^n cost(i, j) x_{ij}$$

$$\begin{aligned}
\text{s.t. } \sum_{j=1}^n x_{ij} &\leq \text{stock}(i) & i=1,2\dots m & \text{ (stock limitation in } i) \\
\sum_{i=1}^m x_{ij} &\geq \text{demand}(j) & j=1,2\dots n & \text{ (demand satisfaction)} \\
x_{ij} &\geq 0 & x_{ij} & \text{ integer}
\end{aligned}$$

where the variable x_{ij} indicates the number of containers sent from depot i to destination j . The modeler MPL allows a very compact formulation of this problem. The complete file associated with this example is:

```

TITLE "Transportation problem";

DATA

  m=3;

  n=4;

INDEX

  i=1...m;

  j=1...n;

DATA

  stock[i]:= (5,7,8);

  demand[j]:= (6,3,4,5);

  cost[i,j]:= (5,3,7,10,5,9,10,11,2,5,3,4);

DECISION VARIABLES

  x[i,j];

MODEL

  MIN total cost = SUM(i,j: cost*x);

SUBJECT TO

  depot[i]: SUM(j:x) <= stock;

  city[j]: SUM(i:x) >= demand;

END

```

The CPLEX solver gives the following solution: $x_{12} = 3$; $x_{13} = 1$; $x_{21} = 6$; $x_{33} = 3$; $x_{34} = 5$; a container remains in stock in each of the ports P_1 and P_2 . The transfers to be made are described in Figure 1.4.

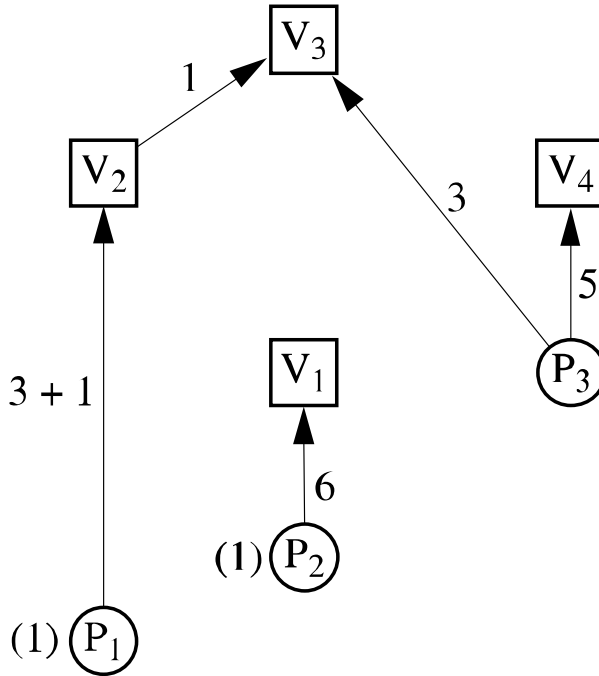


Figure 1.4. Container transfer

Since it is a question of numbers of containers to transfer, only integer solutions are feasible. Transportation problems have the remarkable property that all the basic solutions (and consequently the optimal solutions obtained by the simplex method) are integers if the data $stock(i)$ and $demand(j)$ are integers. In other words, the inverse basis B^{-1} does not contain any fractional values. In effect, the elements of B^{-1} are of the form $cofactor/detB$ and $detB = \pm 1$. The coefficient matrices of the constraint that have this property are called *totally unimodular*.

The transportation problem is a model that is found in every toolbox of operations research methods. Well solved by universal linear programming software, totally unimodular problems (such as transportation problems) are therefore easy cases.

Let us study the following problem. A city with a bilingual population is divided into school zones. The students in the city must learn a foreign language and have the choice between English and German. Two high schools can accept the students: North High School and South High School. The distribution of the choices and distances to travel according to the zones of origin of the students are given in Table 1.3.

Zone	Language (number of students)		Distance (km)	
	German	English	North High School	South High School
1	50	200	1	2
2	50	250	2	1
3	100	150	1	1

Table 1.3. Data of the problem of assigning students to high schools

The city government wants each of the two city high schools to accept between 20% and 30% of the students who chose German. They equally want to have 300 to 500 students enrolled in each high school. The objective is to assign students from different zones to the high schools so that the total distance covered by them is minimized.

The data of the problem are two matrices:

– $student(i, j)$, the number of students from zone $i = 1, 2, 3$, of language $j = E$ (English), D (German);

– $distance(i, k)$, the distance from zone $i = 1, 2, 3$ to the high school $k = North, South$.

The variable x_{ijk} represents the number of students from zone i , of language j assigned to high school k . The problem of assigning the students to the high schools can in such a case be modeled as follows:

$$\begin{aligned}
& \min \sum_{i,j,k} distance(i,k)x_{ijk} \\
& \text{s.t.} \quad \sum_k x_{ijk} = student(i,j) \quad \forall i,j \\
& \quad \quad 300 \leq \sum_{i,j} x_{ijk} \leq 500 \quad \forall k \\
& \quad \quad 0.2 \sum_{ij} x_{ijk} \leq \sum_i x_{iDk} \leq 0.3 \sum_{ij} x_{ijk} \quad \forall k \\
& \quad \quad x_{ijk} \geq 0 \quad x_{ijk} \text{ integers}
\end{aligned}$$

The MPL software finds a minimum of 800 km. The associated solution is fractional (Table 1.4). Most software programs propose a command that enables us to declare integer variables. For the MPL modeler, this command is `INTEGER x[i,j,k]`.

In fact, the total distance of 800 km is optimal and an optimal integer solution can be obtained by rounding the fractional values (Table 1.4).

Zone	High school	Language	Fractional solution	Integer solution
1	North	D	50	50
1	North	E	200	200
1	South	D	0	0
1	South	E	0	0
2	North	D	0	0
2	North	E	0	0
2	South	D	50	50
2	South	E	250	250
3	North	D	87.5	87
3	North	E	150	150
3	South	D	12.5	13
3	South	E	0	0

Table 1.4. Solution to the problem of student assignment

The preceding example is simple. The integer solution is found in the neighborhood of the fractional solution. Unfortunately, the situation is not always so simple. Let us consider the following linear program:

$$\begin{array}{llll}
 \min & z = x_1 & & \\
 \text{s.t.} & x_1 - 17x_2 & & = 3 \\
 & x_1 & - 11x_3 & = 4 \\
 & x_1 & & - 6x_4 = 5 \\
 & x_1, x_2, x_3, x_4 \geq 0
 \end{array}$$

We know that $z = x_1 = 6x_4 + 5 \geq 5$. Consequently, a solution for which $x_1 = 5$ is optimal: $z = 5$; $x_1 = 5$; $x_2 = 2/17$; $x_3 = 1/11$; $x_4 = 0$.

If we impose the integrality of the variables x_1, x_2, x_3 and x_4 (that is, x_i are integers for $i = 1, 2, 3, 4$), we are surprised to discover that the solution and the optimal value have completely changed. We obtain: $z = 785$; $x_1 = 785$; $x_2 = 46$; $x_3 = 71$; $x_4 = 130$.

Another very intuitive approach exists that can seem attractive. Let us consider the admissible region of a hypothetical linear program (Figure 1.5).

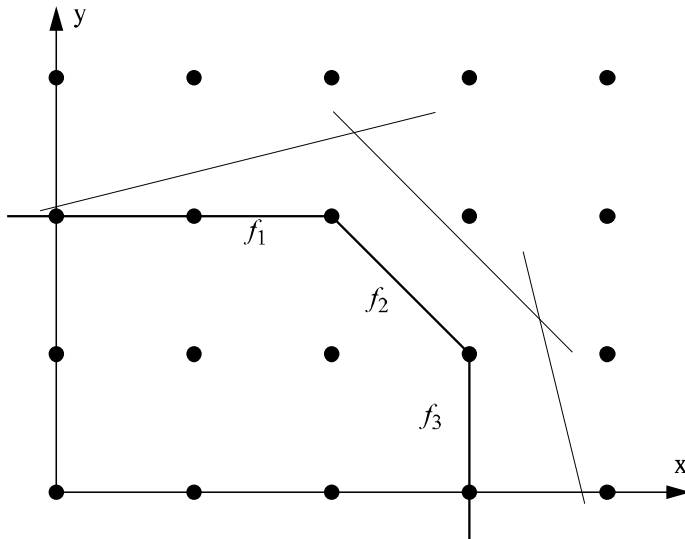


Figure 1.5. Convex hull of integer solutions

We concentrate on integer points in the admissible region and we determine the contours (or facets) f_1, f_2, f_3 that form the convex hull of integer solutions. We can thus use the standard simplex method with the constraints f_1, f_2, f_3 . By construction, the optimum is one of the integer extreme points.

The following example in the plane (x, y) shows the disaster that such an approach can bring [RUB 70]. We consider below the well-known Fibonacci sequence:

$$S_1 = 1; S_2 = 1; S_k = S_{k-1} + S_{k-2}; k = 3, 4, \dots$$

which gives

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55 \dots$$

Let R_k be the admissible region in the plane (x, y) defined by

$$S_{2k}x + S_{2k+1}y \leq S_{2k+1}^2 - 1$$

$$x, y \geq 0.$$

Each R_k is simply a triangle with three extreme points. A linear program with an objective function of the form $ax + by$ can be solved in a trivial manner since one of these three extreme points is optimal. But let us consider I_k , the convex hull of the integer points of R_k . For example, for the case $k = 3$, I_3 is given by $8x + 13y \leq 168$; $x \geq 0$; $y \geq 0$; x and y are integers. We can verify that I_3 has 6 facets and 6 extreme points. In general, it is shown that the region I_k has $(k + 3)$ facets and extreme points. By choosing an appropriate objective, we can force the simplex method, which in general starts at the origin, to make about $k/2$ iterations. Consequently, by choosing k large enough, any software will have to make as many iterations as we wish.

Commercial software has evolved a lot during the last ten years, in particular for the integration of integer variables. The character of the solution procedure with or without integer variables is completely different. Recent linear programming solution methods in integer variables are enumerative: we traverse a decision tree and we solve, with the simplex algorithm, a linear program for each vertex. The method loses its universality and obtaining an optimal integer solution is not guaranteed for instances of large size.

1.4. Modeling with integer variables

In the preceding section, the linear program applications presented concern fractional variables and integer variables that describe non-divisible units such as containers, people, etc. The linear models in integer variables are in fact quite rich. Numerous applications enable such formulations. Problems that *a priori* are non-linear can be linearized. In this case we must introduce a large number of binary variables (variables with values 0 or 1).

We present a problem for the choice of sites that illustrates this aspect. An enterprise wishes to construct k communication centers and has n possible locations to install these centers ($k \leq n$). A center installed in location i has a functioning cost of a_{ii} , $i = 1$ to n . The cost of communication from center i to center j is a_{ij} (the matrix $A = (a_{ij})$ is positive). The objective is to position the centers in order to minimize the total cost (functioning and communication). For this problem, we analyze different models in the form of linear programs in integer variables.

We define the following variables:

$x_i = 1$ if a center is established at location i ; and $x_i = 0$, if not.

A quadratic program can then model the positioning of the centers:

$$\begin{aligned} \min \quad & \sum_{ij} a_{ij} x_i x_j \\ \text{s.t.} \quad & \sum_i x_i = k, x_i \in \{0, 1\} \end{aligned}$$

Let us set $y_{ij} = x_i x_j$ ($y_{ii} = x_i x_i = x_i$). The preceding program becomes:

$$\begin{aligned} \min \quad & \sum_{ij} a_{ij} y_{ij} \\ \text{s.t.} \quad & \sum_i y_{ii} = k, \end{aligned}$$

The constraints $y_{ij} = x_i x_j$ can be formulated in several ways. We compare four linear formulations.

VERSION 1.

$$y_{ii} + y_{jj} - 1 \leq y_{ij} \text{ for all } i, j$$

$$y_{ij} \in \{0, 1\} \text{ for each } i, j$$

These constraints are redundant except if $y_{ii} = y_{jj} = 1$. In this case, $y_{ij} = 1$ and the cost of communication between centers i and j must be paid.

VERSION 2. In version (1), it is sufficient to impose $y_{ii} \in \{0, 1\}$ and $y_{ij} \geq 0$. In effect, the objective is to minimize a function whose coefficients are non-negative. Thus, the y_{ij} take the smallest value possible, that is, either 0 or 1.

VERSION 3.

$$y_{ii} + y_{jj} \geq 2y_{ij} \text{ for all } i, j$$

$$\sum_i \sum_{j \neq i} y_{ij} = k(k-1)$$

$$y_{ij} \in \{0, 1\} \text{ for all } i, j$$

The first constraint implies that if $y_{ij} = 1$, then $y_{ii} = y_{jj} = 1$. The second constraint imposes the correct number of connections. Consequently, if $y_{ii} = y_{jj} = 1$, then $y_{ij} = 1$.

VERSION 4. We replace, in version 3, the second constraint by stronger constraints:

$$\sum_j y_{ij} = ky_{ii} \text{ for all } i$$

We want to analyze the efficiency of these models. To do that, we use the following data: $n = 12$; $k = 5$ and

$$a_{ij} = \begin{pmatrix} 1 & 2 & 3 & 5 & 4 & 6 & 7 & 8 & 7 & 8 & 1 & 2 \\ 5 & 1 & 23 & 4 & 21 & 2 & 2 & 1 & 5 & 1 & 3 & 5 \\ 8 & 7 & 1 & 5 & 1 & 4 & 2 & 6 & 4 & 7 & 1 & 3 \\ 21 & 1 & 5 & 42 & 3 & 4 & 54 & 7 & 2 & 1 & 1 & 5 \\ 54 & 12 & 45 & 3 & 12 & 4 & 5 & 4 & 1 & 7 & 2 & 4 \\ 12 & 1 & 45 & 64 & 6 & 9 & 3 & 13 & 4 & 5 & 4 & 2 \\ 2 & 4 & 5 & 7 & 2 & 54 & 6 & 8 & 7 & 1 & 5 & 1 \\ 1 & 8 & 7 & 2 & 3 & 5 & 11 & 3 & 5 & 11 & 2 & 1 \\ 1 & 2 & 4 & 65 & 4 & 6 & 7 & 5 & 2 & 1 & 5 & 1 \\ 12 & 12 & 5 & 49 & 16 & 17 & 52 & 4 & 2 & 5 & 4 & 2 \\ 1 & 5 & 4 & 3 & 4 & 5 & 46 & 7 & 21 & 1 & 5 & 5 \\ 2 & 4 & 5 & 1 & 2 & 6 & 4 & 5 & 2 & 1 & 2 & 1 \end{pmatrix}$$

Table 1.5 summarizes the results obtained for the different models of the problem (versions 1 to 4). The “linear relaxation” line indicates the optimal value of the objective function if the constraint $y_{ij} \in \{0, 1\}$ is replaced by the constraint $0 \leq y_{ij} \leq 1$. The “number of vertices” corresponds to the size of the decision tree covered, thus to the number of linear programs solved. The total number of bases generated by the simplex method is given by the “number of iterations”.

	V. (1)	V. (2)	V. (3)	V. (4)
optimal value	80	80	80	80
number of iterations	476	407	2884	1848
number of vertices	28	25	246	53
time(s) (sec)	2:64	1:59	10:10	8:29
linear relaxation	17	17	36.9	45.49

Table 1.5. *Computation time and results for the different versions*

The performances of these four versions are very different. For small size problems, such as the one presented above, version 2 is the most efficient. For large

size problems, enumerative methods must be used. In this case, it is the quality of the linear relaxation that counts and versions 3 and 4 are preferable instead. To decide what model is the most appropriate is often the result of experimentation. In Chapter 3, models, formulations and classical operations research solution methods are described.

We present a last example that shows the limit of universal software.

A truck driver must make four deliveries with only one truck. The duration p_j of the delivery L_j is known. This duration p_j includes loading time, driving to the client, unloading and returning to the depot with an empty truck. We assume that this return time is negligible. The truck can only make one delivery at a time and must return to the depot between two deliveries. The clients impose the delivery deadlines d_j and each day of delay is penalized by w_j . The problem data are described in Table 1.6.

delivery L_j	duration p_j (days)	delay d_j (days)	penalty w_j (monetary units per day)
1	2	4	200
2	5	6	100
3	7	10	400
4	4	5	200

Table 1.6. *Data of the delivery problem*

The objective is to minimize the total cost of the lateness penalties. If deliveries are made in the order L_1, L_2, L_3, L_4 , we can evaluate the penalties as indicated in Table 1.7.

L_j	end of $L_j : c_j$	delay $t_j = \max\{c_j - d_j, 0\}$	penalty $w_j t_j$
L_1	2	0	0x200
L_2	7	1	1x100
L_3	14	4	4x400
L_4	18	13	13x200
			total $\sum w_j t_j = 4300$

Table 1.7. Penalties for the order L_1, L_2, L_3, L_4

The optimal permutation is L_1, L_4, L_3, L_2 and the minimal penalty is $\min \sum w_j t_j = 2600$.

The formulation of this type of problem in form of a linear program in integer variables is a little more elaborate. The data are the n deliveries L_1, L_2, \dots, L_n , their duration p_j , the delay d_j and the penalties w_j . The continuous variables s_j designate the instant of the beginning of delivery L_j , ($c_j = s_j + p_j$ is the end date of L_j).

The $n!$ permutations are expressed by linear constraints. For each delivery pair (L_i, L_j) with $i < j$, we must decide if

- L_i is before L_j , that is $s_i + p_i \leq s_j$ or
- L_j is before L_i , that is $s_j + p_j \leq s_i$.

We define the variables $y_{ij} \in \{0, 1\}$ and we impose the following two constraints:

$$My_{ij} + s_i \geq p_j + s_j \text{ and } M(1 - y_{ij}) + s_j \geq p_i + s_i$$

where M is a sufficiently large constant. Thus, if $y_{ij} = 0$, then $s_i \geq p_j + s_j$ and $M + s_j \geq p_i + s_i$. In this case, L_j is executed before L_i and the second constraint is redundant. If $y_{ij} = 1$, then $M + s_i \geq p_j + s_j$ and $s_j \geq p_i + s_i$. Thus, the first constraint is redundant and L_i is executed before L_j . Delays are represented by the continuous variables $t_j \geq 0$ that verify the constraint:

$$s_j + p_j - d_j \leq t_j \text{ for all } j$$

The objective is to minimize $\sum_j w_j t_j$. With random data, we obtain the results given in Table 1.8.

n	4	5	6	7	8	9	10
calculation time (sec)	0:61	0:75	0:92	1:08	1:81	5:06	21:17
number of iterations	43	66	164	269	670	2,811	10,286
number of vertices	6	17	38	60	164	711	3,662

Table 1.8. *Solution times for the delivery problem*

For $n = 13$, the software attains the limit of 300 constraints authorized in the reduced version, but for $n = 12$ deliveries, the computation time explodes. In effect, it is greater than 15 minutes with a number of vertices greater than 159,250 and the number of iterations greater than 338,670. There are a number of theoretical reasons (of complexity) that explain this mediocre performance. For this type of “difficult” problem, we must often be content with approximate methods (see Chapter 3).

1.5. Conclusion

We have presented the fundamental concepts of modeling problems in the form of a linear program, as well as a highly capable solution tool. Numerous applications have illustrated the utility and the richness of this model.

With the simplex method we have a universal procedure, capable of solving large size problems of any structure, on the condition that fractional variables are acceptable.

However, the most common problems include integer and binary variables. For this type of model, there is no universal solution method. We are obliged to exploit the particular structure of a problem and to test the efficiency of diverse formulations. Solution approaches become enumerative and we must often be content with approximate solutions. In the following chapters, the most important models will be presented and studied.

1.6. Bibliography

- [BAZ 90] M.S. Bazaraa, J.J. Jarvis, and H.D. Sherali, *Linear Programming and Network Flows*, 2nd edn, John Wiley & Sons, 1990.
- [CHA 96] I. Charon, A. Germa, and O. Hudry, *Méthodes d'optimisation combinatoire*, Masson (Collection pédagogique de télécommunications), 1996.
- [CHV 83] V. Chvatal, *Linear Programming*, W.H. Freeman and Company, New York, 1983.
- [DAN 63] G.B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, 1963.
- [MIN 83] M. Minoux, *Programmation mathématique, théorie et algorithmes*, volume 1, Dunod, CNET-ENST edition, 1983.
- [MUR 85] K.G. Murty, *Linear and Combinatorial Programming*, R.E. Krieger, 1985.
- [NEM 88] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley & Sons, 1988.
- [RUB 70] D.S. Rubin, "On the unlimited number of faces in integer hulls of linear programs with a single constraint", *Operations Research*, vol. 18, 940–946, 1970.
- [SAK 83] M. Sakarovitch, *Linear Programming*, Springer-Verlag, 1983.
- [SAL 94] M.S. Saltzman, "Broad selection of software packages available", *OR/MS Today*, vol. 21 (2), 42–51, April 1994.

Chapter 2

Graphs and Networks

2.1. The concept of a graph

To illustrate a complex situation we always try to make the maximum simplifications in order not to overload our description with unimportant details. Let us imagine that Figure 2.1 represents four cities connected by a system of expressways. The heavy traffic necessitates two roads between 1 and 2 and a bypass around 4. The diagram is very simplified. The distances between the cities are not respected and the cities are symbolized only by points. The exact layout of each expressway is not detailed and the figure only shows if the expressway directly connecting two cities exists or not. These simplifying steps are at the basis of the concept of a graph. The only data retained are the points and the existence of relations between the pairs of points.

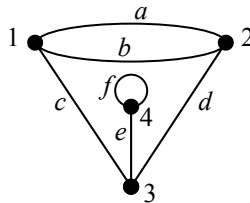


Figure 2.1. *A graph*

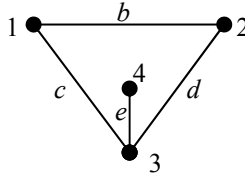


Figure 2.2. A simple graph

To define an (*undirected*) graph $G = (X; E)$ it suffices to specify the set of points $X = \{x_1, x_2, \dots, x_n\}$ where $n \geq 1$ (vertices of the graph) and the family of its edges $E = \{e_1, e_2, \dots, e_m\}$. The edges are considered as being unordered pairs from X . For an edge $e = \{x, y\}$ the vertices x and y are called the *ends* of e . When $x = y$, e is a *loop*. The network of expressways connecting four cities is thus modeled on Figure 2.1 as the graph $G = (X; E)$ with $X = \{1, 2, 3, 4\}$, $E = \{a, b, c, d, e, f\}$ where $a = \{1, 2\}$, $b = \{1, 2\}$, $c = \{1, 3\}$, $d = \{3, 2\}$, $e = \{3, 4\}$, $f = \{4, 4\}$; f is a loop and a, b are two parallel edges. We thus consider the edges as labeled pairs. Figure 2.2 represents a *simple graph* which tells us only that certain direct connections between the different cities exist or not (we do not pay attention to the exact number of these connections).

The set of vertices connected to x by an edge is called the *neighbors* of x and noted $N(x)$. The *degree* $d(x)$ of a vertex x is the number of edges having x as end (a loop is counted twice). If $d(x) = 0$ the vertex x is *isolated*.

A *complete graph*, noted K_n , is a simple graph with n vertices where any two vertices are adjacent. Figure 2.3 represents the complete graph K_5 .

The following example illustrates another situation where the graph becomes an efficient tool of modeling which enables us to find the optimal organization.

EXAMPLE 2.1. To guarantee security, a bank wants to install a safe with a minimum number of locks which satisfies the following constraint: each of the five bank personnel should have a certain number of keys, but it should not be possible to open the safe unless at least three (any three of the five) personnel are present.

- What is the number of locks installed?
- What is the number of keys that each person has?

According to the terms, the opening of the safe necessitates the presence of at least three people; thus, when only two try to open the safe, there exists a lock they cannot open. This situation can be modeled by the graph where a vertex represents a

person and an edge is interpreted as a lock the two people corresponding to the ends of this edge cannot open. This graph is illustrated in Figure 2.3 where the people are numbered from 1 to 5 and the edge a represents the lock that cannot be opened by 1 and 5 alone; b the lock that cannot be opened by 3 and 5 only; c the lock 2 and 4 cannot open alone, etc. We thus obtain a simple graph (it is useless to multiply the edges or make loops!) complete with 5 vertices (K_5) and which has 10 edges. The safe has ten locks because the edges obviously represent different locks; if a is the same lock as b , then in this case the three people, 1, 3 and 5, cannot open the safe.

The answer to the second question is that each person must be in possession of six keys corresponding to the non-incident edges at the vertex that represents this person ($6 = 10 - 4$).

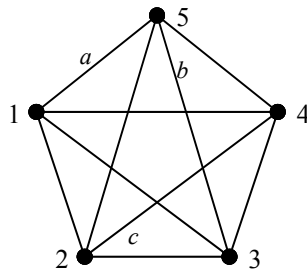


Figure 2.3. Personnel and locks

This problem, in a generalized form, is covered in [XUO 92].

2.2. Sub-structures and exploration

We are sometimes interested in the fragments of the given graph $G = (X; E)$.

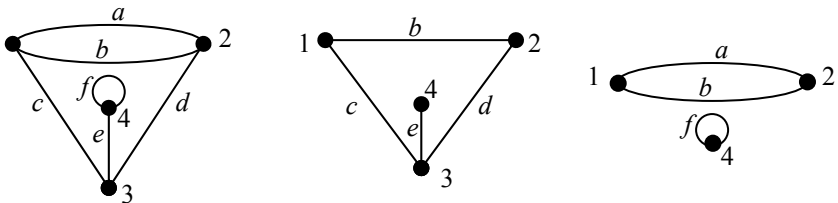


Figure 2.4. Graph G , a partial graph of G and the sub-graph induced by 1, 2, 4

When we consider all the vertices and a sub-set of edges we speak of a *partial graph*. When we consider a subset of vertices $Y \subseteq X$ and the totality of the edges of E whose both ends are in Y , we speak of a *sub-graph induced by Y* . Figure 2.4 illustrates these two concepts.

A subset of vertices that are pairwise adjacent is a *clique*. A *stable set* is a set of vertices that are pairwise non-adjacent. When we look at the central graph in Figure 2.4, the vertices 1, 2 and 3 form a clique and the set 1, 4 is a stable set. We can still partition the set of all the vertices in a graph without a loop into several stable sets. Figure 2.5 gives an example of a partition into 5 stable sets.

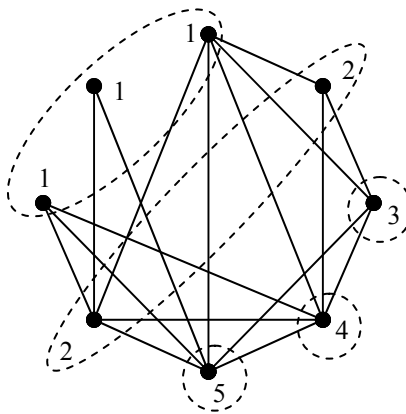


Figure 2.5. A partition of the set of vertices of a graph in 5 stable sets

The graph is *bipartite* when we can partition the sets of its vertices into two stable sets, noted $G = (X, Y; E)$. If each vertex of X is connected to each vertex of Y by an edge, we say that G is *complete bipartite* (it will be noted $K_{p,q}$ where p and q are the numbers of vertices in X and Y respectively). Two examples of bipartite graphs are represented in Figure 2.6.



Figure 2.6. Complete bipartite graphs: $K_{1,3}$ and $K_{3,3}$

Numerous applications are concerned with the interconnection between the vertices of a graph. This connection is realized by an alternate sequence of vertices and edges, and is called a *chain*. This notion corresponds to an itinerary, that is to say a series of cities to go through and roads to take in order to arrive at destination **D** from origin **O**.

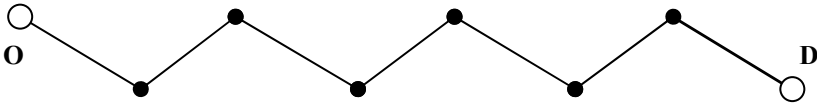


Figure 2.7. *A chain*

It is possible to cross the same city or the same road several times during a trip. If the edges of a chain are distinct, we speak of a *simple chain*; if the vertices are distinct we speak of an *elementary chain*. The chain is *closed* if this sequence begins and ends with the same vertex. A *cycle* is a simple closed chain.

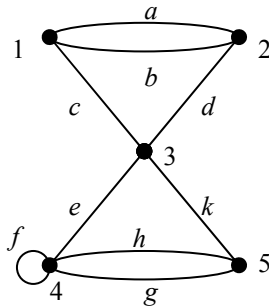


Figure 2.8. *An undirected graph*

Here are some examples, as illustrated in Figure 2.8:

- simple chain: $(4e3c1a2d3k5)$, the edges are distinct;
- elementary chain: $(1a2d3e4h5)$, the vertices are distinct;
- cycle: $(1c3e4g5k3d2a1)$, and elementary cycle: $(1c3d2b1)$.

2.3. Edge- and vertex-connectivity

A graph $G = (X; E)$ is called *connected* if, between any two vertices, there exists a chain. In graph G we call *connected component* each maximal subgraph having this property. In other words, a graph is connected if it possesses only one connected component. The notions introduced are illustrated in Figures 2.9 and 2.10.

In this section we present the elementary notions and proprieties that enable the study of these two concepts. The interested reader will be able to consult specialized works [BER 73, GOL 80].

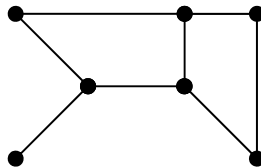


Figure 2.9. *Connected graph*

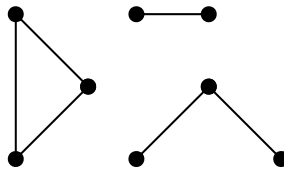


Figure 2.10. *Graph with three connected components*

We are often interested in the *degree of connection* in communication networks or transportation networks in general. If we imagine that the graph in Figure 2.8 represents a road network, we can easily see that, in order to go from 1 to 5, we can move even when the road c is blocked by an accident. On the other hand, if intersection 3 is blocked, areas 1 and 2 will be separated from 4 and 5. Two simultaneous accidents on roads c and d would have the same effect.

It is easy to verify that the graph in Figure 2.8 cannot be disconnected by the elimination of less than two edges; we say that this graph is *2-edge-connected*.

The structure of the graphs with a high level of connection is complicated. We are limited here to the presentation of graphs having the weakest connection. We call a graph that is connected and without cycle a *tree*. A tree is then a graph *1-edge-*

connected. A collection of trees is a *forest* (some vertices are not connected by any chain). These objects are illustrated in Figures 2.11 and 2.12.

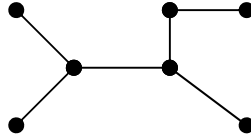


Figure 2.11. *A tree*

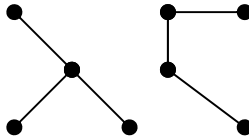


Figure 2.12. *A forest with 2 trees*

It is easy to see that a tree with n vertices always has $n-1$ edges.

The following proprieties explain that the connection in the trees is very weak:

- In a tree every pair of vertices is connected by a single chain.
- If we remove any edge from a tree, the graph is no longer connected.

An edge for which the suppression causes the disconnection of the graph is an *isthmus*. For example, each edge of a tree is an isthmus.

We are also interested in the property of connection when we successively eliminate vertices (we speak of *vertex-connectivity*). This corresponds to the case of the blocked intersections. For example, it suffices to eliminate vertex 3 in the graph in Figure 2.8 in order to obtain a subgraph with two connected components. This vertex is called an *articulation point* of the graph. A graph with at least three vertices and without an articulation point is *2-vertex-connected*.

2.4. Directed graphs

We obtain a *directed graph* $D = (X; A)$ if we order the ends of each edge $e \in E$ in an undirected graph $G = (X; E)$. Each edge is transformed into an *arc* which is a pair of vertices (*tail* and *head*). We say that $a = (x, y)$ *leaves* x and *enters* y .

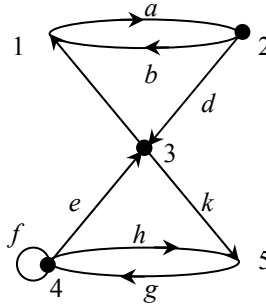


Figure 2.13. A directed graph

We note $d^-(x)$ (resp. $d^+(x)$) the *in-degree* (resp. *out-degree*) of a vertex x , that is the number of arcs entering x (resp. leaving x). In Figure 2.13 $d^+(2)=2$ and $d^-(2)=1$. We always have: $d^+(x) + d^-(x) = d(x)$. We call a *source* a vertex s with $d^-(s) = 0$. A *sink* is a vertex t with $d^+(t) = 0$.

The notion of chain can be used in the directed case, but the orientation enables it to be refined. A *path* Γ in a directed graph is an alternate sequence of vertices and arcs so that the vertex, which precedes an arc, is its tail and that which succeeds it in this sequence is its head. The path Γ is of length k if the sequence is composed of $(k+1)$ vertices and k arcs. The path (composed of at least one arc) of which the first and the last vertex coincide is a *closed path*. If the arcs are distinct, Γ is a *simple path*. If the vertices are distinct, Γ is an *elementary path*. We call every simple closed path a *circuit*.

Here are some examples, illustrated in Figure 2.13:

- simple path: $(4e3c1a2d3k5)$ – the arcs are distinct;
- elementary path: $(1a2d3k5g4)$ – the vertices are distinct;
- circuit: $(1a2d3k5g4e3c1)$, and elementary circuit: $(3k5g4e3)$.

Table 2.1 presents certain non-oriented objects with their oriented counterparts.

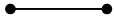



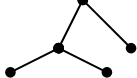
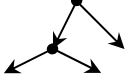
Non-oriented graph	Oriented graph
 <i>edge</i>	<i>arc</i> 
 <i>chain</i>	<i>path</i> 
 <i>tree</i>	<i>arborescence</i> 

Table 2.1. *Some oriented and non-oriented objects*

2.5. Valued graphs and networks

We can easily enrich the concept of a graph by adding supplementary information. Thus, when an edge represents, for example, a road, we can associate it with its length in kilometers or even with the time of traversal in hours. Also, if an arc represents a decision, we can associate the cost (or the profit) with this decision. Next we present two classical problems concerning valued graphs: the shortest spanning tree problem in a connected undirected graph and the shortest path problem between two vertices in a directed graph.

2.5.1. The shortest spanning tree problem in a connected graph

Let $G = (X; E)$ be a connected graph with n vertices where each edge e of E owns a real value $l(e)$, called *length*.

Let us consider a graph with lengths of edges, as represented in Figure 2.14 with, as represented in Figure 2.15, some spanning trees (that is, which contain all the vertices of the original graph) together with their lengths.

Let us suppose that this graph schematizes five water reservoirs which must be connected in order to re-equilibrate their water level. The construction of some canals is envisaged (an edge represents a possible canal) and their cost of construction is estimated (valuation of the edges). We try to determine which canals should be retained so that the total cost for the construction of the system is minimal.

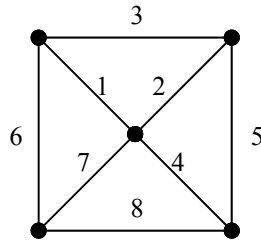


Figure 2.14. *A valued graph*

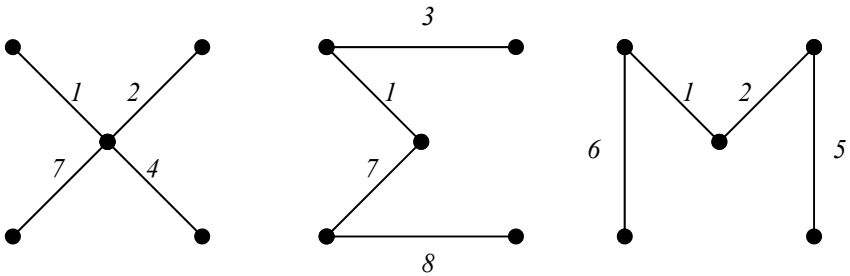


Figure 2.15. *Some spanning trees and their lengths: $l(X) = 14$, $l(\Sigma) = 19$, $l(M) = 14$*

Let us try to formalize this problem. Being given a connected graph with the edges valued by a weight (a cost), we are looking for a partial graph which must be connected. The minimal connection is sufficient; thus we are interested in the trees covering all the vertices of the graph. For each tree $A=(X;T)$ of G we define its weight as being the sum of all the lengths of its edges:

$$l(A) = \sum_{e \in T} l(e)$$

and we want to find the shortest spanning tree in G – that is, one of minimum length. In our example we will be able to make a list of all spanning trees and thus find a tree of minimum weight. In general, the number of spanning trees in a given graph is an exponential function of the number of its vertices; thus it is impossible to realize the search for the tree of minimum weight by explicit enumeration. The need for an efficient method seems evident.

Now we are going to present the Kruskal's algorithm, remarkable for the simplicity of its concept. It examines the edges of the graph in the order of their lengths. To simplify the presentation we suppose that the edges in $E = \{e_1, e_2, \dots, e_m\}$ have been numbered so that $l(e_i) \leq l(e_{i+1})$ for $i=1, 2, \dots, m-1$. Then, starting from the

set of all vertices without any edge, we add an edge at each step, the one with the smallest length among the edges not yet accepted, under the sole condition that this edge does not create a cycle with the edges already accepted. Thus each edge is examined (to be accepted or rejected) only once. This method yields a shortest spanning tree.

Let us apply this algorithm to the graph in Figure 2.14. The edges are arranged naturally; we accept 1 and 2, but reject 3 because it forms a cycle with 1 and 2 which are already chosen; we accept 4, but reject 5 (it forms a cycle with 2 and 4); we accept 6 and reject 7 and 8; we obtain a shortest spanning tree with length $l(A) = 13$, as represented in Figure 2.16.

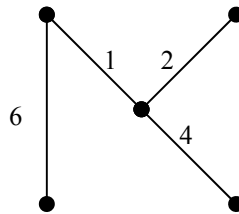


Figure 2.16. The shortest spanning tree of the graph in Figure 2.14

The concept of arranging and then examining in order is called a *greedy algorithm*. The reader interested in other efficient algorithms can consult the works [EVA 92, GOL 80].

The following example proposes an application of the shortest spanning tree model.

EXAMPLE 2.2. Figure 2.17 represents the plan of a neighborhood with the existing buildings and the sidewalks. We are trying to lay out pedestrian passages to connect the houses with each other and with the exterior sidewalk so as to reserve the largest surface for the lawn. Since the width of the pedestrian passages is normalized, it is their total length that must be minimized.

Let us replace each house and also the exterior pavement by a vertex; we thus obtain the complete graph K_{10} . Its edges are valued by the shortest distances between the houses. These distances are easy to determine by geometric considerations. This graph is not illustrated here because its virtual existence is sufficient for us to continue the reasoning.

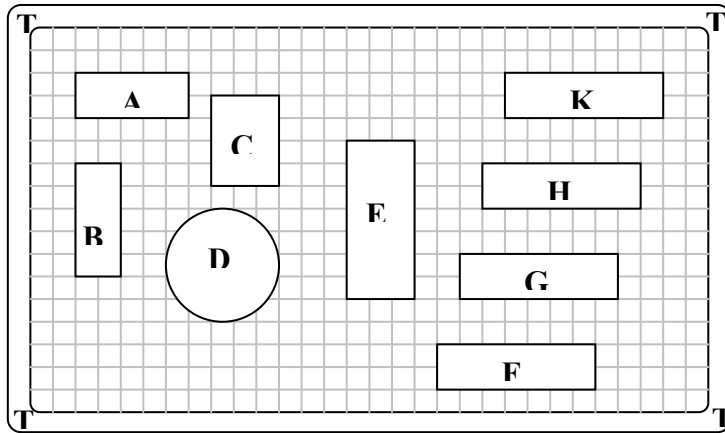


Figure 2.17. *Placement of buildings in a neighborhood*

It is not necessary to make a preliminary calculation of all the distances because the algorithm presented examines them in ascending order. Therefore, we can look for them as we go along. The distances 1 are: AC, CD and FT (T signifies the exterior sidewalk); we accept them all; the distances 2 are: AT, BT, AB, BD, EG, GF, GH, HK and KT; we accept all except AB, BD and KT. Figure 2.18 represents the shortest spanning tree thus obtained.

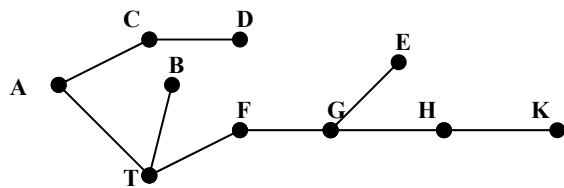


Figure 2.18. *The formal solution for the pedestrian passages*

After having applied Kruskal’s algorithm, we first obtain a formal solution that we must transfer to the application. The optimal solution is not unique because it depends first on the choice in Kruskal’s algorithm when we encounter edges with the same length and, later, its realization for the given application (see Figure 2.19).

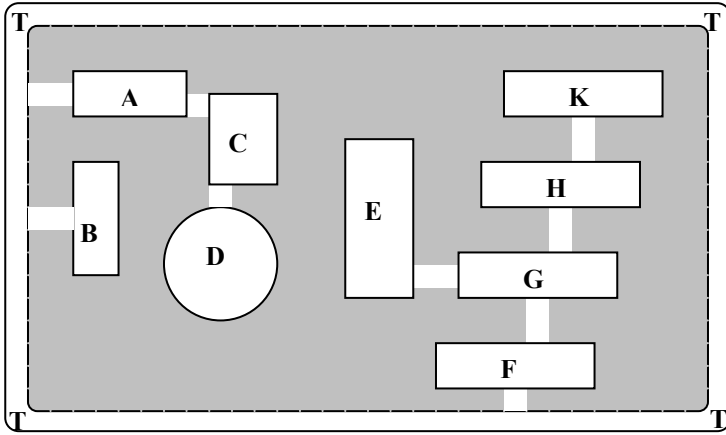


Figure 2.19. *An optimal realization of the pedestrian passages*

In our optimal solution we note also that to mow the lawn we are not obliged to cross the planned pavement. Is it a persistent property of all optimal solutions or just chance? If the planned pavement divides the lawn into two separate pieces, then from the exterior sidewalk we can cross the neighborhood and return on the exterior sidewalk. The corresponding edges of the pavement form a cycle in the graph, which models the chosen solution. This is impossible because the optimal solution is a shortest spanning tree, that is, a connected graph without cycles.

2.5.2. The shortest path

Let $G = (X; E)$ be a directed graph where each arc e of E owns a real length $d(e)$. Often we speak of *distance* associated with the arc, but since there is no restriction $d(e) \geq 0$ the value $d(e)$ can also be interpreted as the cost or the profit of a decision. For each path Γ in G we define its length:

$$\pi(\Gamma) = \sum_{e \in \Gamma} d(e)$$

that we set as zero if Γ has no arc.

Very frequently $\pi(\Gamma)$ is identified with the length of the path Γ but, evidently, this value can also be interpreted as the cost or the profit of a policy considered as a result of decisions. When the arcs all carry the value 1, the length thus defined corresponds to the number of arcs of the path.

We limit the presentation here to the problem of the shortest path between two fixed vertices x and y , but there exist adapted methods to find the set of the shortest paths between all the pairs of vertices [EVA 92, SAK 84].

We are looking for the minimal value $\pi(\Gamma)$, that is to say the shortest distance from x to y , and the path Γ that realizes this minimum.

In small graphs we can determine the shortest path by examining all the paths. In order to go from x to y in the graph illustrated in Figure 2.20, we must decide whether or not to pass through the intermediate vertices a , b , c or d ; we then find $2^4 = 16$ different paths between x and y and the shortest, $xbxy$, has the length 8.

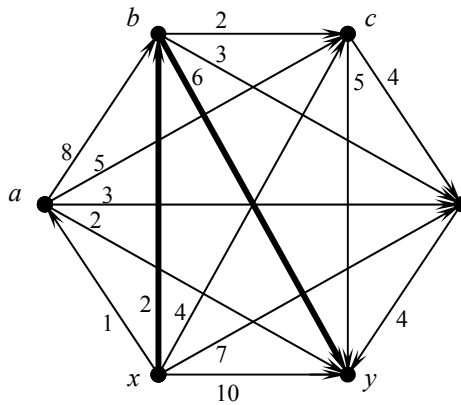


Figure 2.20. *The shortest path from x to y*

It is surprising to notice that this method becomes rapidly unexploitable because in a graph with about 60 vertices the explicit enumeration realized by a computer undertaking a million operations a second would take five centuries. This remark shows the necessity to use an efficient algorithm.

In situations similar to that of Figure 2.20, where the graph does not have a circuit, we can apply Bellman's algorithm [SAK 84]. But we cannot limit ourselves to studying only graphs without a circuit. It is evident that in the case of non-oriented graphs, where an edge can be covered in both directions, or in numerous models tied to road or air traffic or in communication networks, the circuits are indeed present. We therefore propose Dijkstra's algorithm [SAK 84], which determines the shortest path between two given vertices chosen, even in the presence of circuits in a graph. On the other hand, it is necessary that the distances assigned to the arcs are not negative.

Let us imagine that the circulation of traffic between some strategic points of a city takes place on one-way roads as represented in Figure 2.21, and the numbers represent the rush hour travel time in minutes. We are looking for the shortest path (in the sense of minimum travel time) between x and y .

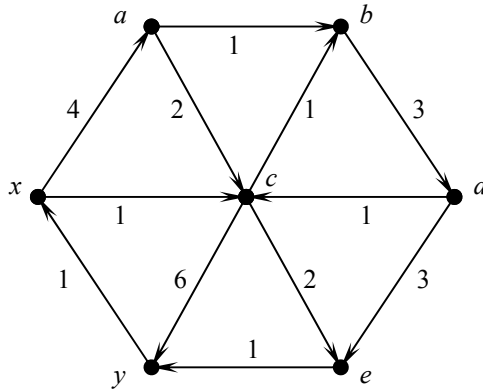


Figure 2.21. *The one-way sections with travel times*

Dijkstra's algorithm will label the vertices with some numbers (noted π) on any outstanding arcs in the set T . We suppose that at departure the vertex x definitely carries the mark zero ($\pi(x) = 0$) and all the other vertices temporarily carry the mark ∞ (infinity) (in practice a sufficiently large number) and that the set T is empty. The first iteration corrects the temporary marks on all the successors of x by the addition of the distance. Thus, in our example, the vertices a and c receive the temporary marks:

$$\pi(a) = \pi(x) + d(x, a) = 0 + 4 = 4 < \infty$$

$$\pi(c) = \pi(x) + d(x, c) = 0 + 1 = 1 < \infty$$

Then, we examine all the temporary marks and we definitely accept the temporary minimal mark: $\pi(c) = 1$ and we put in the set T the arc (x, c) , thanks to which the vertex c has received its mark. In general, this algorithm consists of the repetition of the two following operations:

- from the last accepted vertex we correct the temporary marks on all its successors, i.e. we add the distances carried by the arcs connecting the last accepted vertex with its successors; if the new values are inferior to the former we replace them;

– we examine all the temporary marks in order to accept the minimal temporary mark.

Thus, in our example, beginning with c , the vertices b , e and y receive the temporary marks:

$$\pi(b) = \pi(c) + d(c,b) = 1 + 1 = 2$$

$$\pi(f) = \pi(c) + d(c,f) = 1 + 2 = 3$$

$$\pi(y) = \pi(c) + d(c,y) = 1 + 6 = 7.$$

Then, we accept the minimal temporary mark $\pi(b) = 2$, and we put the arc (c, b) in the set T . We repeat the two operations, correction and acceptance, until all the vertices are marked. Figure 2.22 presents the final result.

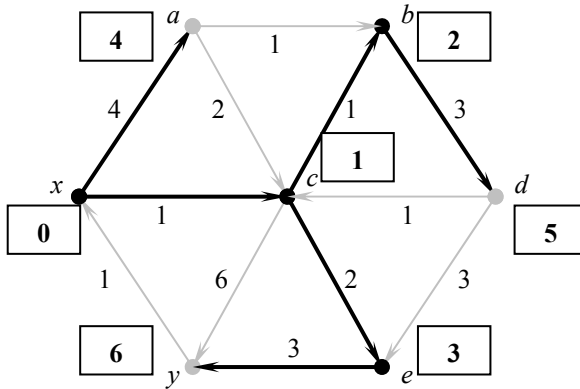


Figure 2.22. *The shortest paths from x to the other vertices*

The vertices have been definitely accepted by Dijkstra's algorithm in the order x, c, b, e, a, d, y , which corresponds to the increasing order of the marks: **0, 1, 2, 3, 4, 5, 6**. We notice that the response is more complete than the question asked: we have obtained the set of the shortest paths from x to the other vertices. We can also stop the algorithm when the final vertex of the sought path is definitely marked.

If we are interested in extremal circuits and paths, we obtain remarkable routings that are frequently encountered in numerous applications and widely studied in theory. In order to determine the optimal organizations of routes to pick up household refuse, we look for the circuit of minimum length that enable the route to

pass through each garbage site. In a graph, a path that passes only once through each vertex of the graph is called *Hamiltonian* (it consists of $n - 1$ arcs and n vertices). A *Hamiltonian circuit* is a circuit that passes through each vertex only once. Such a circuit consists of n arcs and n vertices. A graph is called Hamiltonian if it possesses a Hamiltonian circuit. In the non-oriented case we naturally obtain the concepts of chain, of cycle and of Hamiltonian graph. We owe the very first result in this matter to Sir William R. Hamilton (1805–1865), an Irish mathematician. The problem of the existence of Hamiltonian elements in an arbitrary graph is very difficult. Figure 2.23 represents the Petersen graph, which is the smallest graph that contains a Hamiltonian chain, but does not contain a Hamiltonian cycle.

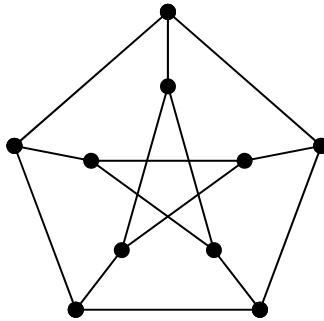


Figure 2.23. *Petersen graph*

The preceding concept is based on the constraint of passing only once through each vertex. Another idea is to cross all the edges (all the arcs) only once. This problem is given for a postman who must distribute the mail in the letterboxes that are found along the streets (modeled by the edges).

In a graph, a path that passes only once through each arc is called *Eulerian*. An *Eulerian circuit* is a circuit that passes only once through each arc. A graph is called Eulerian if it possesses a Eulerian circuit. In the non-oriented case we obtain the concepts of chain, of cycle and of Eulerian graph. These objects were studied by Leonhard Euler (1707–1783), a Swiss mathematician.

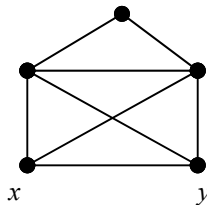


Figure 2.24. *Graph that admits a Eulerian chain between x and y*

A connected graph admits a Eulerian chain if and only if the number of vertices of odd degree is 0 or 2. Figure 2.24 represents a graph that is not Eulerian, but that admits a Eulerian chain between the vertices x and y .

In spite of the similarity of these two problems, it is necessary to know that the problem of the existence of a Eulerian element in a graph is easy, unlike the Hamiltonian problems.

2.6. Assignment and coloring

2.6.1. Matchings

To introduce the notion of matching, let us begin with an example.

EXAMPLE 2.3. [BER 73] During the Second World War, British flights were made with two-seater airplanes. To form a team, complementary competencies were necessary (piloting and firing for example) and certain pilots could not form a team in the same airplane because of differences in language or habits. How should the teams be organized in order to have the maximum number of airplanes?

In order to model this problem we can construct a graph whose vertices represent the pilots and whose edges represent the possibilities to put together two pilots in the same team. The teams that can be formed simultaneously constitute a set of edges that are two by two non-adjacent. Such a set of edges is called matching. If we are interested in the maximum number of planes that can be used simultaneously, we look for a *matching of maximum cardinality* (see Figure 2.26). We say that a vertex x is *saturated* by a matching M if there exists an edge of M attached to x . In the opposite case, we say that the vertex x is *unsaturated* by M . The matching is *perfect* if it saturates all the vertices in the graph.

Let us consider a matching M in a graph (the thick edges in Figure 2.25). The fine edges belong to a graph $G = (X; E)$, but do not belong to the matching chosen.

We call an *alternating chain* (relative to the matching M) a simple chain (i.e. not using the same edge twice) composed of the edges that are alternately in M and in $E - M$ (i.e. alternately in the thick and fine edges).



Figure 2.25. *An alternating chain*

An alternating chain joining two vertex unsaturated by M is called an *augmenting chain*: by interchanging the fine edges and the thick edges along this chain we obtain a new matching with a number of edges augmented by 1.

This method works efficiently in bipartite graphs.

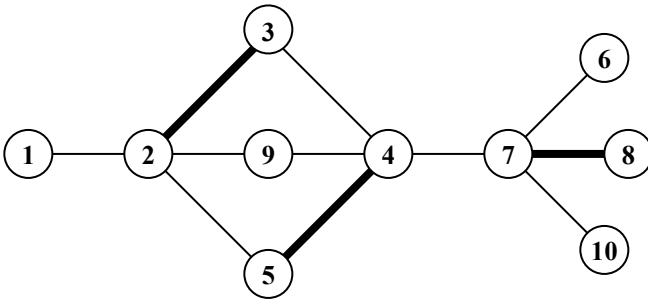


Figure 2.26. *Matching of maximum cardinality*

Let us suppose that the edges in the graph represented in Figure 2.26 indicate the possibilities of forming a team in the situation of the example in Figure 2.3. We have ten people but we cannot form five teams. The thick edges represent a maximum matching. Without detailing the algorithm that enables us to find it, we will only justify that the proposed matching is maximum: each edge of the graph is incident to at least one vertex of the set of three vertices $\{2, 4, 7\}$; thus no matching can have more than three elements.

We can generalize the method of alternate chains in arbitrary graphs, which are not necessarily bipartite. The following examples show some situations where we search for matchings with certain properties.

EXAMPLE 2.4. Let us suppose that we know the exact performances of $2n$ cyclists (for example, their best time over 20 km). We want to form n teams for the tandem race. We suppose that each couple will obtain the result that is the weakest of the

two. How should the teams be created so that the total time of the n tandems is the least possible?

EXAMPLE 2.5 [LOV 86] Let us suppose that a geological study enables us to localize numerous deposits of oil. Drilling from the surface is very expensive; we can envisage the exploitation of a second deposit beginning with an already explored neighboring deposit (see Figure 2.27). We can estimate the cost of each extension envisaged. We must determine the least expensive drilling plan.

Examples 2.4 and 2.5 have a common model. We must consider the complete graphs in the sets of vertices (the cyclists or the oil deposits). The edges will be valued by the weakest result of two cyclists in the first case and by the cost of drilling between two oil deposits in the second case. We are searching for a matching of maximum cardinality, of which the sum of the weights of all its edges is minimal – we are speaking about the matching of minimum weight.

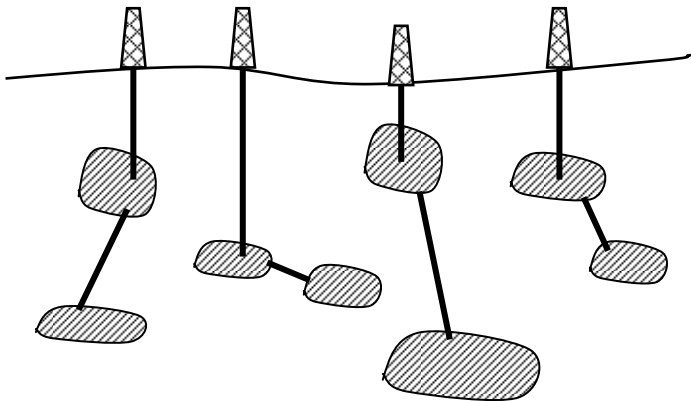


Figure 2.27. *Oil deposits*

Efficient algorithms also exist to search for a perfect matching of minimum weight (with valued edges) [EVA 92, LOV 86]. This problem can also be formulated as a linear program and thus becomes an example of the transportation problem (see Chapter 1).

2.6.2. *Vertex colorings*

To find the solution for certain problems, we search for a partition of the sets of vertices in an undirected graph into stable subsets. Figure 2.5 shows a graph whose

vertices have been partitioned into five stable subsets. It is convenient to identify this partitioning by the coloring of the vertices. We will first number the colors 1, 2, ..., k . We call a k -coloring an assignment of these k colors to the vertices, so that the two ends of each edge receive two different colors. This implies that the set of vertices of the same color is a stable set of G (subset not containing any edge). The graph in Figure 2.5 is therefore 5-colorable.

We generally try to find a coloring as economically as possible, that is, that minimizes the number of colors used. This minimum number of colors is called the *chromatic number* of the graph.

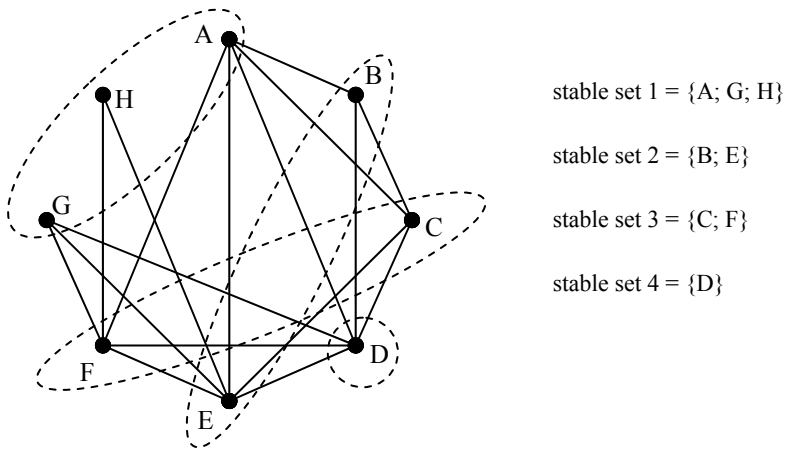


Figure 2.28. A 4-coloring of the graph in Figure 2.5

In Figure 2.28 we have represented a partition of the graph in Figure 2.5 in four stable sets. In this case we can easily prove that the chromatic number is 4 because the four vertices, labelled A, B, C and D generate as a subgraph, the complete graph K_4 , for which 4 colors are obviously necessary.

Unfortunately, it is not always possible to use this obvious certificate, as is shown in the example of the cycle of length 5, represented in Figure 2.29. Here the largest complete graph generated is K_2 and nevertheless the chromatic number is 3. The existence or not of this certificate has led to the definition of an interesting class of graphs called *perfect* [BER 73].

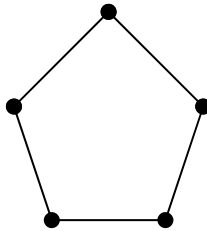


Figure 2.29. *Imperfect graph*

The determination of the chromatic number is, in general, a very difficult problem which has given rise to much of the leading research in graph theory. We can easily observe that this number is always less than the number of vertices in a graph since the assignment of a different color to each vertex is a coloring. Also, if K_p is a clique of G (set of vertices two by two adjacent), all colorings of G must use at least p colors in K_p (but eventually this number of colors is not sufficient to color the whole graph). This means that the chromatic number of a graph is situated between the number of vertices of its clique of maximum size and the total number of vertices.

The following examples show numerous situations where the coloring of the graph can be easily applied.

EXAMPLE 2.6. A chemical factory manufactures eight products A, B, C, D, E, F, G and H. The following table indicates that stocking certain chemicals together in the same storage area is prohibited because it is dangerous:

	A	B	C	D	E	F	G	H
A	no	yes	yes	yes	yes	yes	no	no
B		no	yes	yes	no	no	no	no
C			no	yes	yes	no	no	no
D				no	yes	yes	yes	no
E					no	yes	yes	yes
F						no	yes	yes
G							no	no
H								no

Table 2.2. *Incompatibilities (yes) of the chemicals*

It is necessary to determine the minimum number of storage areas the factory must have at its disposal.

EXAMPLE 2.7. A car rental company wants to satisfy eight requests in the following time intervals (start; finish): A(5h; 13h), B(6h; 9h), C(7h; 11h), D(8h; 15h), E(10h; 19h), F(12h; 20h), G(14h; 17h), H(18h; 21h). One vehicle can satisfy several requests as long as the time intervals do not overlap. We must find the minimum number of vehicles that the company must have.

EXAMPLE 2.8. Figure 2.30 represents the diagram of aerial traffic between certain airports. The planes fly extremely fast, but for security reasons it would be preferable to assign two different levels of cruising, to two lines whose trajectories cross each other. It is necessary to determine the minimum number of levels necessary.

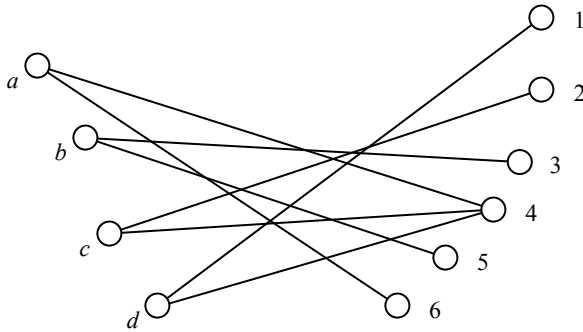


Figure 2.30. *Flight lines*

The situation of these three examples must first be modeled as graphs. Example 2.6 is the simplest. We consider the products as the vertices and an edge between two vertices represents the danger of stocking two products together. In this graph model, the products stocked in the same storage area must represent a stable set. The minimum number of storage areas is thus the chromatic number of the graph. Figure 2.28 gives an optimal solution to this problem.

In Example 2.7, it suffices to study the graph whose vertices are the requests (considered as intervals of time) and to connect two vertices by an edge if and only if the intersection of their intervals of time is not empty. Such a graph is called an *interval graph*. A vehicle can satisfy several requests if those requests form a stable set. We are finally in a situation identical to the preceding example (see Figure 2.28).

In Example 2.8, the notion of time disappears and it is necessary to consider the lines as vertices and to define an edge between two vertices if the two routes carried out at the same level have a point of intersection. We can assign without risk the same level to the routes that form a stable set. The minimum number of levels necessary for all the lines is, once again, the chromatic number of the graph. By assigning to the different flights the following labels: $A = c2$, $B = a4$, $C = b3$, $D = d1$, $E = a8$, $F = b5$, $G = c4$ and $H = d4$, we once again obtain the graph in Figure 2.28.

EXAMPLE 2.9 (inspired by [GOL 80]) The diagram in Figure 2.31 represents the organization of the traffic at an intersection with five roads.

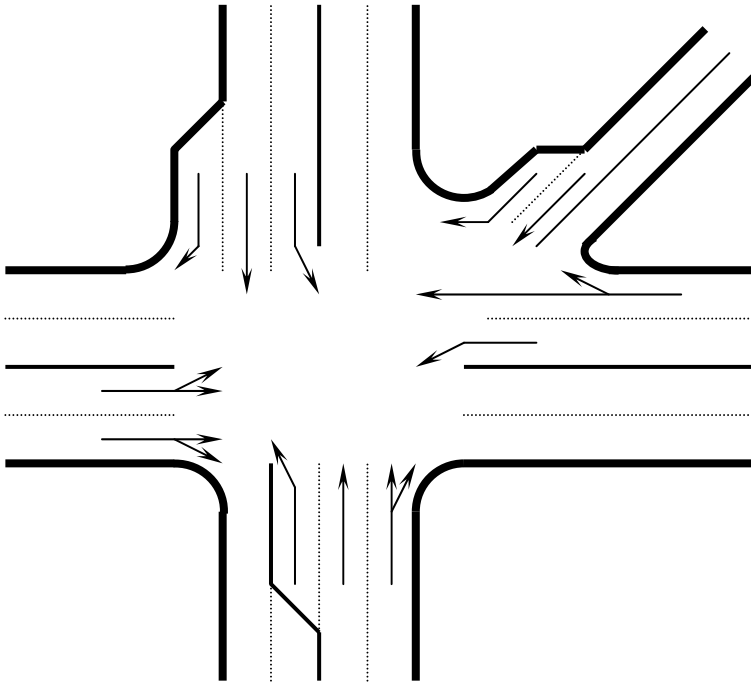


Figure 2.31. *Intersection with traffic light*

Numerous routes intersect and we envisage managing the traffic with a three-color traffic light (associated with each letter). Security requires that the green light be lit only on certain roads in order to allow the circulation of cars whose directions do not intersect. In a cycle of traffic lights it is thus necessary to foresee several

green light phases to move all the traffic. What is the number of green light phases at this intersection?

To answer this question it suffices to set up a graph whose vertices represent the roads where we envisage putting a traffic light, and the edges indicate a possible collision (see Figure 2.32). The partition of the set of vertices in 5 stable sets: $S_1 = \{a, b, c, d\}$; $S_2 = \{e\}$; $S_3 = \{f, g\}$; $S_4 = \{h, i, j\}$; $S_5 = \{k, l\}$ is minimal because the set $\{b, e, g, i, k\}$ generates the complete graph K_5 . This means that our intersection needs five traffic light phases.

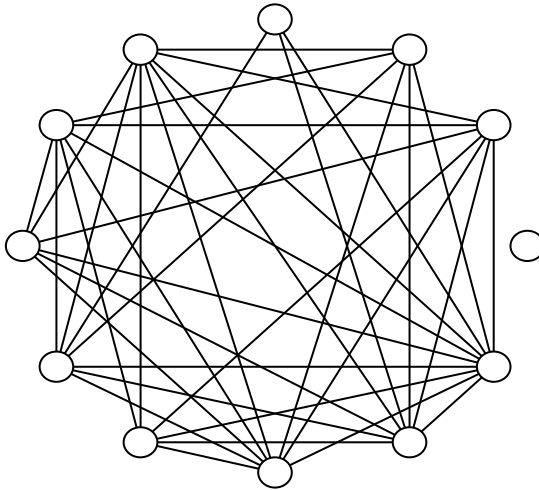


Figure 2.32. Conflict graph at an intersection

2.7. Flow in networks

To present an interesting model that can be applied in numerous real situations, we will begin with an example.

EXAMPLE 2.10. To deal with an exceptional demand, a travel agency wishes to send the maximum number of clients from Paris to Athens. There is no more space on the direct flight Paris-Athens, but we are studying flights with a stopover. Table 2.3 indicates all the seats now available in European airline companies. What is the maximum number of passengers that can fly from Paris to Athens under these conditions?

Flight	Seats available
Paris-Nice	6
Paris-Geneva	6
Nice-Rome	5
Geneva-Rome	4
Geneva-Vienna	3
Rome-Athens	7
Vienna-Athens	4

Table 2.3. Seats available on certain flights

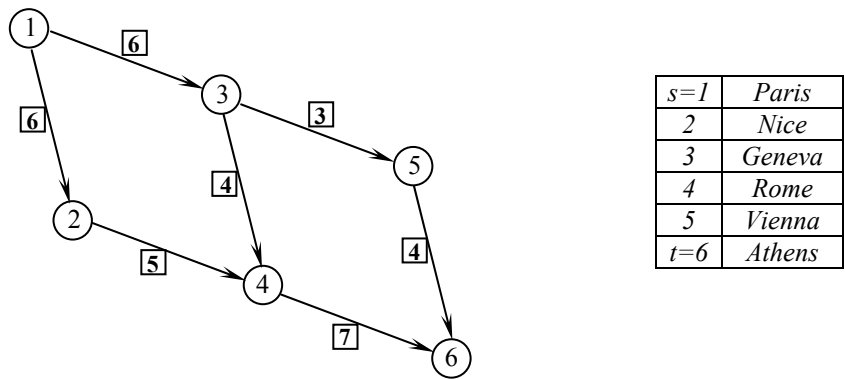


Figure 2.33. Flights with a stopover

In Figure 2.33 the simple graph, oriented with the arcs valued, regroups all the data of the problem. We note the existence of two particular vertices: 1 (the source) and 6 (the sink). The framed numbers that represent the tickets available on each flight are called the *capacities* of the arcs. A simple directed graph $G = (X;E)$ with two favored vertices: s (the source) and t (the sink) and the capacities of the arcs is a *network*.

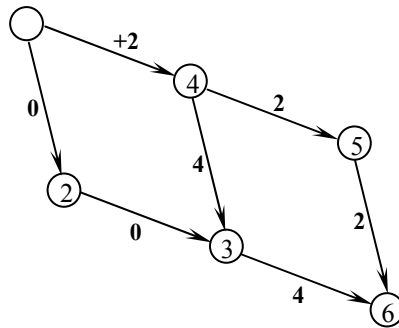


Figure 2.34. *Two first ticket assignments*

In this network each traveler must take a path from 1 to 6. In order to guarantee the stopover we must pay attention to the limited passenger capacity on the path chosen. It is clear that we cannot send more than four passengers in the direction $1 \rightarrow 3 \rightarrow 4 \rightarrow 6$. On each flight $(1, 3)$, $(3, 4)$ and $(4, 6)$, the agency can therefore reserve four tickets and the transfer (with stopover) of four passengers will be possible. The path $1 \rightarrow 3 \rightarrow 4 \rightarrow 6$ is no longer available because arc $(3, 4)$ is saturated. However, many other possibilities remain. To augment the number of passengers, we can put two more passengers on flight $1 \rightarrow 3 \rightarrow 5 \rightarrow 6$. In total it is now necessary to reserve six tickets on $(1, 3)$ (this arc becomes saturated), four on $(3, 4)$ and $(4, 6)$, and two on $(3, 5)$ and $(5, 6)$. The last path that remains available is $1 \rightarrow 2 \rightarrow 4 \rightarrow 6$ where we can send three more passengers. There is no longer a path augmenting the number of passengers. The respective quantities of tickets to reserve on each flight are represented in Figure 2.35.

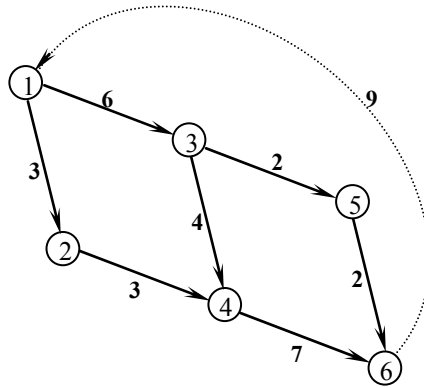


Figure 2.35. *A possible solution to the problem of flights with stopover*

The agency wants to know the number of tickets $f_{x,y}$ to reserve on each line (x, y) . The quantity $f_{x,y}$ carried by an arc $(x, y) \in E$ is called the *flow*. This flow must always be situated between zero and the capacity of the arc. If we want to calculate the total number of passengers that will travel on this network we can, for example, add the reserved tickets on two arcs leaving from 1: $3 + 6 = 9$. Also, we can make this addition on the arcs arriving in 6 ($7 + 2 = 9$). Figure 2.35 thus represents a solution (possible, but not optimal) that enables the transportation of a total of 9 passengers between Paris (the source) and Athens (the sink). By adding an arc, called the return arc, between Athens and Paris (see also Figure 2.35) the flow $f_{6,1}$ describes this total quantity.

To be able to continue the flights at each airport, the flow of arriving passengers must be equal to the flow of departing passengers. Thus we arrive at the principal notion of this chapter: we call *flow* in the network the set of the flow values $\{f_{x,y}\}$ verifying the two following properties:

- 1) for each arc $(x, y) \in E$, $0 \leq f_{xy} \leq c_{xy}$;
- 2) for each vertex x , $x \neq s$ et $x \neq p$, we have: $\sum_y f_{xy} = \sum_z f_{zx}$.

The first property demands that the flow on an arc is not negative and does not exceed the capacity of this arc. The second property, the flow conservation law in x , means that the total flow leaving from x is equal to the total flow entering in x . To simplify the notations, we consider here that the flow f is defined for all the pairs of vertices, setting the flow equal to zero on all the non-specified arcs.

The flow value f , that one notes $val(f)$, is the sum of the flows leaving from the source. It is also the sum of the flows entering the sink. In the problem of the maximum flow we are searching to maximize this value.

Let us mention that this problem of sending the maximum number of passengers can be naturally formulated as the following linear program:

$$\begin{aligned}
 \text{maximize:} \quad & z = val(f) = f_{6,1} \\
 \text{subject to:} \quad & 0 \leq f_{1,2} \leq 6, \\
 & 0 \leq f_{1,3} \leq 6, \quad f_{6,1} = f_{1,2} + f_{1,3} \quad (\text{depart from Paris}) \\
 & 0 \leq f_{2,4} \leq 5, \quad f_{1,2} = f_{2,4} \quad (\text{stopover in Nice}) \\
 & 0 \leq f_{3,4} \leq 4, \quad f_{1,3} = f_{3,4} + f_{3,5} \quad (\text{stopover in Geneva}) \\
 & 0 \leq f_{3,5} \leq 3, \quad f_{2,4} + f_{3,4} = f_{4,6} \quad (\text{stopover in Rome}) \\
 & 0 \leq f_{4,6} \leq 7, \quad f_{3,5} = f_{5,6} \quad (\text{stopover in Vienna}) \\
 & 0 \leq f_{5,6} \leq 4, \quad f_{4,6} + f_{5,6} = f_{6,1} \quad (\text{arrive in Athens}) \\
 & f_{i,j} \text{ integer}
 \end{aligned}$$

The problem can thus be solved by the general methods of linear programming presented in the preceding chapter. In the case of the flow, the simplex method furnishes by default an integer solution. However, numerous specific methods that are more efficient exist to solve the maximum flow problem. We have easily obtained an interesting solution by augmenting the flow on the paths between s and t . We will now explain how this idea can be refined to obtain an optimal solution.

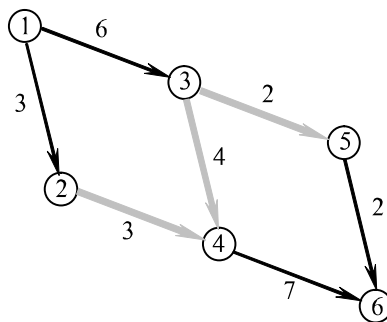


Figure 2.36. The passengers in a cut of the network

Remember that the total flow is calculated at departure (at the source) and on arrival (at the sink). This flow is also conserved during the flights. To verify it, we

must be capable of controlling all the paths between the source and the sink. For example, by removing from our network the three arcs $(2, 4)$, $(3, 4)$ and $(3, 5)$ we get a rupture between 1 and 6 because any path from 1 to 6 is strictly obliged to pass through one of these arcs. This means that we find on the three arcs all the passengers who are travelling from 1 to 6. In effect, we note that: $3 + 4 + 2 = 9$ (see Figure 2.36). It is convenient to introduce the notion of a *cut*, which separates the source s and the sink t . Such a cut is a set of arcs so that, if we remove them, the partial graph remaining no longer contains a path from the source to the sink. For example, the suppression of the three gray arcs in Figure 2.37 would have this effect. The cut partitions the set X of the vertices of the graph in two subsets S and $X - S$, so that $s \in S$ and $t \in X - S$ (in Figure 2.37 we note that $S = \{1, 2, 3\}$). A cut separating the source from the sink is naturally associated with this set S because this cut consists of all arcs of E leaving S and entering $X - S$. If no ambiguity results from this we will identify the cut and the set S with which it is associated.

We know the capacities of the arcs, so we can also calculate the *capacity* of a cut, denoted by $cap(S)$ and given by the formula:

$$cap(S) = \sum_{\substack{x \in S \\ y \notin S}} c_{xy}$$

The cut $\{(2, 4), (3, 4), (3, 5)\}$, associated with the set of vertices $\{1, 2, 3\}$, is illustrated in Figure 2.37. Its capacity is $5 + 4 + 3 = 12$.

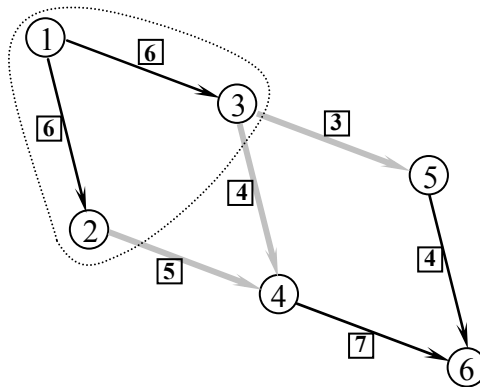


Figure 2.37. The cut associated with $\{1, 2, 3\}$ and its capacity

Since we find all the passengers who travel from s to t on any cut separating s and t , it is obvious that any flow value will always be less than or equal to the

capacity of any cut. In particular, the maximum flow is less than or equal to the minimum capacity of a cut. One of the most important results in combinatorics says that the maximum value of a flow in a network is always equal to the minimum capacity of a cut in this network. If, for a flow f and a cut S , we observe equality: $val(f) = cap(S)$, then the flow is of maximum value and the cut of minimum capacity. We can use this fact to confirm optimality.

The two problems relative to a network – to determine a flow of maximum value and to determine a cut of minimum capacity – are closely related. We present below an algorithm to find the maximum flow, to indicate a minimal cut and also to demonstrate formally the result cited above.

In Table 2.4 we enumerate all the cuts that separate 1 and 6 in the network of Figure 2.33.

Set S	Associated cut	Capacity	Set S	Associated cut	Capacity
1	$\{(1, 2); (1, 3)\}$	12	1, 3, 4	$\{(1, 2); (3, 5); (4, 6)\}$	16
1, 2	$\{(1, 3); (2, 4)\}$	11	1, 3, 5	$\{(1, 2); (3, 4); (5, 6)\}$	14
1, 3	$\{(1, 2); (3, 4); (3, 5)\}$	13	1, 4, 5	$\{(1, 2); (1, 3); (4, 6); (5, 6)\}$	23
1, 4	$\{(1, 2); (1, 3); (4, 6)\}$	19	1, 2, 3, 4	$\{(3, 5); (4, 6)\}$	10
1, 5	$\{(1, 2); (1, 3); (5, 6)\}$	16	1, 2, 3, 5	$\{(2, 4); (3, 4); (5, 6)\}$	13
1, 2, 3	$\{(2, 4); (3, 4); (3, 5)\}$	12	1, 2, 4, 5	$\{(1, 3); (4, 6); (5, 6)\}$	17
1, 2, 4	$\{(1, 3); (4, 6)\}$	13	1, 3, 4, 5	$\{(1, 2); (4, 6); (5, 6)\}$	17
1, 2, 5	$\{(1, 3); (2, 4); (5, 6)\}$	15	1, 2, 3, 4, 5	$\{(4, 6); (5, 6)\}$	11

Table 2.4. Set of all the cuts that separate 1 and 6 in the network of Figure 2.33

The number of cuts is exponential (in our case, from four intermediate vertices $\{2, 3, 4, 5\}$ we can create $2^4 = 16$ subsets) and in general we will not be able to search for the cut of minimum capacity by enumeration. The following example shows that we are sometimes directly interested in the minimum cut in a network.

EXAMPLE 2.11. Let us assume that the graph in Figure 2.33 illustrates a system of canals planned for the evacuation of rainwater from the area $s = 1$ toward the area $p = 6$. The number on each arc represents the capacity of evacuation in m^3/s . We must determine the cuts where we must anticipate in priority an augmentation of the capacity of evacuation in case of an exceptional rising of the water.

Table 2.4 indicates that the cut $\{(3, 5); (4, 6)\}$, associated with $\{1, 2, 3, 4\}$, has a minimum capacity of 10 ($= 3 + 7$).

REMARK. Formally, the value of a flow f is defined by:

$$val(f) = \sum_{\substack{x \in S \\ y \notin S}} f_{xy} - \sum_{\substack{x \in S \\ z \notin S}} f_{zx}$$

where S is a cut separating the source from the sink because it is sometimes necessary to take into account the entering arcs in a cut. In Figure 2.36 for example, in order to find the 9 travellers in the cut $\{1, 2, 4\}$ we must subtract: $6 + 7 - 4$, i.e. it is also necessary to subtract the entering flow.

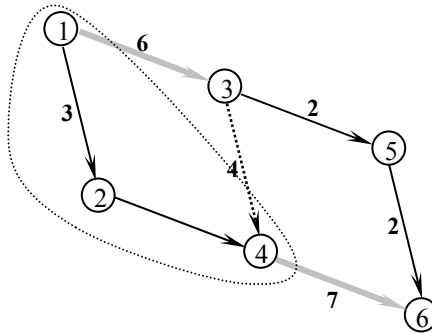


Figure 2.38. The cut associated with $\{1, 2, 4\}$

Here is a method, historically the first, to solve the problems stated. Let us suppose that we have already determined a flow f (may be not yet optimal) in a network. We are next interested in chains $\Gamma = \{s = x_0, e_1, x_1, e_2, \dots, x_{k-1}, e_k, x_k\}$, that is, alternate sequences of vertices and arcs so that, for every i , the arc e_i has x_{i-1} and x_i as extremities (without specifying which is tail or end, as in the undirected case). Such a chain is called *augmenting* to x_k , if its forward arcs are not saturated and its reverse arcs carry a flow that is not zero. In other words, for each arc e_i we have:

$$f(e_i) < c(e_i) \text{ if } e_i = (x_{i-1}, x_i) \text{ and } f(e_i) > 0 \text{ if } e_i = (x_i, x_{i-1})$$

If $x_k = t$ (the sink), Γ is called simply an *augmenting chain*.

Let us designate by Γ^+ (Γ^-) the set of forward (reverse) arcs of an augmenting chain Γ . Let us consider a new flow defined by:

$$f(e_i) := \begin{cases} f(e_i) + \delta & \text{if } e_i \in \Gamma^+ \\ f(e_i) - \delta & \text{if } e_i \in \Gamma^- \\ f(e_i) & \text{if } e_i \notin \Gamma \end{cases}$$

where $\delta = \min \{ \min_{e_i \in \Gamma^+} [c(e_i) - f(e_i)]; \min_{e_i \in \Gamma^-} f(e_i) \}$.

This new flow is realizable (the law of conservation and the capacities of the arcs are respected) and the value of the flow thus augments by δ .

The augmenting chain method, an algorithm attributed to Ford and Fulkerson, consists of the repetition of the two following operations:

- from an existing flow (initial flow is zero everywhere) search in G for an augmenting chain Γ and determine its value of augmentation δ ,
- change the flow according to the given formula.

In practice, we realize some direct or indirect labeling procedures.

The existence of an augmenting chain clearly implies that the flow f is not of maximum value. Since the reverse is then also true, either the flow f obtained after several iterations is of maximum value, or an augmenting chain exists and we can reiterate the algorithm. It is still necessary to ensure that this algorithm stops, but we will not enter into these theoretical conditions here.

If the network with the flow f contains no augmenting chain (to the sink), the set of vertices x , for which there exists an augmenting chain to x , gives a cut of minimum capacity.

We can now finish our numerical example. From the flow in Figure 2.35 we note that there is no longer an augmenting chain between 1 and 6. In effect, if we take away two arcs from this network, those of capacity 6 and 7, saturated by the present flow, no path between 1 and 6 exists. The augmenting paths arrive at 2 and 4. The cut associated with $S = \{1, 2, 4\}$ is of capacity 13. The flow obtained only has the value 9, which means that this flow is poorly distributed in this network because we have a capacity sufficient to leave the cut that is now blocking us. In order to allow a better distribution we find an augmenting chain $\Gamma = \{1 \rightarrow 2 \rightarrow 4 \leftarrow 3 \rightarrow 5 \rightarrow 6\}$. The change with $\delta = 1$ results in a flow of value 10, as illustrated in Figure 2.39.

When we try to carry out another iteration of the algorithm we notice that beginning with vertex 1 there exist augmenting chains to 2, 4 and 3. We easily verify that the cut associated with the set $S = \{1, 2, 3, 4\}$ attains the value 10 (see also Table 2.5), which also confirms that the flow is maximum and the cut minimum.

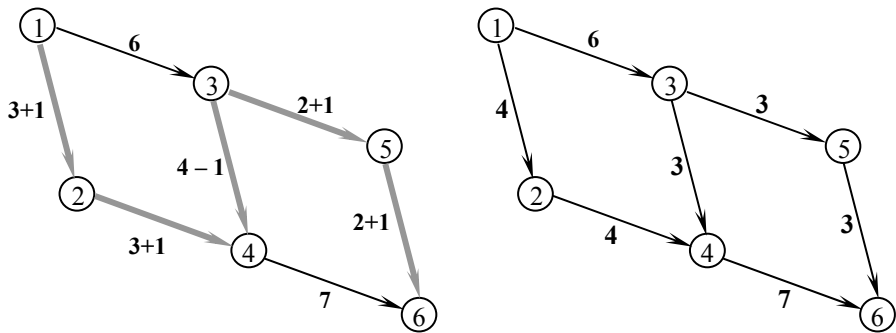


Figure 2.39. An augmenting chain and the resulting flow

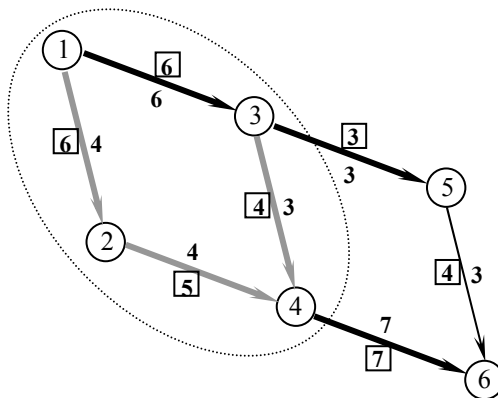


Figure 2.40. The maximum flow and the minimum cut

The total flow can always be decomposed into several flows on paths between the source and the sink (in the example presented, the path $1 \rightarrow 3 \rightarrow 5 \rightarrow 6$ and the path $1 \rightarrow 3 \rightarrow 4 \rightarrow 6$ each transport the flow of value 3; the path $1 \rightarrow 2 \rightarrow 4 \rightarrow 6$ carries a flow of value 4). However, *a priori*, it is difficult to choose the right paths. The chains enable possible corrections.

Here are some examples of the application of the flow model:

EXAMPLE 2.12. The mail is currently transported from a source S toward its destination D by train (T), plane (A) and truck (C), passing by an intermediate city (service stop) according to Figure 2.41 where, on each arc, we have indicated the capacity of the means of transport (the framed numbers).

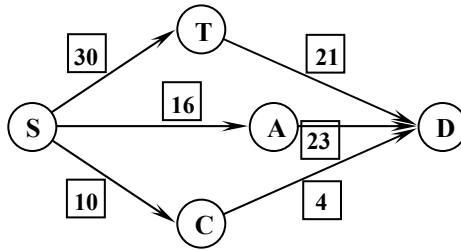


Figure 2.41. Capacities of mail transport

The flow model enables us to find the maximum volume of mail that can be sent from S to D, as well as its distribution among the different means of transportation. There are three augmenting paths and we easily arrive at a value of 41. This limit is justified by the cut of minimum capacity.

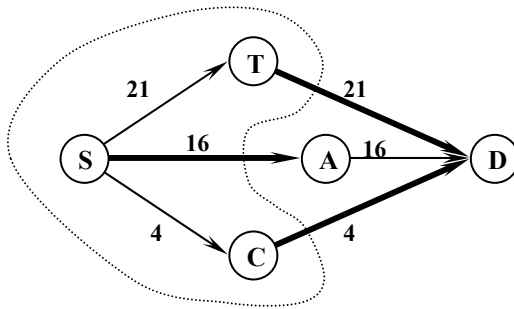


Figure 2.42. Maximum volume of the mail

To augment the volume of mail, we install in the intermediate city a transfer center where the mail can be redistributed among the different means of transport with the following capacities: the transfers from A toward T, from T toward A and from C toward A are each of capacity 3; the transfers from A toward C and from T toward C are each of capacity 4 and the transfer from C toward T is impossible.

To find the maximum volume of mail that can be sent in these conditions, we can remark that only the transfers that are represented by the arcs leaving the cut are interesting. This concerns TA and CA.

The new flow of value 47 is maximum, which is confirmed by the cut of the same capacity.

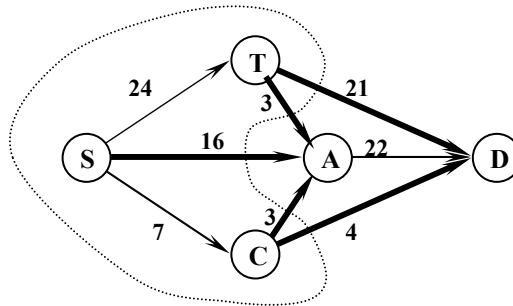


Figure 2.43. *Augmentation of the volume of mail due to transfer*

The maximum flow problem finds a rather unexpected application where one wants to determine the optimal exploitation plan for an open-air pit of a mineral deposit. The following example is inspired by a result of J.-C. Picard [PIC 76].

EXAMPLE 2.13. The mineral deposit is partitioned in thousands of relatively small blocks (cubes of 10 m sideways, to give an idea). Through the rate of mineralization we can estimate the commercial value of the ore in each of these elementary blocks. The open-air pit exploitation obviously supposes that a block cannot be extracted unless the part above has been removed beforehand. In addition, security constraints impose a maximum angle (generally 45°) in front of the advancement of the quarry walls to eliminate the risk of collapse.

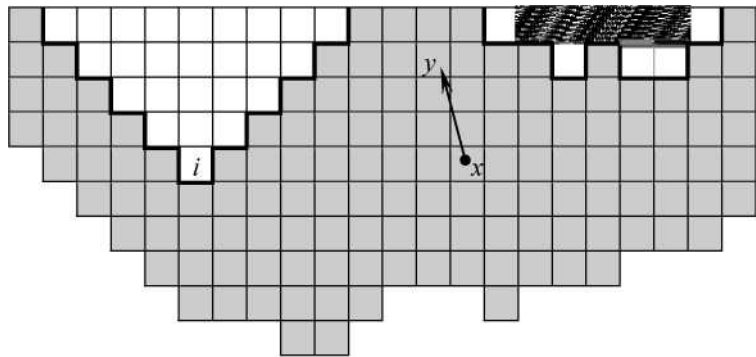


Figure 2.44. *Open-air mineral deposit*

We suppose as known the cost of extraction of each block number i (not cumulated with the cost of extraction of the part above). This determines the net

profit p_i of the extraction of the block i . By considering a pit P as a set of cubes, we obtain the total net profit of the creation of this pit equal to $\sum_{i \in P} p_i$. It is clear that

every set does not have to constitute a realizable pit, because in order to extract an element we must also extract all the cubes forming a vertical cone pointing to this element.

To enable the reader to better understand the model, we have presented in Figure 2.44 the situation in two dimensions (width and depth); the generalization in three dimensions is not a problem. We consider the oriented graph $G = (X; E)$ where the vertices X are all elementary cubes and where we put an arc (x, y) between the two cubes x and y when it is necessary to extract y before x . The subset P of the vertices that form a realizable pit are characterized by the following property: if a vertex x belongs to the pit P and if there exists an arc (x, y) , then y also belongs to the pit P .

We can associate with each vertex x of the graph G its net profit from extraction p_x . The problem consists of finding a realizable pit P that maximizes $\sum_{x \in P} p_x$. Here is a clever transformation of the problem of minimum cut. We can first partition the set of vertices X in two parts A and B , where $A = \{x \in X \mid p_x \geq 0\}$ and $B = \{x \in X \mid p_x < 0\}$.

We will construct a new graph $G' = (X'; E')$ by adding two particular vertices: s (a source) and p (an abstract sink) and the supplementary arcs (s, a) for all vertices $a \in A$ and (b, p) for all vertices $b \in B$. Let us assign to each new arc (s, a) the capacity $c_{sa} = p_a$ and the capacity $c_{bp} = (-p_b)$ to each arc (b, p) . The capacities of the original arcs of $G = (X; E)$ are infinite.

It is very easy to see that P is a realizable pit if, and only if, the capacity of the cut $P \cup \{s\}$ is finite, because an infinite capacity for an arc (x, y) departing from this cut ($x \in P$ and $y \notin P$) means that we must extract y before x and $y \notin P$. We can thus calculate the capacity of the pit $P \cup \{s\}$:

$$\sum_{a \in A \cap P} c_{sa} + \sum_{b \in B \cap P} c_{bp} = \sum_{a \in A \cap P} p_a + \sum_{b \in B \cap P} (-p_b) = \sum_{x \in A} p_x - \sum_{x \in P} p_x = \text{const.} - \sum_{x \in P} p_x$$

The maximizing of $\sum_{x \in P} p_x$ is thus equivalent to the minimization of the capacity of a cut. We can conclude that the minimum cut in the graph $G' = (X'; E')$ determines the optimal realizable pit in our problem. This categorically transforms the fact that in reality we are trying to find an ideal equilibrium, because the extraction of blocks rich in ore implies costly preliminary extractions.

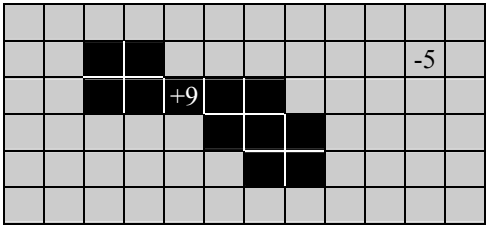


Figure 2.45. *Commercial values of ore*

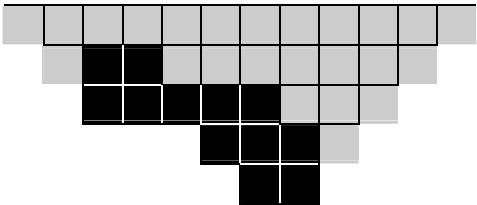


Figure 2.46. *Vertices of the future graph*

Let us study the numerical example illustrated in Figure 2.45. The commercial value of the ore in each elementary gray block is -5 and in each elementary black block $+9$.

Before modeling as a graph, we can limit ourselves to the interesting blocks and those whose extraction is required by the constraints of security (a maximum angle of 45°).

The graph model is presented in Figure 2.47. In order not to overload the drawing, certain obvious arcs have been purposely left out (if it is necessary to extract y before x and also z before y , that obviously implies that one must extract z before x).

We leave to the reader the determination of the maximum flow; its value is 105. The arcs of the minimum cut are illustrated in Figure 2.48.

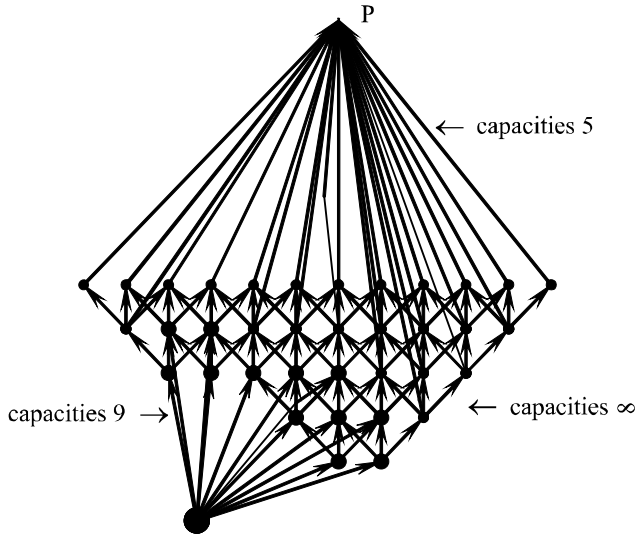


Figure 2.47. Modeling the ore deposit as a graph

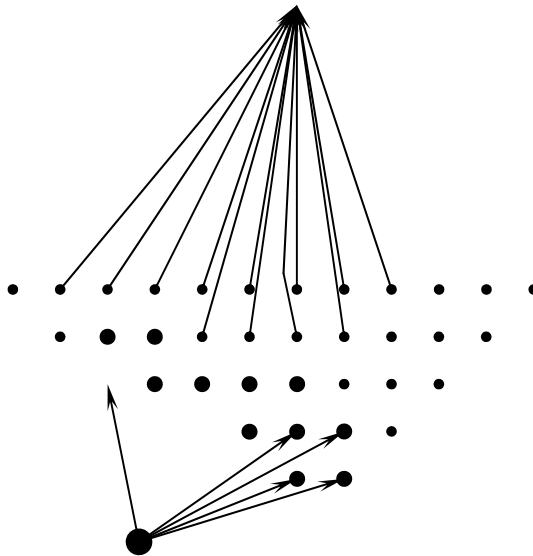


Figure 2.48. The cut of minimum capacity

The terminal vertices of the arcs leaving from s indicate the elementary blocks s rich in ore that must not be extracted. The initial vertices of the arcs entering p indicate the elementary blocks whose extraction is costly but necessary to attain certain rich blocks. We present in Figure 2.49 the profile of the corresponding mineral deposit (in gray dots).

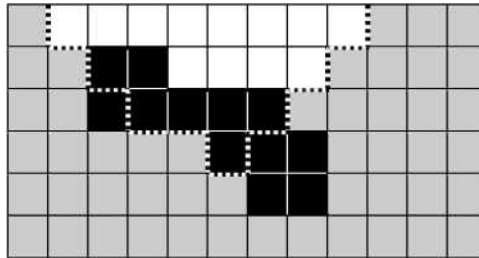


Figure 2.49. *Optimal mineral deposit*

2.8. Conclusion

In this chapter we have presented the fundamental concept of a graph. This abstract object has been introduced using real situations, and we have illustrated its utility in numerous examples. We have shown that a graph formalizes complicated problems in their realistic description, and that its simple structure enables us to systematically deduce an algorithmic solution. The concepts presented are the basis of the subjects treated in the other chapters of this work. As a more important example we have illustrated how research becomes operational: a study of the maximum flow and of the minimum cut, which is of great theoretical utility, also finds an unexpected application in the realization of an open-air pit of mineral deposit.

2.9. Bibliography

- [BER 73] C. Berge, *Graphes et hypergraphes*, 2nd edn, Dunod, 1973.
- [EVA 92] J.R. Evans and E. Minieka, *Optimization Algorithms for Networks and Graphs*, Dekker, 1992.
- [GOL 80] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, 1980.
- [GOU 88] R. Gould, *Graph Theory*, The Benjamin/Cummings Publishing Company, 1988.
- [LOV 86] L. Lovasz and M.D. Plummer, "Matching theory", *Annals of Discrete Mathematics*, vol. 29, Elsevier Science Publishers, 1986.

- [PIC 76] J.-C. Picard, “Maximal closure of a graph and applications to combinatorial problems”, *Management Science*, vol. 22, 1268–1272, 1976.
- [SAK 84] M. Sakarovitch, *Optimisation combinatoire*, Hermann, 1984.
- [XUO 92] N.H. Xuong, *Mathématiques discrètes et informatique*, Masson, 1992.

Chapter 3

Classical Combinatorial Problems and Solution Techniques

3.1. Introduction

The objective of this chapter is firstly to present the classical combinatorial problems used for modeling concrete situations. These problems will be described in different paragraphs, and regrouped according to common characteristics. Secondly (section 3.3), we will present some elements of the theory of complexity. This theory allows us to classify optimization problems as easy problems and difficult problems according to the level of difficulty of their solution. To conclude, some solution approaches of difficult problems will be presented (section 3.4). A section will be dedicated to the methods of relaxation and followed by a description of methods of approximate solutions and optimal solutions. The methods of approximate solutions, also called heuristics, allow us to obtain a realizable solution (that is, which respects the constraints) and for which the value of the performance criterion is not too bad. The optimal solution methods, also called exact methods, allow us to obtain better solutions by means of an execution time that is often elevated. In order to illustrate our proposal, we will use the so-called *traveling salesman problem*, a problem that will be defined in the first part of this chapter.

3.2. Classical optimization problems

3.2.1. Introduction

The combinatorial optimization problems addressed in the literature most of the time come from concrete situations in industry or services. They represent a simplified view of a situation and their solution helps us to make decisions.

Most often, an optimization problem presents itself as an objective to pursue (maximization of a profit, minimization of a distance or of a cost, etc.). This objective can be mono- or multicriteria. In this chapter, we will stress and speak about monocriteria problems; some elements on multicriteria optimization can be found in [ROY 93]. Possible solutions to these problems must respect a set of constraints linked to the situation studied. One of the basic techniques for the formalization of these problems is linear programming, as we have seen in Chapter 1.

In the following discussion, we will present the classical combinatorial problems by trying to class them according to their common characteristics. We will address, in order, problems of combinatorial optimization in graph theory, problems of assignment, transportation, location and scheduling. Most of the problems presented here will be addressed in more detail in other chapters; that is why we will limit ourselves to a summary description. We explain for each problem described whether it is easy or difficult to solve in general. This judgement about the difficulty of the problems is directly connected to notions detailed in section 3.3, which is dedicated to the theory of *complexity* [GAR 79].

3.2.2. Combinatorial optimization problems in graph theory

Graphs are a powerful modeling tool and are very intuitive. They are of course used to represent transportation or communication networks, and more generally the flow of materials or information. They are also used to model problems in which *objects* are in relation, the vertices of the graph representing these objects and the edges (or arc if an orientation is considered) the relations between these objects [DIE 00].

In order to solve an optimization problem using a graph representation, it is often necessary to search for a particular element in the graph modeling the situation. The classical problems have already been presented in Chapter 2. We will refer to them

briefly and add extensions; some of them will be detailed in the following chapters. The classical problems are:

- the minimum spanning tree problem (easy problem);
- the maximum cardinality matching problem (easy problem). In the case of graphs for which weights are associated with the edges, the most well-known variants are the search for a maximal matching of maximum weight (easy problem) and the search for a maximal matching of minimum weight (difficult problem);
- the search for the shortest path (easy problem). While the problem of the search for the shortest path can be solved by efficient algorithms, the problem of finding the longest path is a difficult one;
- the coloring of graphs (difficult problem). These problems are frequently used in telecommunications, for the assignment of cell phone frequencies for example (see Chapter 8);
- the maximum flow problem (easy problem). In a classical extension, a cost is associated with each arc and the goal is then to find a maximum flow of minimum cost (easy problem, by linear programming for example).

There are many other problems defined on graphs; let us mention one last model, the *partitioning of graphs*. The problem of partitioning graphs [RAD 97] consists of partitioning the set of vertices in subsets so that each subset possesses a certain property. For example, we can search for a partition of the vertices so that each induced subgraph is a complete graph, that is, all the vertices of the graph obtained are directly connected by an edge.

3.2.3. Assignment problems

In *assignment problems*, the objective is to assign the objects to places able to contain them. In this section, we regroup different problems that all have this point in common.

The assignment problem, strictly speaking [AHU 93], as it is described in the literature, consists of associating each object in a set with an object in a second set. Without loss of generality, we can consider that each set contains the same number of objects, and then speak of the assignment of an object to a place. The assignment of a given object to a given place involves a cost. The objective is to realize the assignment (each object has a place and there is a place for each object) while minimizing the sum of the costs. This problem can be solved easily.

The so-called *generalized assignment problem* [AHU 93] is difficult to solve. It consists of assigning a set of objects to a set of places, but this time, one place can

accommodate several objects. However, each place cannot accommodate all the objects because there are limitations, for example, of volume.

The *knapsack problem* [MAR 90] is one of the classics of combinatorial optimization. In this problem, a weight and a value are associated with each object. The objective is to fill the knapsack, which is of limited capacity, in order to maximize the value of its contents. This problem is difficult to solve. Some variants of this problem exist, for which several constraints must be taken into account, such as weight and size, for example.

The so-called *bin packing problem* is also a classical problem [RAD 97]. Each object is characterized by a certain volume. We must put all the objects in containers that may or may not be identical, and are of finite capacity. The objective is to put all the objects into a minimum number of containers. In the simplest version, this problem is already difficult to solve. Numerous variants of this problem exist, such as the one which involves two characteristics for each object (for example, length and width).

3.2.4. Transportation problems

In this section, we have regrouped some classical problems, most often modeling problems of transportation such as the assignment of round-trips (tours), and models used to represent road transportation, for example.

We will now describe the most famous problem of combinatorial optimization: the *traveling salesman problem*. This problem consists of searching for a route of minimum length passing through the cities that a traveling salesman must visit [LAW 85]. More formally, it is defined as a theoretical graph problem. Let us consider a graph whose vertices represent the cities to visit and the edges represent the connections among these cities. The weight associated with an edge represents, for example, the distance between the two cities at the extremities of the edge. The objective is to find a Hamiltonian cycle (a cycle passing once and only once through all the vertices of the graph) of minimum length. This problem is difficult to solve. Numerous variants of this problem exist (particular properties of the graph, particular weights on the edges, time windows to visit the cities, etc.). The same problem exists equally in oriented graphs. In this case, the routes of the traveling salesman must respect the orientations of the arcs.

An extension of the traveling salesman problem is the *vehicle routing problem* [GOL 88]. A fleet of vehicles of finite capacity, based in a depot, must assure a set of deliveries. Each city must be visited by a vehicle and the tours (set of the cities visited) by this vehicle must not exceed its capacity. The objective is to minimize

the total length of the trips traveled by the fleet of vehicles. This problem in fact contains an assignment of vehicles to trips and routing problems as in the traveling salesman problem. Some variants of the problem are minimizing the number of vehicles used or also considering time windows for visiting the cities.

The *Chinese postman problem* [COO 97] consists of realizing a tour passing through all the edges of a graph at least once, with the objective of minimizing the total length of the tours. This problem models, for example, the tours to pick up household refuse or to distribute the mail. If the graph only contains edges (the vehicle can go from a city a toward city b or reciprocally), the problem is easy to solve. It remains so when all the connections between cities are arcs (orientation of the connections). Unfortunately, the problem becomes difficult to solve when there are oriented and non-oriented connections in the graph. It becomes equally difficult when the weights on the arcs must be taken into account, weights that can represent capacities, for example.

The *transportation problem*, strictly speaking [RAD 97] (see example in Chapter 1, section 1.3), can be seen as the determination of the transfer of products from a set of storage areas toward a set of clients. Each storage area contains a certain quantity of product (volume for example) and each client asks for a certain quantity. It is always possible to go back to a problem for which the sum of the quantities in the storage areas is equal to the sum of the quantities demanded. The transfer of a product unit in stock to a client entails a cost. The objective is to minimize the sum of the transportation costs. This problem can easily be solved by linear programming (see Chapter 1) or by specific methods. The transshipment problem is an extension of the transportation problem. Now in this problem the storage areas (offers) and the clients (demands) are situated on the vertices of a graph, the edges being weighted by a shipment cost of a product unit. A simple transformation makes it possible to go back to the classical transportation problem and thus to solve this problem easily.

3.2.5. Location problems

A *location problem* [DRE 96] consists of placing a set of services (storage areas, factories, firewalls, etc.) in a given territory, with the objective of minimizing a cost function.

In the *p-center problem*, a fixed number of services have to be placed on the vertices or the edges of a graph. If a service is placed on an edge, the distance between this service and each adjacent vertex is calculated according to where the service is placed on the edge. If the service is placed in the middle of the edge, the distance to each vertex will be half the length of this edge. The objective is to minimize the longest distance between the vertices and the nearest service. In other

words, for each vertex in the graph, the minimum distance from this vertex to the nearest service must be calculated. This brings us back to the problem of the shortest path, described above. Then, it is necessary to determine which vertex in the graph is the most distant from its nearest service. This distance has to be minimized.

The *p-median problem* is similar; only the objective function changes. The objective is to minimize the sum of the distances between each vertex and the nearest service.

These two problems are difficult in their general formulation, and only some particular cases can be solved easily. Numerous variants exist, considering for example the costs of installation of services on each vertex or edge, and also including the capacities for the services, that is, where a limited number of vertices are able to benefit from the same service.

3.2.6. *Scheduling problems*

The *scheduling* theory regroups problems for which the utilization of resources must be planned in time. We will give the main types of problems encountered.

Project scheduling consists of planning in time the execution of the different tasks which constitute the project. Chapter 4 is dedicated to these problems.

The production enterprises must sequence the operations taking place in their workshops [BLA 96]. According to the type of products manufactured and the process used, these problems can be quite different. The simplest problems to formalize are the problems on a single machine. In these, a known number of tasks must be sequenced (a sequence must be fixed), the machine being able to execute only one task at a time, with the goal of optimizing an objective function (minimize a maximal lateness for example). In the problems of parallel machines, two types of problems must be solved: the assignment of the tasks to the machines, and the determination of the sequences of tasks on each of the machines. There exist numerous problems called workshop problems, as they propose a simplified workshop model of production. We can cite the problems of *flowshop*, *jobshop* or even *openshop*. In the flowshop, or workshop in line, all the products pass on the machines in the same order. In the jobshop, each product follows its own course in the workshop. Finally, in the openshop, the products can pass on the machines in any order.

The problems of scheduling tours can be seen as traveling salesman problems with time windows or vehicle routing problems with time windows. These problems have already been described above and will be detailed in Chapter 5.

Team planning of work is also a part of scheduling problems [RAD 97]. It consists of the constitution of teams, and of the assignment in time of these teams with different tasks to execute. These problems are very often encountered in the organization of work for medical teams in hospitals or the management of crews for airline companies. Most of these problems are difficult to solve optimally.

The theory of scheduling also includes the problems of generating *timetables* [RAD 97]. This somewhat joins the problems of team scheduling described previously. The simplest example, but difficult to solve, is the creation of timetables in a university, where it is necessary to take into account the groups of students, the teachers, the classrooms and the priorities among class sessions.

3.3. Complexity

We have just described several classical optimization problems as well as a classification of these problems according to their domains of application. In this section we will see that it is also possible to classify the problems according to their *complexity*: easy or difficult problems. We will then define the terms more precisely: easy and difficult. The interest of such a classification is to determine if, for a given problem, an exact solution can be found in a reasonable time. In other words, such a classification tells us if the search for an approximate solution of good quality is interesting or not. In order to make such a classification of optimization problems, the theory of complexity is used. We will here make only a brief remark. Several works are dedicated in whole or in part to this theory; we can notably cite the work of Garey and Johnson [GAR 79]. This work proposes, among other things, a list of classical combinatorial problems known to be difficult.

To determine the difficulty of an optimization problem, one must know a little about *algorithmic complexity*. Also, we will first describe what we mean by an efficient algorithm. The complexity of an algorithm is measured by the number of elementary operations necessary in the worst case; that is, when the problem to solve is such that a maximum number of elementary operations is necessary for its solution. In other words, to say of an algorithm that it is in $O(n)$ means that in the worst case a multiple of n elementary operations must be carried out. An algorithm is called *polynomial* if its complexity can be expressed as a polynomial of the given characteristic input data. Such an algorithm is called rapid (or also efficient). An algorithm will be called *pseudo-polynomial* if its complexity can be expressed as a polynomial of the problem parameters. For example, let us consider the traveling salesman problem. An algorithm whose complexity can be expressed as a polynomial of the number of cities will be called polynomial. However, an algorithm whose complexity is expressed as a polynomial of the number of cities and of the greatest distance separating two cities will be called pseudo-polynomial.

In addition, an algorithm whose complexity cannot be expressed in the form of a polynomial will be called *exponential*.

The interest in making such a distinction among the different existing algorithms, or those able to be developed, resides in the processing time. Let us consider a traveling salesman problem with $n = 5$ cities and let us suppose that an elementary operation demands a microsecond (ms) to be executed. If, in order to find a solution to this problem, a polynomial algorithm of complexity $O(n)$ is used, then the processing time will be a multiple of 5 ms. However, if we use an exponential algorithm of complexity $O(n!)$, the processing time will be a multiple of $5!$ or a multiple of 120 ms. Let us now consider a traveling salesman problem with 50 cities. The polynomial algorithm needs a multiple of 50 ms for its execution while the exponential algorithm has a processing time corresponding to a multiple of $50!$ or a multiple of $3 \cdot 10^{64}$ microseconds or about 10^{51} years.

Having defined the notion of an efficient algorithm (polynomial algorithm), we will now look into the classification of optimization problems in easy and difficult problems.

To study the complexity of an optimization problem, it is necessary to define the corresponding *decision problem*. A decision problem is a statement for which the response must be expressed by yes or no. For example, for a traveling salesman problem, the corresponding decision problem $P1$ is: "If y is an integer, does there exist a Hamiltonian cycle whose length is less than or equal to y ?"

Decision problems are divided into two classes: *decidable* and *undecidable problems*. A problem is called undecidable if it is impossible to write an algorithm to solve it.

A decidable decision problem belongs to the *class NP*, if there exists a polynomial algorithm that enables us to recognize a positive instance (example for which the response is yes). For example, when a tour is defined, it is possible to find a polynomial algorithm that verifies its length in order to know if it allows us to reply yes to the corresponding decision problem. The decision problem corresponding to the traveling salesman problem thus belongs to the class NP.

The problems of the class NP can be divided into two different groups, on one hand the problems called *polynomial* (problems of *class P*) and, on the other hand, the problems called *NP-complete*.

A decision problem is called polynomial if there is a polynomial algorithm to solve it. Such a problem is called *easy*. For example, the problem "is there a path between city A and city B with a length less than or equal to 10 km?" can be solved

in $O(n^2)$ where n is the total number of cities to consider. This problem can be solved in polynomial time so this problem is a problem of the class P , and is therefore called easy. By extension, the optimization problem that consists of searching for the shortest path between A and B is called polynomial.

The NP-complete problems are *difficult* problems. For each of these NP-complete problems, there is little likelihood that a polynomial algorithm could be constructed. The principal notion in order to define NP-completeness is that of *polynomial reduction*. A decision problem $P1$ is called reducible to another decision problem $P2$ (noted $P1 \propto P2$) if there exists a polynomial function f , which transforms each instance of $P1$ into another instance of $P2$ so that the response for $P1$ is yes if, and only if, the response for $P2$ is yes. This means that if there exists a method to solve $P2$, then a polynomial algorithm using this method could solve $P1$. To demonstrate that a problem $P1$ is NP-complete, we have to show that it is in class NP and that there exists a problem $P2$ known to be NP-complete so that $P2 \propto P1$. Cook, in 1971 [COO 71], was the first to show the NP-completeness of a problem. This is a problem in logic, called a *satisfiability problem*. Let us note that the decision problem corresponding to the traveling salesman is an NP-complete problem [KAR 72]. It is thus not very likely that a polynomial algorithm can be found to solve the decision problem corresponding to the traveling salesman.

An optimization problem is called *NP-hard* if the corresponding decision problem is NP-complete. The traveling salesman problem is thus NP-hard.

In addition, an NP-complete problem can be *NP-complete in the strong sense* or in the *ordinary sense*. A problem is called *NP-complete in the ordinary sense* if it is NP-complete and there exists a pseudo-polynomial algorithm to solve it. To demonstrate that a problem $P1$ is *NP-complete in the strong sense*, we have to show that it belongs to the class NP and that it remains NP-complete even if we limit the size of the parameters of the problem (by a polynomial of the characteristic input data). To demonstrate that a problem is *NP-complete in the ordinary sense*, it would be necessary to show that it is NP-complete and to give a pseudo-polynomial algorithm to solve it.

The borderline between difficult problems and easy ones is often not easy to determine. In fact, the addition or the suppression of a constraint can make a problem pass from the status of an easy problem to a difficult problem or the other way around. Let us consider a depot and several clients. If we wish to find, for a given client, the shortest path connecting the depot to this client, we can solve this problem in polynomial time. On the other hand, if the objective is to find the shortest Hamiltonian cycle, we are then confronted with a problem that is NP-hard in the strong sense. In fact, this problem corresponds to that of the traveling salesman.

In this section, we have seen that the problems could be classified as easy or difficult problems. However, it is important to note that problems always exist that have not yet been classified. No polynomial algorithm has been proposed for these problems and it has also not yet been demonstrated that these problems were NP-hard. The importance of such a classification is to know if it is useful to search for an algorithm that gives us an exact solution or if it is preferable to search for an algorithm that gives us an approximate solution. We will see below this type of algorithm, called *heuristic*. Let us note meanwhile that for certain NP-hard problems, the size of the real instances is limited. This allows us to envisage the elaboration of algorithms that enable us to obtain the exact solution in a reasonable time although these algorithms may be exponential or, in the best case, pseudo-polynomial. We will bring up this point in section 3.4.5 about the exact methods.

3.4. Solution of hard problems

3.4.1. *Introduction*

As we have previously discussed, there exist combinatorial problems of different complexities. Problems belonging to class P have polynomial algorithms (efficient) making it possible to solve them. For the NP-hard problems, the existence of efficient algorithms to find the best solution seems not to be very realistic. Thus, different solution methods (exact methods or approximate) are widely used to tackle these difficult problems.

We discuss below the main methods used and indicate their functioning scheme. The minimization of an objective (minimization of the length of a tour, for example) is considered.

First of all, we will speak about the use of relaxation methods to eliminate certain constraints of the given problem in the hope of efficiently solving an easier problem. The solution found for this sub-problem is, in case of a minimization, a lower bound for the initial problem. Then we will be interested in the development of approximation methods (heuristic methods of construction and heuristic methods of improvement) to try to obtain solutions of the best quality possible in a short time. The solutions obtained are then suboptimal and constitute upper bounds for the optimal solution of the problem. Finally, exact methods, based mainly on the principles of enumeration, try to give the exact solution to the problem. The performance of such methods is measured by the size of the problems that they can solve in a reasonable time. Lower bounds and upper bounds, calculated respectively by relaxation and approximation methods, are used to reduce the enumeration as

much as possible. The execution of the exact methods depend to a great extent on the quality of these bounds.

3.4.2. *Relaxations*

The principle of *relaxation* methods is to simplify the initial difficult problem to obtain a problem that is easy to solve. For example, to solve a linear program containing variables that must be integer is a difficult problem. By relaxing this constraint, the variables can take real values. This gives a linear program that is easy to solve (see Chapter 1). The result obtained, in the case of a minimization problem, is a lower bound of the optimal solution. This simple observation for linear programs is generalized in most of the problems in combinatorial optimization.

Simplifying the problem is a first step toward the design of solution methods. Let us suppose that the relaxed linear program gives a solution in which all the variables that should be integer are indeed integer valued. The solution is then realizable because it verifies all the constraints of the problem. Knowing that the solution to the relaxed problem gives a lower bound for the initial problem, the solution obtained in this case is optimal. When some variables that must be integers are fractional, a natural heuristic is to round up these values to the nearest integer in verifying that the constraints of the problem remain satisfied.

Let us now take the example of the traveling salesman. Any Hamilton cycle of the traveling salesman problem becomes a particular tree if an edge is removed (see Chapter 2 for the formal definition of a tree). Then, the search for a tree of minimum weight is a relaxation of the original problem. The tree of minimum weight, which is easy to find, thus gives a lower bound for the value of the optimal solution and can be seen as an embryonic solution. Moreover, the very well-known heuristic of Christophides [LAW 85] uses this point of departure to construct a realizable Hamiltonian cycle for the traveling salesman problem (see Chapter 6).

Relaxation methods, because they provide lower bounds for the optimal value, are applied to branch and bound methods, which will be discussed at the end of this chapter.

3.4.3. *Heuristic methods of construction*

The objective of the development of *heuristic construction methods* is to rapidly construct good solutions for a difficult problem. These solutions can then serve as initial solutions for *improvement methods* (see section 3.4.4). The performance of such algorithms is generally calculated by the ratio between the value of the solution

produced by the heuristic and the value of the optimal solution for the *worst case*. Other performance analyses can be made. For example, the *analysis of the average* looks at the average behavior of the heuristic in calculating the ratio, not only for the worst case, but the average for several instances. In addition, when the optimal solution is not calculable (problem too complex and too big in size), it is also possible to study the behavior of the heuristic experimentally by comparing its performances with other heuristics or with *lower bounds* (calculated as a result of a relaxation, for example).

The methods of progressive construction are iterative methods where, at each iteration, a partial solution is completed. Most of these methods are *greedy algorithms* because they construct a solution without ever questioning a choice once it has been made.

Among these methods we cite the *list algorithms* that calculate, in a first phase, a list of priorities that is used, in the second phase, for the construction of the solution. This list is calculated only once and is never modified. It is to be noted that these algorithms, in spite of their simplicity, can be very efficient. A second type of method consists in choosing, at each step of the construction, the element to consider by using a *priority rule*. Note that if the choice of the element is guided by the application of a dominance theorem, the method can give the optimal solution. For each problem, numerous rules have been developed.

We will also cite the *methods of decomposition* that consist of partitioning a complex problem into several smaller problems, to search for a solution for each one of these small problems and to assemble these partial solutions.

From these construction schemes, some more or less complex heuristics have been proposed to respond to specific problems. Given their great number, we cannot cite here all the heuristics of construction developed for the traveling salesman problem. Here are just a few to illustrate the different types of methods discussed above:

- construction by increasing abscissa (*list algorithm*): the vertices are arranged in increasing order of their abscissa, then they are connected in this order;
- heuristic of the nearest neighbor (priority rule): an initial vertex is chosen, then the algorithm goes toward the nearest neighbor not already visited;
- geographic partitioning (decomposition method): a set of vertices is divided in several subsets following a grid, for example. Each subset constitutes a sub-problem solved by a heuristic or an exact method. The partial solutions are regrouped to form the final solution.

3.4.4. Improvement methods

The objective of this section is to present different methods of improvement, also called *metaheuristics*, since their vocation is to adapt to diverse problems. These methods are initialized with one (or several) feasible solution(s), calculated either randomly, or with the help of a heuristic construction (see section 3.4.3). At each iteration, these metaheuristics try to move closer to the optimal solution by applying local modifications until a stopping criterion is satisfied. We should then, before discussing the solution scheme of the different methods, define the notion of the neighborhood of a solution.

3.4.4.1. Neighborhood of a solution

For every feasible solution S , we can associate a set of neighbors, generated from S with the help of a local transformation. This transformation only modifies the solution S very locally and does not change its structure. Thus, the majority of the properties of S are kept.

It is useful, for each problem studied, to precisely define the transformation used. The neighborhood of S , $V(S)$, is then the set of the neighbors obtained from S by applying this local transformation.

To illustrate this notion of neighborhood, let us take the example of a traveling salesman problem with seven cities to visit (A, B, C, D, E, F, G). Solutions are coded using the ordered list of the vertices visited. Different possible neighborhoods could be obtained by:

- change of place of an element;
- exchange of two elements;
- inversion of a subsequence.

Let S be the following initial solution: A D C B E G F (see Figure 3.1):

- by changing the place of vertex E, we obtain $S1$: A D C B G F E;
- by exchanging the vertices A and E in S , we obtain $S2$: E D C B A G F;
- by inverting the subsequence D C B of S , we obtain $S3$: A B C D E G F.

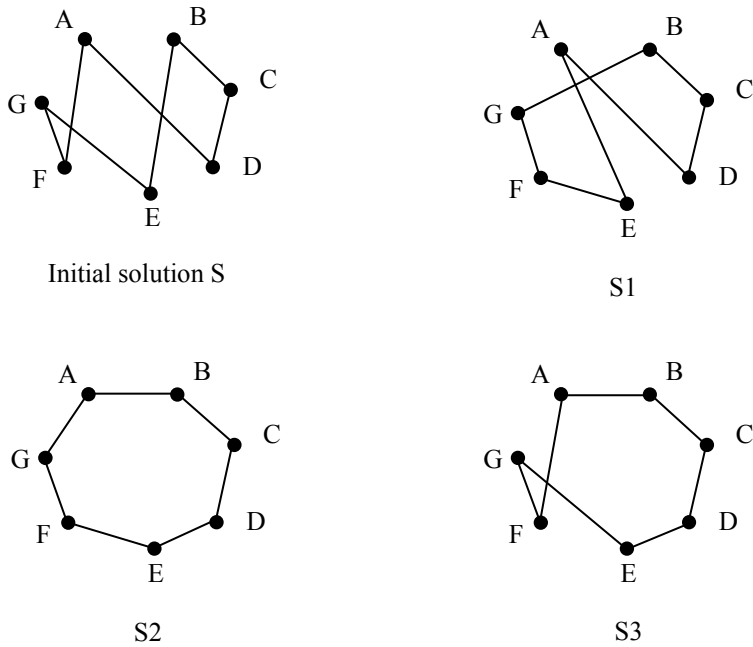


Figure 3.1. *Neighborhood of a solution*

3.4.4.2. Gradient methods

These are part of the simplest heuristics of the local search. They consist of searching in the neighborhood of the current solution (defined from a local transformation) for a lower-cost solution that then becomes the current solution. They proceed in this way until reaching a *local optimum* that can no longer be improved.

The exploration of the neighborhood can be performed randomly (random application of the transformation on the current solution) or systematically (application of the transformation with several parameters until obtaining a better solution) or even exhaustively (determination of the whole neighborhood and selection of the best solution).

Method of descent

```

/* Initializations */
  Select the initial solution  $S$ 
   $S_{better} \leftarrow S$ 
   $Cost(better) \leftarrow Cost(S)$ 
   $NbIteration \leftarrow 0$ 
/* Loop */
  While  $NbIteration < NbMaximum$  do
     $NbIteration \leftarrow NbIteration + 1$ 
    Select  $S'$  in the neighborhood of  $S$ 
    If  $Cost(S') < Cost(S)$ 
      then  $Cost(better) \leftarrow Cost(S')$ 
         $S_{better} \leftarrow S'$ 
         $S \leftarrow S'$ 
    End If
  End While

```

Algorithm 3.1. *Gradient method*

Among these methods, we describe here the *methods of exchanges* of the r -optimal type. These methods of optimization were initially proposed by Lin [LIN 65] to solve the traveling salesman problem (TSP), but they are also applied to every combinatorial problem whose solution consists of a permutation of elements. The term *r-optimal* indicates that a solution can no longer be improved by exchanging r elements. The method consists of selecting r components and checking if, by interchanging these components, a better solution may be found. We notice then that an n -optimal solution is an optimal solution (for a problem of size n). Thus, the more r augments, the more we approach an optimal solution, but the longer the calculations are. In practice, we limit ourselves to $r = 2$ or 3.

Algorithm 3.1 is a sample outline of a method of descent.

This approach has the advantage of being rapid, but remains blocked as soon as a local optimum is reached, even if it is not of good quality, because it cannot help us to find a better solution in the neighborhood than the current solution. Thus, we should look for other more sophisticated methods that allow us to move out of the local *optima*.

3.4.4.3. The tabu method

This method, whose origin goes back to 1977 [GLO 77], was formalized later, in 1986, by Glover [GLO 86]. It has no stochastic character and uses the notion of memory to avoid staying in a local optimum.

The principle of the algorithm is the following: at iterations, the neighborhood (complete or a subset of the neighborhood) of the current solution is examined. The best solution (neighboring solution with the maximal cost reduction or the one augmenting the cost by the smallest amount) is selected. By applying this principle, the method may lead us toward solutions that seem less interesting, but that perhaps have a better neighborhood.

The risk is to cycle between two solutions. To avoid the phenomenon of cycling, the method bars itself from revisiting a solution recently visited. For that purpose, a tabu list containing the last solutions visited is kept up to date. Each new solution considered eliminates from this list the solution that has been the longest on the list. We sometimes prefer to keep forbidden transformations on the list rather than the forbidden solutions. The length of the list, which is a parameter to be defined, determines the number of iterations during which a solution having been visited cannot be reconsidered. In practice, the size of the list is generally between 7 and 20 and is determined experimentally, for each type of problem, by testing several values.

Thus, the search for the following current solution is done in the neighborhood of the present current solution without considering the solutions belonging to the tabu list.

This method does not stop by itself and it is necessary to determine a stopping criterion in the function of the search time allocated. This criterion can be, for example, the execution of a certain number of iterations or the non-improvement of the best solution during a certain number of iterations. Thus, all along the algorithm, the best solution must be retained because one rarely stops with the optimal solution.

Algorithm 3.2 is an example of the outline of the tabu method.

Tabu method

```

/* Initializations */
  Select the initial solution  $S$ 
   $S_{better} \leftarrow S$ 
   $Cost(better) \leftarrow Cost(S)$ 
   $T \leftarrow \emptyset$  /* Tabu List initialized to empty /
   $NbIteration \leftarrow 0$ 
/* Loop */
  While  $NbIteration < NbMaximum$  do
     $NbIteration \leftarrow NbIteration + 1$ 
    Determine a set of candidate solutions:  $Candidates(S)$ 
    Choose  $S'$ , solution of least cost, among  $Candidates(S)$ 
    Update list  $T$  (suppression of the oldest element and addition of  $S$ )
    If  $Cost(S') < Cost(better)$ 
      then  $Cost(better) \leftarrow Cost(S')$ 
       $S_{better} \leftarrow S'$ 
    End If
     $S \leftarrow S'$ 
  End While

```

Algorithm 3.2. *Tabu method***3.4.4.4. Simulated annealing**

This class of optimization methods is due to physicists Kirkpatrick, Gelatt and Vecchi [KIR 83]. It is inspired by the simulation methods of Metropolis (in 1950) in statistical mechanics.

The historical analogy is inspired by annealing metals in metallurgy: a metal cooled too quickly presents numerous microscopic defects; it is the equivalent of a local optimum for a combinatorial optimization problem. If it is cooled slowly, the atoms rearrange themselves, the defects disappear and the metal then has a very ordered structure, equivalent to a global optimum.

The method of simulated annealing, applied to optimization problems, considers an initial solution and searches in the neighborhood for another solution able to become the current solution. The originality of this method is that it is possible to get to a neighboring solution of lower quality with a probability that is not zero. This lets us escape from the local optima. At the beginning of the algorithm, a parameter T , related to the temperature, is determined and decreases all along the algorithm to tend toward 0. The acceptance probability p of the deteriorating solutions depends

on this parameter (the higher the temperature T , the stronger this probability will be).

The performance of simulated annealing depends on the rule of cooling (that is, the decreasing of parameter T) used. Too rapid cooling would lead toward a local optimum that can be of very bad quality. Cooling too slowly would be very costly in calculation time. Adjusting the different parameters (initial temperature, number of iterations by temperature level, decrease in temperature, etc) can thus be difficult.

Simulated annealing

```

/* Initializations */
Select an initial solution  $S$ 
 $S_{better} \leftarrow S$ 
 $Cost(better) \leftarrow Cost(S)$ 
Initialize  $T$                                 /* Initialization of the temperature */
/* Loop */
While Condition 1 do
     $NbIteration \leftarrow 0$ 
    While  $NbIteration < NbMaximum$  do
         $NbIteration \leftarrow NbIteration + 1$ 
        Choose  $S'$  in the neighborhood of  $S$ 
         $\Delta Cost \leftarrow Cost(S') - Cost(S)$ 
        If  $\Delta Cost < 0$ 
            then
                 $S \leftarrow S'$ 
                If  $Cost(S') < Cost(better)$ 
                    then
                         $Cost(better) \leftarrow Cost(S')$ 
                         $S_{better} \leftarrow S'$ 
                End If
            Else
                Choose  $p$  randomly in  $[0, 1]$ 
                If  $p \leq e^{-\Delta Cost / T}$ 
                    then  $S \leftarrow S'$ 
                End If
            End If
        End While
         $T = f(T)$ 
    End While

```

Algorithm 3.3. *Simulated annealing*

Algorithm 3.3 simulates a decrease in temperature by level. For each level, a maximum number of transformations are carried out. Condition 1 represents the

stopping criterion chosen for the end of the algorithm (temperature sufficiently low, time elapsed, etc). $T = f(T)$ represents the rule of cooling of the temperature.

The interest of this class of methods is that there is a proof of convergence. Thus, if certain conditions are verified, there is a guarantee of obtaining the optimal solution.

3.4.4.5. Genetic algorithms

This class of methods is based on an imitation of the phenomenon of adaptation of living beings. The application of these methods to optimization problems was formalized by Goldberg in 1989 [GOL 89].

Genetic algorithms work in analogy to the reproduction of living beings. Contrary to the preceding methods, genetic algorithms do not consider a single current solution, but a set of solutions (*individuals*) composing a *population*. Thus, the method is initialized by a first population in which reproduction, crossover or mutation operations will be realized with the objective of also exploiting the possible characteristics and properties of this population. These operations should lead to an improvement (in terms of quality of the solutions) of the whole population since good solutions are encouraged to exchange by crossing their characteristics and creating still better solutions. However, solutions of very bad quality can also appear and allow us to avoid converging too rapidly to a local optimum.

The difficulties in applying genetic algorithms reside in the need to code the solutions and to implement different operators.

Indeed, the choice of coding is difficult, but very important. We must be able to code every solution represented by a *chromosome*, which is a chain of characters (*genes*) belonging to a certain alphabet. For permutation problems, such as problems like the traveling salesman problem, the alphabet is defined (set of cities to visit) and the objective is to look for the best position of each character (each character appearing once and only once in each chromosome). Thus, a TSP solution with seven cities could be coded by S : 2461357.

To execute the different operations, a *selection* is necessary to choose the individuals on which the operators will act. The better the quality of an individual is, the better the chances of being chosen. Having defined f_i , the force of an individual i and (*fitness*) F , the total force of the population, the probability of selection of the individual i will be proportional to f_i/F .

The crossover operator can then create, from two selected parents, one or two individual children who inherit characteristics from the parents. For that, one or several points of crossing are chosen in the chromosomes of the parents, and the pieces of chromosomes thus obtained are recombined so as to form the children. For a permutation problem such as the TSP, this operation does not guarantee the generation of feasible solutions (some cities will perhaps be repeated while others will be missing). Let us take the following example of the two parents: P1: 2461357 and P2: 6421573. A crossover after the third component would give the two solutions S1: 2461573 and S2: 6421357 (feasible solutions since each city appears once and only once), while a crossover after the fifth component would generate P3: 2461373 and P4: 6421557 that are two unfeasible solutions.

The *mutation* operator can bring diversity in the population by disturbing the solution locally. It can be inspired by local transformations discussed during the presentation of the notion of neighborhood, like the exchange of two components, for example. Thus, solution 2461357 could, after mutation, become 2471356.

Finally, the *reproduction* operator will, from new individuals, regenerate a new population, by keeping the good individuals without systematically eliminating the less good individuals.

Thus, if these algorithms are very rich regarding the diversity of solutions generated, the parameters (size of the population, coefficient of reproduction, probability of mutation, etc) are difficult to define and demand quite a large calculation effort.

3.4.5. Exact methods

In the preceding sections, we have seen that in order to find one or more solutions to difficult problems, several heuristic methods can be developed. However, it can be necessary to know one or more exact solutions and this for several reasons, notably to evaluate the performances of the proposed heuristics. We have also seen that for difficult problems it is not very probable that a polynomial (in certain cases a pseudo-polynomial) algorithm will be found. The exact algorithms that can be developed for this type of problem are in most cases exponential algorithms. The result is that these algorithms cannot be used except for instances of relatively small size. Note however that for certain types of problems, the maximal size of instances that can be solved by exact algorithms correspond to the maximal size routinely encountered in real applications. In this section, we will try to describe two main types of exact methods. We will first present the method of *branch and bound*; then we will discuss dynamic programming. These two methods

are among the most used methods. However, other methods that we do not describe here can be developed, notably the Lagrangian relaxation.

3.4.5.1. *Branch and bound*

The objective of a *branch and bound procedure* is to search for an optimal solution by making an intelligent enumeration, which will not necessarily be exhaustive, of the set of solutions. This research is founded on the construction of a *tree*. In addition, the method of branch and bound is constituted of two procedures: a *branching* procedure and a *bounding* procedure. We will first describe the structure of the search tree used, constructed by the method of branch and bound; then we will describe the bounding procedure and, finally, the branching procedure. Throughout, we assume that the objective function is a minimization.

The root of the search tree corresponds to the empty solution. The leaves are elements of the set of solutions. Each node of the tree corresponds to a partial solution. At each node, the solution is generated from the solution of the node that precedes it in the tree. Each node appears at a certain level in the tree. Level 0 corresponds to the root. Level 1 corresponds to the nodes of the first partial solutions generated. Level 2 regroups the nodes of the solutions generated from the solutions of the nodes in level 1. For example, the complete search tree for a *traveling salesman problem* with three cities (A, B, C) would be that presented in Figure 3.2. Note that certain leaves correspond to equivalent solutions. Indeed, as the solution to the traveling salesman is a cycle, the solutions ABC, BCA and CAB are equivalent, and the same is true for solutions ACB, CBA and BAC.

We will now discuss the bounding procedure. Each node is associated with a value. In the case of a leaf, this value corresponds to the value of the objective function for the solution associated with this leaf. In the case of a node that is not a leaf, this value represents a *lower bound* of the objective function for the solution partially constructed in this node. In the case of the traveling salesman, the value associated with a leaf is the length of the cycle and that associated with the nodes could, for example, be the corresponding value of the distance covered in the partial solution.

To obtain all the solutions, the search tree must be completely developed and, in this case, the bounding procedure is not used. On the other hand, if the research objective is to find an optimal solution in a relatively reasonable time, the complete search tree must not be generated; it is thus necessary to determine for which criteria certain branches can be cut. For that, the value associated with each node will be used. In addition, an upper bound must be defined. It will either be calculated from a heuristic or it will correspond to the smallest value of the leaves already generated.

For each node generated, the following verification is done. If the value associated with the node is strictly greater than the upper bound, then the node is abandoned. To facilitate the comprehension of this procedure, consider the problem of the traveling salesman. Suppose that at a certain stage in the search a partial solution S , for which the distance covered is strictly superior to the length of a known cycle T , is found. The partial solution S will never have a cycle shorter than T ; it is thus not necessary to generate solutions from the node of S . Values associated with the nodes are not the only elements that allow us to cut branches; the particularities or the known properties of the problem must be used.

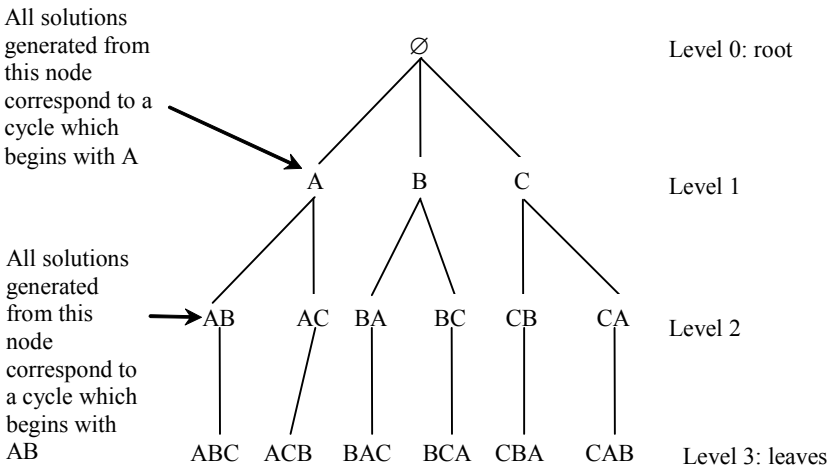


Figure 3.2. Complete search tree

We will now discuss the branching procedure that lets us construct and go through this search tree. To construct and cover this tree, we have two possibilities: depth first or width first approaches. We will describe these two types of approach and to illustrate our proposal we will use a traveling salesman with four cities (A, B, C, D) whose distances are given in Table 3.1.

The procedure in *depth first* takes place in the following manner. From a root, a first partial solution at level 1 is generated and evaluated. From this first partial solution, a partial solution at level 2 is generated and evaluated and so on until reaching a leaf. If the value associated with this leaf is strictly greater than the upper bound, then this leaf is cut; if not, this value becomes the new *upper bound*. Then the branching proceeds by *backtracking* in order to go back up along the branch until finding a node for which all the possible solutions have not been generated and

so on until there are no more solutions to generate. To facilitate comprehension, Figure 3.3 presents the different steps in this procedure, for the example for the traveling salesman with four cities described in Table 3.1. Note that the structure of the problem makes it possible to diminish the size of the tree. Indeed, the solution sought is a cycle; it is of no worth to know which city will be visited first. Also, to explore all the solutions, it suffices to consider only the cycles where the first city is A. In addition, we consider as an initial upper bound the value 47, which is the length of cycle A, B, C, D.

Destinations Origins	A	B	C	D
A		24	4	3
B	20		6	5
C	7	5		14
D	3	3	16	

Table 3.1. *Distance table*

The procedure in *width first* takes place in the following manner. From the root, the first level of solutions is generated. All the solutions are evaluated. The second level of solutions is generated from the solutions in the preceding level that were not abandoned and so on until all the nodes generated are leaves. The most promising solutions, which are those having the weakest associated value, are developed first. As soon as a leaf is found, it is evaluated and its value is compared to the *upper bound*. If the value associated with the leaf is less than the upper bound, then this solution becomes the new upper bound. Figure 3.4. presents the different steps of this procedure for the traveling salesman problem with four cities described in Table 3.1. For the same reasons as before, only the tours beginning at A are considered. As before, we consider as an initial upper bound, the value 47 that is the length of the tour A, B, C, D.

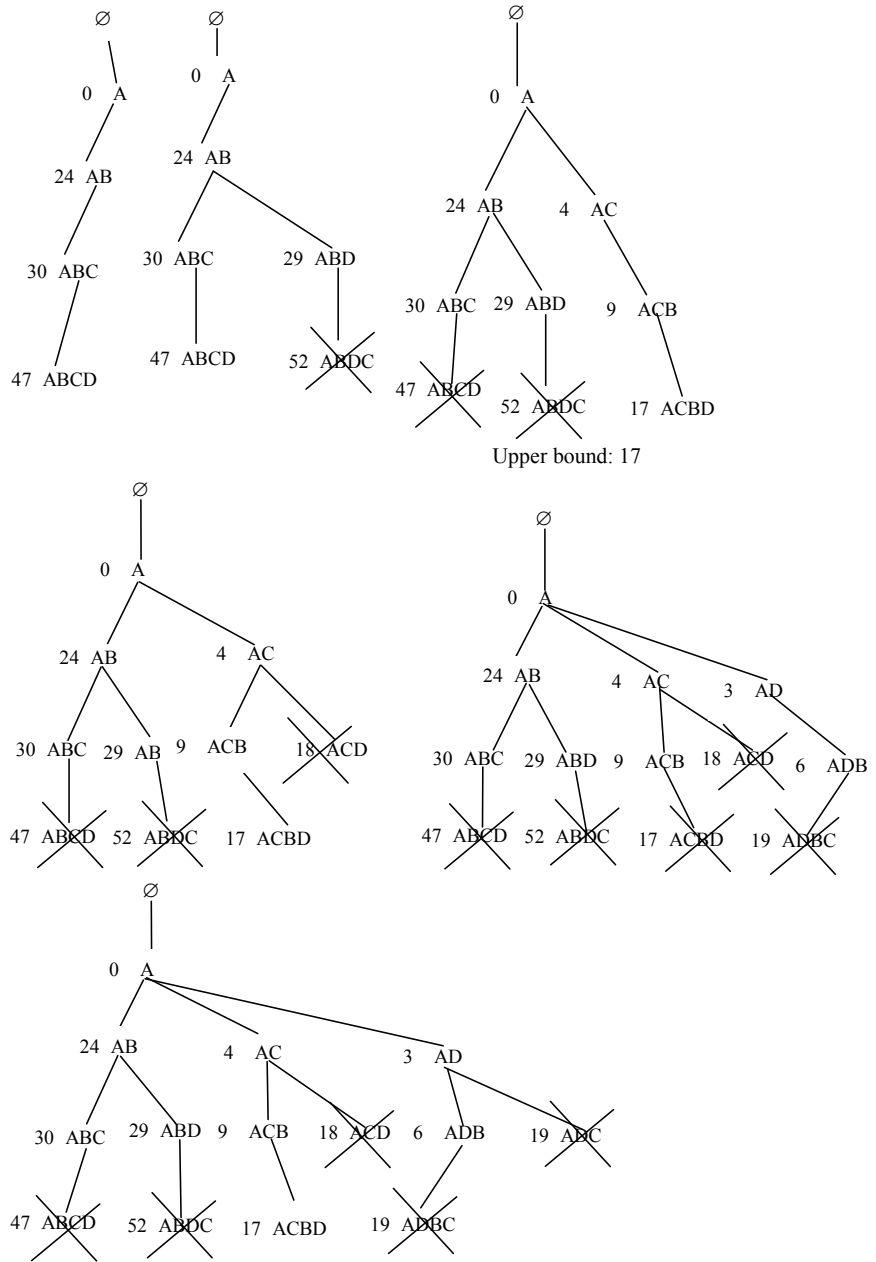


Figure 3.3. Depth first procedure

An efficient branch and bound procedure is a procedure where an optimal solution is obtained without having to enumerate all the nodes of the search tree. Indeed, let us consider a traveling salesman problem with n cities. The number of leaves of the complete search tree is $(n-1)!$ if a single initial city is considered. If $n = 5$, we obtain $4!$ ($=24$) leaves. If we now consider a problem with 50 cities, the number of leaves becomes $49!$ which reaches $6 \cdot 10^{62}$ solutions. The quality of the upper and lower bounds used is decisive. However, it is important to find equilibrium between the time to develop the tree and the time to calculate the bounds. To obtain interesting bounds, the particularities of the problem considered must be used. Note as well that the procedure in width first demands more memory space than the procedure in depth first since the number of nodes to keep at each step (nodes not developed and not abandoned) is more important.

3.4.5.2. Dynamic programming

The *dynamic programming* method [BEL 57, BEL 62] is based on a recursive description of optimization problems. Dynamic programming cannot be used for all problems. Indeed, it is based on the fact that the optimization problem, or more precisely the value of the objective to optimize, must be able to be written in the form of an equation using recurrence. The value of the objective is calculated step by step. To calculate the value of the objective at a given stage, only a subset of values for the objective from the preceding steps must be known. It is thus not necessary to know all the choices made since the beginning. The choice made at a step affects the following step. The principle of optimality in this method is not verified for all optimization problems. However, it is important to note that numerous problems can be solved using dynamic programming.

Dynamic programming can notably be used for the problem of the shortest Hamiltonian path (passing through all the cities). In this problem, the objective is to find a path of minimum length leaving the city of origin o and passing through n cities of a set X . In dynamic programming, the length of such a path will be calculated from the length of a path passing through the $n-1$ cities of the set $X - \{o\}$. Dynamic programming takes into account the path passing through $n-1$ cities, which minimizes the length of the path passing through n cities. Let c be the matrix of the distances between the cities, let $c(i, j)$ be the distance (or the cost) to go directly from i to j , and let $long(X, o)$ be the function that returns the length of the shortest path leaving from o and passing through all the cities of X . It can be calculated by the following formula of recurrence:

$$long(X, o) = \min_{k \in \{X - \{o\}\}} \{long(X - \{o\}, k) + c(o, k)\}$$

where k represents here the origin of a path passing through all the cities of $X - \{o\}$ and allows us to obtain a path of optimal length passing through all the cities of X and having o as origin (see Figure 3.5).

Finally, if X only contains a single city that is the city of origin, then the length of the shortest path leaving from o and arriving at o is 0.

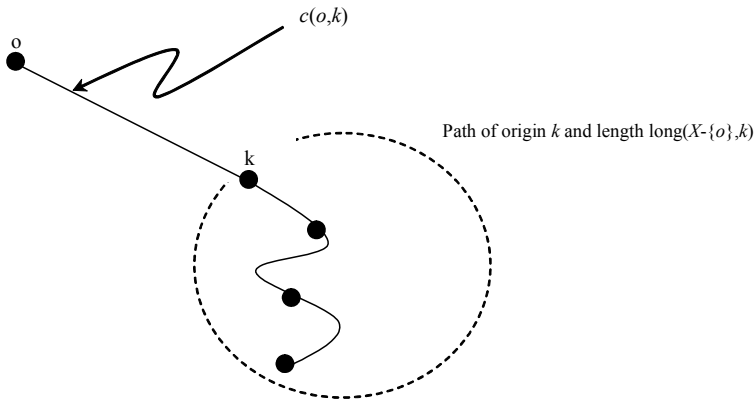


Figure 3.5. *Determination of the shortest path by dynamic programming*

The shortest path problem can be modeled by dynamic programming in the following manner:

– recurrence formula:

$$\text{long}(X, o) = \min_{k \in \{X - \{o\}\}} \{ \text{long}(X - \{o\}, k) + c(o, k) \}$$

– initial condition:

$$\text{long}(\{o\}, o) = 0$$

To facilitate comprehension, we now use this algorithm to solve a problem of the shortest path by considering the distances in Table 3.1, and with city A as the city of origin.

Here, $X = \{A, B, C, D\}$. The length of an optimal path is calculated in the following manner:

$$long(X, A) = \min_{k_1 \in \{B, C, D\}} \{long(\{B, C, D\}, k_1) + c(A, k_1)\}$$

To find the minimum value, three calculations have been made (Table 3.2).

For each case presented in Table 3.2, two calculations have been made, as shown in Table 3.3.

Results presented in Table 3.3 allow us to evaluate each of the cases presented in Table 3.2 (see Table 3.4).

This enables us to calculate the length of the optimal path passing through all the cities of X and having A as origin:

$$\begin{aligned} long(X, A) &= \min_{k_1 \in \{B, C, D\}} \{long(\{B, C, D\}, k_1) + c(A, k_1)\} \\ &= \min\{44; 14; 12\} = 12 \end{aligned}$$

As seen before, the value of the objective at each stage depends on the results obtained by a subset of the preceding stages, notably by the preceding stage. Let us consider the traveling salesman problem; the length of a tour composed of k cities will be calculated from tours of length $k-1$ that is $(k-1)!$ tours. The complexity of an algorithm in dynamic programming is in this case exponential since such an algorithm is in $O((n-1)!)$ where n is the number of cities to visit. It is however important to note that for certain NP-hard problems in the ordinary sense, pseudo-polynomial algorithms in dynamic programming have been, or can be, proposed. These algorithms make it possible to solve in an exact manner and in a reasonable time instances of small size, which can be enough for some practical problems.

In this section, we have seen two exact methods. These methods allow us to obtain an optimal solution, but their complexity does not always allow us to obtain a solution in a reasonable time. They are however very useful in order to evaluate the performance of heuristics or to solve instances of small size.

	$long(X, A)$
$k_1 = B$	$ \begin{aligned} long(X, A) &= long(\{B, C, D\}, B) + c(A, B) \\ &= \min_{k_2 \in \{C, D\}} \{long(\{C, D\}, k_2) + c(B, k_2)\} + c(A, B) \\ &= \min_{k_2 \in \{C, D\}} \{long(\{C, D\}, k_2) + c(B, k_2)\} + 24 \end{aligned} $
$k_1 = C$	$ \begin{aligned} long(X, A) &= long(\{B, C, D\}, C) + c(A, C) \\ &= \min_{k_2 \in \{B, D\}} \{long(\{B, D\}, k_2) + c(C, k_2)\} + c(A, C) \\ &= \min_{k_2 \in \{B, D\}} \{long(\{B, D\}, k_2) + c(C, k_2)\} + 4 \end{aligned} $
$k_1 = D$	$ \begin{aligned} long(X, A) &= long(\{B, C, D\}, D) + c(A, D) \\ &= \min_{k_2 \in \{B, C\}} \{long(\{B, C\}, k_2) + c(D, k_2)\} + c(A, D) \\ &= \min_{k_2 \in \{B, C\}} \{long(\{B, C\}, k_2) + c(D, k_2)\} + 3 \end{aligned} $

Table 3.2. First stage of dynamic programming

		<i>Long</i> ($X - k_1, k_2$)
$k_1 = B$	$k_2 = C$	$long(\{C, D\}, C) = \min\{long(\{D\}, D) + c(C, D)\}$ $= c(C, D) = 14$
	$k_2 = D$	$long(\{C, D\}, D) = \min\{long(\{C\}, C) + c(D, C)\}$ $= c(D, C) = 16$
$k_1 = C$	$k_2 = B$	$long(\{B, D\}, B) = \min\{long(\{D\}, D) + c(B, D)\}$ $= c(B, D) = 5$
	$k_2 = D$	$long(\{B, D\}, D) = \min\{long(\{B\}, B) + c(D, B)\}$ $= c(D, B) = 3$
$k_1 = D$	$k_2 = B$	$long(\{B, C\}, B) = \min\{long(\{C\}, C) + c(B, C)\}$ $= c(B, C) = 6$
	$k_2 = C$	$long(\{B, C\}, C) = \min\{long(\{B\}, B) + c(C, B)\}$ $= c(C, B) = 5$

Table 3.3. *Second stage of dynamic programming*

	$long(X, A)$
$k_1 = B$	$long(X, A) = \min\{14 + 6 ; 16 + 5\} + 24 = 44$
$k_1 = C$	$long(X, A) = \min\{5 + 5 ; 3 + 14\} + 4 = 14$
$k_1 = D$	$long(X, A) = \min\{6 + 3 ; 5 + 16\} + 3 = 12$

Table 3.4. *Third stage of dynamic programming*

3.5. Conclusion

In this chapter, we have presented classical combinatorial optimization problems according to their common characteristics. This list is of course not exhaustive and numerous other problems can be found in various works [AHU 93, BLA 96, DIE 00, DRE 96, GOL 88, RAD 97].

We have presented the basic elements of the theory of complexity. This theory allows us to separate the problems into two categories: those for which efficient algorithms exist to obtain an optimal solution, and those for which heuristic approaches are better adapted. Additional information can be found in the work of Garey and Johnson [GAR 79].

To conclude, we have first presented the principal heuristic techniques to find approximate solutions to difficult problems (constructive heuristics and improvement methods). Second, we have described the principal techniques to obtain an optimal solution, without however having any guarantee of the computation time to obtain this solution: branch and bound and dynamic programming.

3.6. Bibliography

- [AHU 93] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Englewood Cliffs, New Jersey, Prentice Hall, 1993.
- [BEL 57] R. Bellman, *Dynamic Programming*, Princeton, New Jersey Princeton University Press, 1957.
- [BEL 62] R. Bellman and S.E Dreyfus, *Dynamic Programming*, Princeton, New Jersey, Princeton University Press, 1962.
- [BLA 96] J. Blazewicz, K. Ecker,,E. Pesch, G. Schmidt and J. Weglarz, *Scheduling Computer and Manufacturing Processes*, Berlin, Heidelberg, Springer-Verlag, 1996.
- [COO 71] S.A. Cook, “The complexity of theorem-proving procedures”, *Proceedings of the 3rd Annual ACM Symposium, Theory of Computing*, 1971, 151–158.
- [COO 97] W.J. Cook and W.H. Cunningham, *Combinatorial Optimization*, New York, John Wiley & Sons, 1997.
- [DIE 00] R. Diestel, *Graph Theory*, Springer-Verlag, 2000.
- [DRE 96] Z. Drezner, *Facility Location: A Survey of Applications and Methods*, Springer-Verlag, Springer Series in Operations Research, 1996.
- [GAR 79] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, New York, Freeman and Company, 1979.
- [GLO 77] F. Glover, “Heuristic for integer programming using surrogate constraints”, *Decision Sciences*, vol. 8, 1977, 156–166.
- [GLO 86] F. Glover, “Future paths for integer programming and links to artificial intelligence”, *Computers and Operations Research*, vol. 13, no. 5, 1986, 533–549.
- [GOL 89] D. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning*, Pennsylvania, Addison Wesley, Reading, 1989.
- [GOL 88] B.L. Golden, *Vehicle Routing: Methods and Studies*, Elsevier Science, 1988.
- [KAR 72] R.M. Karp, “Reducibility among combinatorial problems”, in R.E. Miller and J.W. Thatcher (eds.), *Complexity of Computer Computations*, New York, Plenum Press, 85–103, 1972.
- [KIR 83] S. Kirkpatrick, C. Gelatt and M. Vecchi, “Optimization by simulated annealing”, *Science*, vol. 220, 1983, 671–680.
- [LAW 85] E.L. Lawler and A.H. Rinnooy-Kan, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, New York, John Wiley & Sons, Wiley-Interscience Series in Discrete Mathematics, 1985.

- [LIN 65] S. Lin, “Computer solutions of the traveling salesman problem”, *Bell System Technical Journal*, vol. 44, 1965, 2245–2269.
- [MAR 90] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, New York, John Wiley & Sons, Wiley Interscience Series in Discrete Mathematics and Optimization, 1990.
- [RAD 97] R.L. Radin, *Optimization in Operations Research*, Englewood Cliffs, New Jersey, Prentice Hall, 1997.
- [ROY 93] B. Roy and D. Bouyssou, *Aide multicritère à la décision: méthodes et cas*, Paris, Economica, 1993.

Chapter 4

Project Scheduling

4.1. Presentation

4.1.1. *Conducting a project*

A project is a methodical approach that is carried out to attain a clearly specified objective in a limited time. To computerize a service, construct a dam or school, or create a new product is a project. A project is broken down into a set of operations (or tasks, in the vocabulary of project management) connected by precedence constraints. A task generally necessitates the utilization of appropriate resources in order to be realized. The object of project management is to coordinate the execution of tasks, while assuring the coherent use of the available resources, to attain the objective in the allotted time.

Conducting the project demands the putting into place of a temporary organization appropriate to the project. This entails three phases: planning, scheduling, and monitoring and controlling.

The planning phase by definition begins with the objectives, constituting the project team, identifying what will be needed and determining the performance criteria (quality, costs, time limits).

The scheduling of the project realizes the operational follow-up of the project: resource management, follow-up of progress, launching activities. Technically, scheduling a project consists of programming the execution of tasks in time while respecting the constraints and optimizing the performance criteria. It is especially at this stage that the operational research techniques presented below are involved.

Monitoring and controlling consists of those processes performed to observe project execution so that variation between the scheduled program and the one carried out, or potential problems, can be identified in order to put corrective actions in place. This follow-up can be efficient not only in the planning, but also in the follow-up of the costs.

4.1.2. Definitions

4.1.2.1. Tasks (or activities)

Tasks are the elementary activities that must take place to attain the objective. The non-execution of a task leads, strictly speaking, to the stopping of the project or questioning the objectives fixed. The definition of the tasks can be a delicate point. For a project spread over several months, we can be content at departure to roughly define the final tasks. We would be content, for example, to consider the task: “put up the roof”. As time passes, this task will be broken down into finer tasks such as: “put up the frame”, “place the tiles”, etc. A task is characterized by a beginning and an end, and is connected to the other tasks by one or several precedence relationships.

4.1.2.2. Resources

A task generally requires certain material means, financial and human, in order to be realized. Generically, these means are called resources. We distinguish two types of resources. We say that a resource is consumable if it is consumed during the realization of the task (money, materials). We say that it is renewable when, the task completed, it is once again available for other tasks (machine, operator, etc). If these resources exist in limited quantity, we are led to share them among the tasks. We will speak about scheduling with resources.

4.1.2.3. Constraints

The precedence relations and the management of resources will cause constraints for the project management. We distinguish four types of constraints.

Precedence constraints among tasks

These specify the sequence of the operations (the logic of the project). The most common are of the form: task A must be finished before task B can begin (end-beginning relationship). We can have partial overlapping: task B can begin if 3/4 of task A is completed. We also find minimum spacing constraints between the end of A and the beginning of B (constraints of drying for example).

The constraints of temporal position

These specify the time interval (or the semi-interval) during which it is possible to realize a task. These constraints are often due to the availability of the staff (human resources); for example, the company that puts up the frame can only come between 15 June and 31 July.

REMARK. We have distinguished here the precedence constraints and the constraints of temporal position. Mathematically, the precedence constraints are shown by the fact that the beginning (or the end) of the task x depends on the beginnings (or ends) of its predecessors, and the time constraints by the fact that the beginning (or the end) of the task x depends on a constant. There is thus no difference between these constraints. They are regrouped under the term of potential constraints.

Disjunctive constraints

A disjunctive constraint imposes the simultaneous non-realization of two tasks A and B. We find such constraints in the case of the utilization of a resource present in a single unit (a crane, a team, etc) or in the case of formulating simultaneous realization bans for security reasons or placement problems. To arbitrate a disjunctive constraint consists of deciding if A will be done before B or the other way round.

Cumulative constraints

We speak of cumulative constraints when the tasks demand a part of one or several resources present in limited quantity. The problem is much more combinatorial than for disjunctive constraints. Consider the example where we have five workers and five tasks to accomplish. Each task demands the presence of a certain number of these workers (see Table 4.1).

Task	A	B	C	D	E
Number of workers	4	3	2	1	1

Table 4.1. *Example of cumulative constraints*

For the sequencing to be realizable, we must use at every moment at most five different actors. This constraint forbids the following schedules to be realized in parallel: (A//B), (A//C), (A//D//E), (B//C//D), (B//C//E). These configurations are minimal in the sense that, if (A//B) is forbidden, every configuration containing (A//B) is also forbidden (for example: A//B//D, A//B//C//E, etc.).

If a minimum configuration involves only two tasks (here A//B or A//C), we can replace it by a disjunctive constraint between these two tasks.

4.1.3. *Scheduling methods*

The very first method used in scheduling dates from the 1920s. This method, attributed to Henry L. Gantt, used bars whose lengths were proportional to the duration of the tasks. The Gantt charts are no longer manually realized, but remain the favored tool for the visualization of schedules.

With the development of computers in the 1950s, three methods were developed almost simultaneously:

- the PERT (Program Evaluation and Review Technique) developed for the conception, manufacture and launching of the Polaris missile;
- the CPM (Critical Path Method) perfected by the DuPont de Nemours company;
- the MPM (Metra Potential Method) created by the Frenchman Bernard Roy to plan the construction of the ocean liner France.

The principal difference between PERT and the two others was the taking into consideration random durations, which enabled the giving of probabilities of the beginning and end dates. PERT and CPM represented a schedule as a graph in which a task is described by an arc, while for MPM a task is represented by a vertex. Over time, these methods have more or less merged. In France, the term PERT is used as a synonym for project scheduling, whatever method is used.

4.2. *Scheduling and graphs without cycles*

In the scheduling phase of the project, we have a description of each of the tasks:

- general information (objective, person in charge, workers, codification);
- scheduled or estimated duration;
- list of precedence constraints;

- necessary resources;
- possibly the costs, charges.

All this information gives us two levels of control and validation. Level 1 consists of controlling the accuracy of the information, notably that no task is missing, that the durations are not overestimated (underestimation rarely happens). This validation is human.

Level 2 consists of verifying the coherence of the set of the data. For small projects, this coherence is verified by making a graph of the project. The most natural graph is to associate a vertex with each task and to trace an arc from A to B if A precedes B. A circuit is a path that, departing from a vertex A, makes it possible to return to this vertex A (see Figure 4.1).

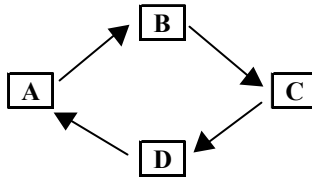


Figure 4.1. *Circuit*

If the project is coherent, the corresponding graph is a graph without a circuit. These graphs have simple properties enabling us to obtain algorithms with an efficient computation.

PROPERTY 4.1. We call *source* a vertex without a predecessor and *sink* a vertex without a successor. In a graph without circuit, there is at least one source and at least one sink.

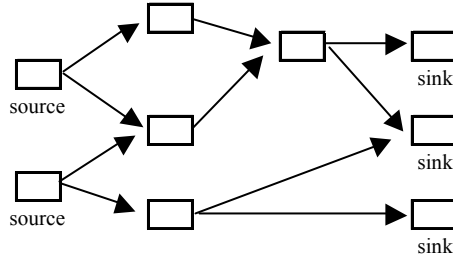


Figure 4.2. *Source and sink*

PROPERTY 4.2. We say that a graph without circuit is normalized if it has a unique vertex source *start* and a unique vertex sink *end*. Let us consider a graph without circuit with several sources (S_1, \dots, S_a) and several sinks (P_1, \dots, P_b). We can normalize it by adding:

- a vertex *start* and arcs ($start, S_i$);
- a vertex *end* and arcs (P_j, end).

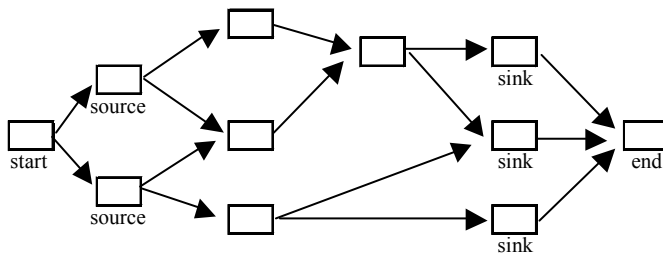


Figure 4.3. *Normalized graph*

In scheduling problems, the vertex *start* is a fictitious task of no duration, corresponding to the beginning of the project and *end* is the fictitious final task.

PROPERTY 4.3. We call a topological order of a normalized graph with n vertices, a renumbering of the vertices from 1 to n so that:

- $number(start) = 1$;

- $number(end) = n$;
- for each x , we have for each predecessor y of x : $Number(x) > Number(y)$.

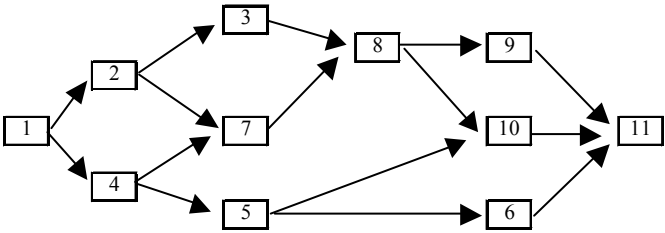


Figure 4.4. One of the topological orders of the graph

Figure 4.4 gives a possible topological order of the graph. For a given graph there are generally many topological orders. The method that we will present allows us at the same time to facilitate the drawing of the graph and to find one of these orders.

EXAMPLE 4.1. Let us consider the construction project (very simplified) given in Table 4.2 where the tasks have been coded from A to J.

Task	Predecessor	Duration
A		2
B		2
C	H	2
D		3
E	B, G	4
F	C, I	2
G	A, D	3
H	B, D	4
I	H	5
J	C	3

Table 4.2. Data of the example

At level 0, we place the fictitious vertex at the start of the project. At level 1 are placed the tasks without predecessors (A, B, D). We eliminate these tasks in the list of predecessors. If there is no circuit, at least one new task can be found without a predecessor. Here we find G and H. They will be placed at level 2. We eliminate in turn G and H. The tasks C, E and I are without predecessors and will be at level 3. Their elimination makes F and J appear at level 4. We place the vertex *end* in level 5.

Task	Level 1	Level 2	Level 3	Level 4
A	*			
B	*			
C	H	H	*	
D	*			
E	B, G	G	*	
F	C, I	C, I	C, I	*
G	A, D	*		
H	B, D	*		
I	H	H	*	
J	C	C	C	*

Table 4.3. *Determination of a topological order based on levels*

By playing a little with the placement of the tasks at the same level, we obtain the graph in Figure 4.5. By considering the tasks level by level and from bottom to top, we obtain a topological order (Table 4.3).

Following this, we suppose that the projects have been normalized, and that the tasks have been renumbered from 1 to n . As we notice, the fact of considering the tasks in topological order lets us systematically explore the graph and assures us that the necessary data to treat a task are available.

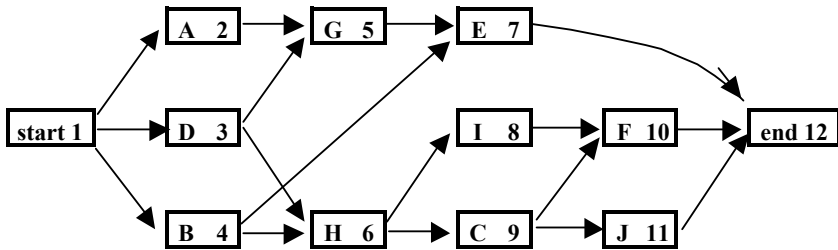


Figure 4.5. Graph of Example 4.1

4.3. Fundamental problem

In the fundamental problem, the durations of the tasks are known perfectly. The date of the beginning of the project is also known. The only constraints that we take into account are the constraints of succession of the type: task y must be finished before the beginning of task x . We note $P(x)$ the set of the predecessors of x , and $S(x)$ the set of the immediate successors of x . The resources are supposed to be in sufficient quantity so as not to create constraints. The objective sought is to determine the periods of time during which the tasks must be executed if one wants to finish the project in a minimum of time or for a given date.

4.3.1. Calculation of the earliest dates

No task A can begin before all the tasks T that precede it are finished:

$$\text{EarliestStart}(A) = \max\{\text{EarliestEnd}(T), T \in P(A)\}.$$

If the tasks are indexed according to a topological order, the index of task A is greater than the indices of all its predecessors. By considering the tasks in the order of 1, 2, 3, ..., n we can calculate the set of the earliest dates all at once. Let us look at Example 4.1, considering that the project starts on date 0. We will use Figure 4.6 to note the results. The fictitious start task begins and finishes at date 0. A, D and B start at 0 and finish respectively at 2, 3 and 2. G can start when A and D are finished, or at date 3. H can start when D and B are finished, or at date 3. By considering the tasks from left to right and from bottom to top, we find the earliest start and finish dates.

With the tasks being numbered according to a topological order, the general algorithm to calculate the earliest dates is as follows:

Calculation of earliest dates
 $EarliestStart(1) \leftarrow DateStartProject$
 $EarliestEnd(1) \leftarrow DateStartProject$
For x from 2 to n
 $EarliestStart(x) \leftarrow Max\{EarliestEnd(y), y \in P(x)\}$
 $EarliestEnd(x) \leftarrow EarliestStart(x) + Duration(x)$
End for
 $DateEndProjectMinimal \leftarrow EarliestStart(n)$
End

Algorithm 4.1. Calculation of earliest dates

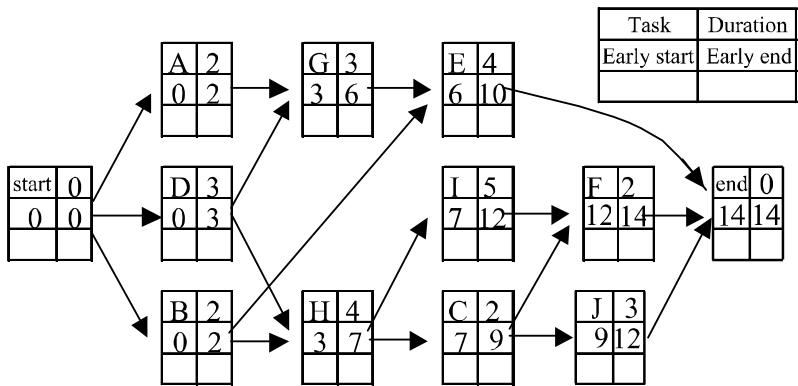


Figure 4.6. Calculation of earliest dates in Example 4.1

4.3.2. Calculation of the latest dates

The calculation of the earliest dates enables us to obtain the minimum date for the end of the project. Note that *DateEndProject* is the date at which we want to end the project. This date is generally equal to or greater than the minimum date found.

If not, the constraints are too strong and we know in advance that the project will be late.

If we know the start dates of the successors of x , it will be necessary to respect these dates so that x can be finished before any successor starts. We thus have:

$$LatestEnd(x) \leftarrow \min\{LatestStart(z), z \in S(x)\}.$$

We will consider the tasks in the inverse order of the topological order. We obtain the following algorithm for the calculation of the latest dates:

Calculation of the latest dates

$LatestStart(n) \leftarrow DateEndProject$

$LatestEnd(n) \leftarrow DateEndProject$

For x from $n-1$ to 1

$LatestEnd(x) \leftarrow \min\{LatestStart(z), z \in S(x)\}$

$LatestStart(x) \leftarrow LatestEnd(x) - Duration(x)$

End For

End

Algorithm 4.2. *Calculation of the latest times*

Let us once again use Example 4.1 when supposing that we are trying to finish accurately, that is, at date 14 (see Figure 4.7). To respect this date, J must be finished at the latest at date 14, and will therefore start at the latest at date 11. F must be finished at the latest at date 14 and start at date 12. C must be finished before J and F start, that is at date 11 (because of J). In reading the graph from right to left and from bottom to top (or from top to bottom), we obtain the set of the latest dates.

In a small project, we read the arranged graph once from left to right for the earliest dates and once from right to left for the latest dates. Note that in the results the dates correspond to the starts of the periods. Task A lasts two periods. If it starts at period 0, it will be executed during the periods 0 and 1, and thus be finished at the start of period 2. Suppose that the durations are in days and that the date 0 is on Monday 10. In everyday language we say that A ends on Tuesday 11 (at the end of the day is understood) and not that A ends on Wednesday 13 at 0 hour. We must take this into account in the presentation of the results.

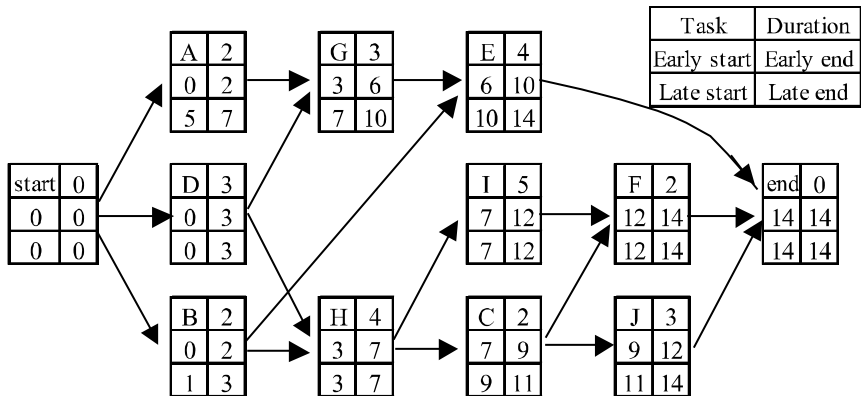


Figure 4.7. Calculations of latest times for the example

4.3.3. Margins

We call the total margin (or simply, margin) the value:

$$TotalMargin(x) = LatestStart(x) - EarliestStart(x).$$

The margin represents the freedom that we have for x without modifying the end date of the project. We call a *critical task* a task whose margins are equal to zero. We call a *critical path* every path that from *start* to *end* is constituted of critical tasks. In the preceding example, there is only one critical path, formed by the tasks *start*, D, H, I, F, *end*. When the margin of x is positive, we have a certain freedom to begin, or to augment the planned duration, or to play with the two. Let us consider task E. It can begin between the dates 6 and 10. Its margin is 4. The planned duration of E is 4 days. The company charged with realizing E will be able to start on date 8 for example and spread the work over 6 days without delaying the end of the project. If the freedom taken is equal to the total margin, the task no longer has any margin and becomes critical, just like a certain number of the tasks that follow it. If the freedom exceeds the total margin, the surplus affects the end date of the project that is delayed by this amount.

The *open margin* of task x is equal to the difference between the minimum earliest starts of the successors of x and the earliest end of task x :

$$OpenMargin(x) = \min\{EarliestStart(z) - EarliestEnd(x), z \in S(x)\}.$$

The open margin represents the degree of freedom that we have for x without affecting any other task. A delay greater than the open margin (but, however, less

than the total) affects the following tasks by diminishing their margins (because the earliest start of at least one successor is delayed). Let us consider task A in the example. It can begin between the dates 0 and 5 without delaying the end of the project. Its total margin is 5. Its only successor is G who can start at the earliest at date 3. Since A has duration of 2, A can start at 0 or 1 without modifying the dates of G. On the other hand, if A starts on 2, 3, 4 or 5 the earliest start dates of G and its successors will be modified, but without the project being delayed.

Number	Code	Total margin	Open margin
1	Start	0	
2	A	5	1
3	D	0	
4	B	1	1
5	G	4	0
6	H	0	
7	E	4	4
8	I	0	
9	C	2	0
10	F	0	
11	J	2	2
12	End	0	

Table 4.4. *Margins of Example 4.1*

4.4. Visualizations

4.4.1. Representation on the graph PERT/CPM

Until now we have represented a task by a vertex, by putting an arc from A to B if A must be finished before the start of B. In the PERT and CPM methods, we again refer to the logical method of Gantt (one bar per task). Task A is represented by an arc (dA, fA). The vertex dA represents the start of A, the vertex fA the end of A, and the length of the arc (dA, fA) is equal to the duration of A. If A must be finished before the start of B, we create a fictitious arc (fA, dB) of no duration.

This initial graph is of little interest. The goal of the game then is to suppress the fictitious arcs that are useless in order to obtain a condensed representation (the PERT graph is reputed to be more concise). For a given project there can be several minimum PERT representations. Note also that the construction of a PERT graph with a minimum number of arcs is an NP-complete problem. In Figure 4.8 we give a PERT graph of Example 4.1. There remain four fictitious arcs represented by dotted lines.

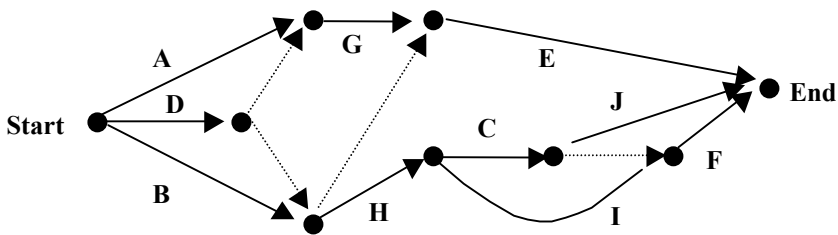


Figure 4.8. *PERT/CPM graph for Example 4.1*

4.4.2. Gantt chart

In Gantt charts, each task is represented by a bar (a rod, a line, etc) whose length is proportional to the duration. We can take two lines per task, one for the earliest dates and one for the latest dates, with possibly a third line representing the real advancement of the work.

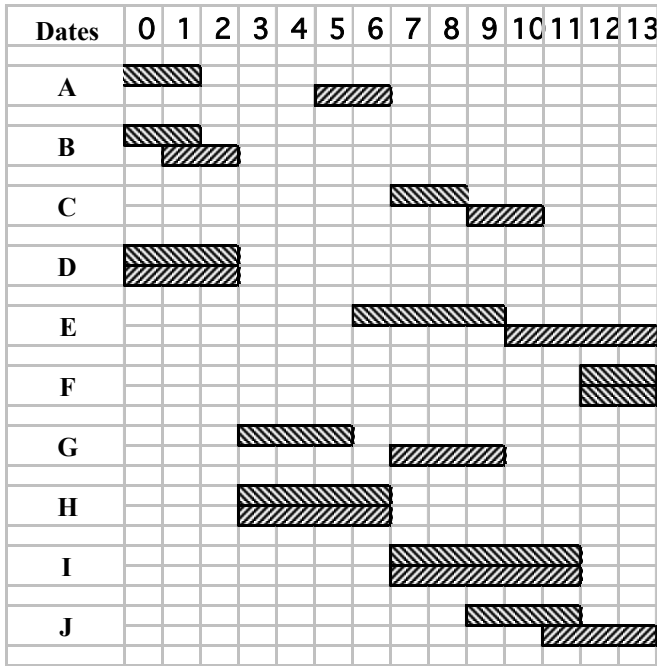


Figure 4.9. Gantt chart for the example

4.5. Probabilistic PERT

As projects by nature are not very repetitive, it is often impossible to give a precise execution time for the tasks. Those in charge prefer to give an estimation of the duration rather than be firmly engaged in a duration they are not sure of. The method used consists of asking for three estimations for each task:

- optimistic duration a : the best time that we could expect if everything went remarkably well (probability of 1%);
- most likely time n : the best estimate of the time required to accomplish a task, assuming everything proceeds as normal;
- pessimistic duration b , if everything went wrong (probability of 1%).

From these three pieces of data we will construct a statistical distribution that we will use in the following discussion. The distribution generally used is the beta distribution (the easiest triangular distribution to calculate can also work).

Beta distribution

This distribution is defined between the values a and b and has a maximum at n (in Figure 4.10). For a beta distribution, the expected mean duration and the variance are given by

$$\bar{d} = \frac{a + 4n + b}{6} \quad \sigma = \frac{(b - a)^2}{36}$$

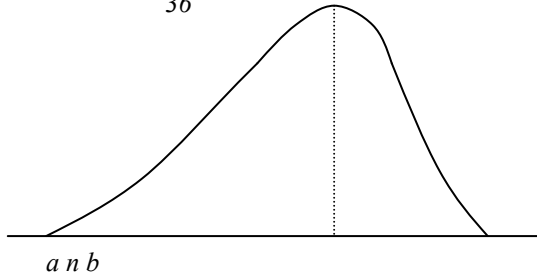


Figure 4.10. Beta distribution

Once the distributions of each task are determined, the solution of the problem can be done either in an analytic manner or by simulation.

4.5.1. Analytic solution

This approach is done in three steps:

- first step: for each task of the project, we use the mean duration. We then get back to a sequencing problem in a certain universe;
- second step: we calculate the corresponding schedule and determine the (or one) critical path CP ;
- third step: if the conditions of the central limit theorem are respected, then the duration of the project follows a normal distribution whose mean is equal to

$$\sum \bar{d}_i / t \in CP$$

and whose variance is equal to

$$\sum \sigma_i / t \in CP .$$

Without detailing the central limit theorem, let us say that two essential points must be verified so that the results obtained are correct:

- the distribution regulating the durations of tasks must be independent. This would not be the case if the optimism or pessimism of those in charge had an identical cause. Let us take the case of constructing a work of art in a foreign country. If for each person in charge, the optimistic or pessimistic duration is connected to the competence of the workers; the duration of the project will not follow a normal distribution, but rather a bimodal distribution. Either the workers are competent and all the durations will be shorter, or they are not be competent and the durations will be longer;

- the number of tasks of the critical path must be large (consisting in theory of about 30 critical tasks and of about 15 critical tasks for practitioners), which requires a large initial project.

EXAMPLE 4.2. We will not take up the previous example because it cannot respond to the criteria that we have just stated. Suppose that the critical path calculated on the mean durations were made up of the 15 tasks given in Table 4.5. The duration of the project follows a normal distribution with the mean being 256.5 days and a variance of 8.61 or a standard deviation of 2.94. This will enable us to answer a certain number of questions.

For a normal distribution, there is a 99.7% probability of deviating by less than three standard deviations from the mean. Here the variance is 8.61 and the standard deviation (square root of the variance) is 2.94. With a 99.7% probability, the project will be finished between 248 and 264 days.

Suppose that we were engaged to terminate in 255 days at most. To know the probability of keeping this engagement, let us first go back to a standard normal distribution (with a mean of zero and a variance of one): $z = (255 - 256.5) / 2.94 = -0.51$. By referring to a table of statistics, we find that the corresponding probability is 30.5%.

Task	<i>a</i>	<i>n</i>	<i>b</i>	Mean	Variance
1	7	10	14	15.25	1.36
2	6	8	9	11.75	0.25
3	12	15	18	22.50	1.00
4	7	10	12	14.75	0.69
5	9	11	13	16.50	0.44
6	13	14	17	21.50	0.44
7	15	17	20	25.75	0.69
8	6	8	9	11.75	0.25
9	8	11	13	16.25	0.69
10	4	7	8	10.00	0.44
11	9	12	13	17.50	0.44
12	7	9	10	13.25	0.25
13	11	13	16	19.75	0.69
14	12	15	17	22.25	0.69
15	10	12	13	17.75	0.25
Sum				256.50	8.61

Table 4.5. *Values of the tasks of the critical path*

4.5.2. Solution by simulation

The approach used here consists of simulating a large number of potential realizations of the project studied to learn as much as possible from it. To simplify, suppose that the estimated duration of task A is 8, 9 or 10 days with respective probabilities of 20%, 50% and 30%. If we make 100 potential realizations, in 20 of the realizations the duration of A should be 8 days, in 50, 9 days and in 30, 10 days. The usual random functions generate a random number between 0 and 1. To fix the duration of A, we will establish the following correspondence:

- if the random number is less than 0.2, the duration will be 8 days;
- if the random number is between 0.2 and 0.7, the duration will be 9 days;
- if the random number is greater than 0.7, the duration will be 10 days.

To simulate a potential realization, we will therefore take the duration of each of the tasks, respecting the distribution of probability, and calculate the dates inferred by these delays. But it is impossible to affirm anything after a single simulation. To verify that the dice are not loaded, only one throw is not enough. We have to throw a large number of times and verify that the sides 1, 2, 3, 4, 5 and 6 come up with noticeably the same frequency. We must also make a large number of simulations (at least 100). From these values we can calculate the mean and the standard deviation of these values; we can calculate the mean standard deviation of the earliest dates of each task and of the duration of the project. Counting the number of times where a task is critical lets us determine the probability of criticality of a task. According to the manner in which the durations will interact, the critical paths can be different.

The information obtained through this approach is richer than that provided by the analytic approach. The calculations of a PERT are very rapid (they are proportional to the number of liaisons between tasks). It seems to us that here the simulation should be preferred to the analytical method.

4.6. Sequencing with disjunctive constraints

The problems with resources are in general difficult (NP-complete), even for a unit quantity of each resource such as the presence of a single operator or a single machine (disjunctive problems treated here). There are thus no algorithms to find the optimal solution, no matter what the problem, with a reasonable calculation time.

A disjunctive constraint between A and B prohibits the simultaneous execution of A and B. We find such constraints when the resources considered are in unit quantities.

EXAMPLE 4.3. Let us take the usual example. Three workers are mobilized for this project. Table 4.6 gives for each task its duration and the necessary workers (•).

Task	Duration	Worker 1	Worker 2	Worker 3
Start	0			
A	2	•		
B	2	•		•
C	2		•	
D	3			•
E	4		•	
F	2			•
G	3	•	•	
H	4		•	
I	5	•		
J	3		•	•
End	0			

Table 4.6. *Disjunctive constraints in the example*

Let us consider worker 1. Its presence will create six disjunctive constraints between the tasks A, B, G and I. The constraints (B – I) and (A – G) are already arbitrated by the existing precedence constraints. The four constraints to conserve are: (B – A), (B – G), (A – I), (G – I).

This problem can be modeled in the form of a linear program with variables 0, 1. The tasks are renumbered from 1 (start task) to n (end task). We have two categories of decision variables:

- F_i the earliest end date of the task i ;
- x_{ij} bivalent variables defining the order of passage of the tasks i and j :
 - $x_{ij} = 1$ if i precedes j ,
 - $x_{ij} = 0$ if not.

The objective is to minimize the end date of the final task n : $\min \{F_n\}$.

We must introduce constraints between the end dates of the two tasks i and j that follow each other (type 1 constraint). For each precedence relation: if i precedes j we have $F_j \geq F_i + d_j$.

For each disjunction between i and j ($i < j$) we must also transform the following conditions (type 2 constraints):

- if i precedes j ($x_{ij} = 1$) then $F_j \geq F_i + d_j$;
- if j precedes i ($x_{ij} = 0$) then $F_i \geq F_j + d_i$.

This is realized by the two following linear equations where M is a large constant:

$$F_j - F_i + M(1 - x_{ij}) \geq d_j$$

$$F_i - F_j + Mx_{ij} \geq d_i$$

We must also fix the start date of the project, also called task 1. For this fictitious task, which has a zero duration, we will set $F_1 = 0$ to initialize.

In Example 4.3, we have 17 precedence constraints (corresponding to the arcs of the graph of precedence of Figure 4.5) and 12 disjunctions. The corresponding program has 12 type 1 equations and 12 pairs of type 2 inequations. The minimum required time to realize the project is 21. The Gantt chart in Figure 4.11 gives one of the solution schedules (another solution would, for example, begin with task B).

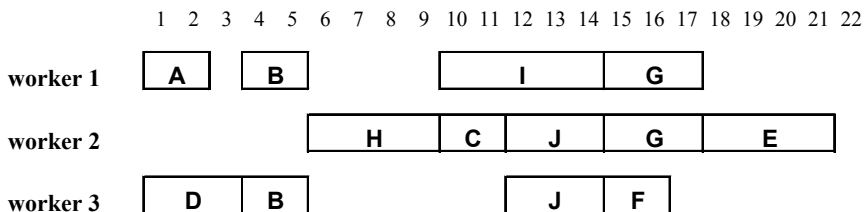


Figure 4.11. Gantt chart of the disjunctive example

The blocking parameter is the number of Boolean variables $\{0,1\}$. Commercial linear programming software is capable of solving, in an exact manner and in a reasonable time, problems in real numbers of very large size (several thousand variables and constraints). We are far from obtaining identical performance for the problems in integers or in Boolean variables $\{0,1\}$. Schematically, these problems are solved by branch and bound techniques and the calculation time becomes prohibitive if the number of integers or Boolean variables increases.

4.7. Sequencing with cumulative constraints: serial methods

For scheduling with resources, the heuristics usually used are called serial methods or lists of priority. On date t , a task is a candidate for placement if it has not yet been placed and if all its predecessors are finished. The principle of serial methods is the following:

Principle of serial methods

initialize the project ($t = 0$)

Repeat

- determine the candidate tasks,
- establish a list of priority among the candidate tasks,
- consider the candidate tasks one after the other in respecting the list of priority and
- assign them if the necessary resources are sufficient,
- go to the next point where a task is finished and where resources are freed,

Until all the tasks have been scheduled.

End

Algorithm 4.3. *Principle of serial methods*

EXAMPLE 4.4. Let us again take Example 4.1. We have four workers and two machines on the construction site. The necessary needs are given in Table 4.7.

Task	Workers	Machines
A	2	1
B	3	
C	2	
D	2	1
E	2	1
F	2	1
G	1	1
H	2	1
I	3	
J	1	1

Table 4.7. *Data for Example 4.4*

We will use the alphabetical order as a list of priority at each step.

At the beginning of the project the three candidate tasks are, in order of priority, A, B and D. We will try to place them by taking them in this order:

- we place A that takes two workers and one machine;
- B cannot be chosen because there are only two workers available;
- we place D that uses the two workers and the remaining machine;
- we will be at instant 2 where A ends and liberates two workers and a machine
- B is the sole candidate task, but cannot be chosen because there is still a worker missing;
- we go to instant 3 where D liberates two workers and a machine;
- the candidates that can be chosen are both B and G.

Table 4.8 summarizes the sets of the steps.

Date	Candidates	Chosen	Rejected
0	A, B, D	A, D	B
2	B		B
3	B, G	B, G	
5	H	H	
6	E	E	
9	C, I	C	I
11	I, J	I, J	
14			
16	F	F	

Table 4.8. Execution of the serial algorithm for Example 4.4

We thus obtain the Gantt chart in Figure 4.12.

The essential question is to choose an order of priority. The lists of priority generally used are obtained from the results of sequencing without resources. We can take the tasks:

- increasing by earliest dates (start or end);
- increasing by latest dates (start or end);
- increasing by total margins;
- decreasing by costs when a notion of cost of lateness exists.

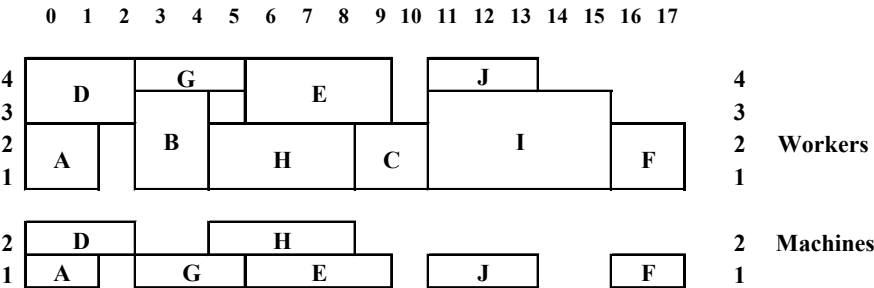


Figure 4.12. Gantt chart of the cumulative example

In addition, this order can be fixed *a priori* and once and for all, (as is the case in Example 4.3) or modified at each step to take into account the tasks already placed. The question we would like to answer is what the best criteria of priority are, or at least in which type of project one criterion can prove to be better than another. Here, we do not know how to reply to either question. In a given project, some tests carried out in varying the times of each task show that the criteria that gave the best result for a problem might give the worst results for the following problem. Statistically, the fact of taking the increasing total margins gives, in percentage terms, the best results (but this can be the least good result in certain cases).

Also, in practice, with the rapidity of calculations, we systematically test several criteria and we keep the best result. With the increasing power of computers, the tendency is to multiply the number of solutions explored. The simplest method consists of applying the serial method many times, choosing by chance the order in which we undertake the candidate tasks (random list of priority). We can also use metaheuristics based on neighborhoods of the tabu search, simulated annealing or genetic algorithms (see Chapter 3).

Advantages and disadvantages of serial methods

Serial methods are simple to use and their calculation time is very short. Fundamentally, these are methods of local optimization: we assign the resources each time that they are liberated. The result of this last point is that the use of resources is most often correct, but aberrations can appear. Let us consider the project with six tasks and four workers for which the graph is given in Figure 4.13.

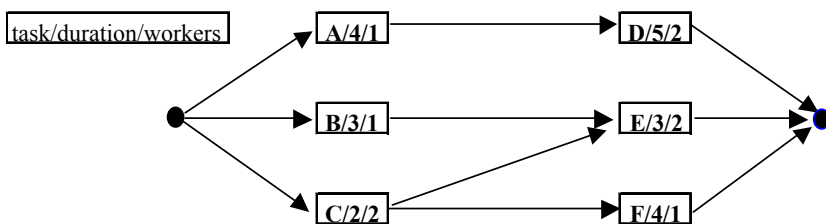


Figure 4.13. *Illustrative example*

Let us apply the serial method. At instant $t = 0$, we have three candidates A, B and C and enough workers for the three. C liberates two workers at 3 and we can begin F. B liberates one worker at 4 and E can begin. We must wait for day 7 to have two workers to begin D. We finish in 11 days. For this project, the solution

given by the serial method is unique and independent of the list of priorities (see Figure 4.14).

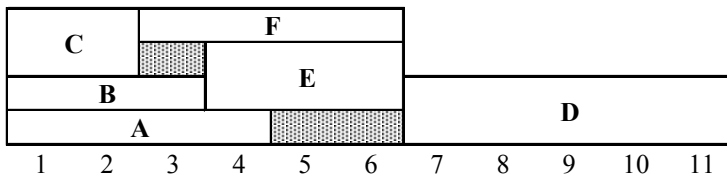


Figure 4.14. *Gantt chart of the serial solution*

However, this solution is not optimal. At instant 4, when B ends, we have two free workers and task E can begin. The serial method works like a good foreman: two workers are available and there is work to do; therefore we begin task E. The optimal solution (see Figure 4.15) would in fact consist of leaving the two workers inactive and waiting for day 5 in order to start D. We would finish at day 9. As we notice, to want to optimize the use of resources at every instant (local optimization) does not always lead to the global *optimum*.

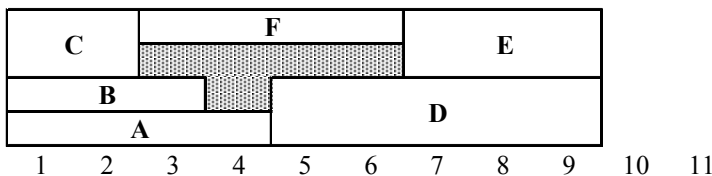


Figure 4.15. *Gantt chart of the optimal solution*

If we diminish the duration of a task or if we augment the number of resources, we very logically expect that the duration of the project will diminish or stay the same as before. The serial methods do not guarantee this. Let us take Example 4.3 and use 5 days as the duration of B. We obtain a solution in 9 days (see Figure 4.16) no matter which priority is chosen, while if B lasts 3 days, the solution is 11 days.

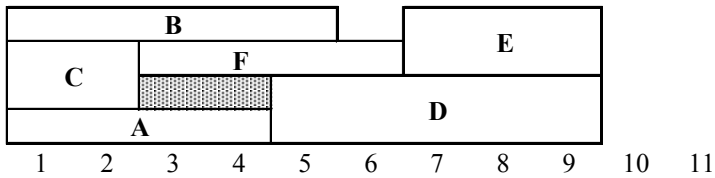


Figure 4.16. Gantt chart of the solution with duration $B = 5$

Let us now consider the small project in Figure 4.17 and let us take alphabetical order as the order of priority. If we have three workers, we need 7 days, and with four workers, we need 9.

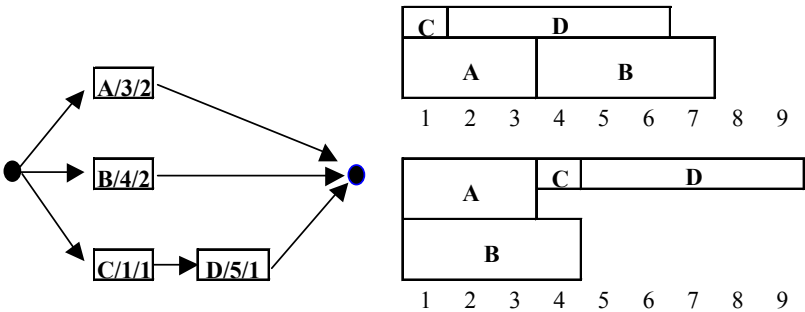


Figure 4.17. Instability of the results when the durations augment

4.8. Time-cost trade-off problem

Until now, we have not taken into account the possible interdependence between the duration of a task and its cost. For certain projects, it is possible to reduce the duration normally predicted for a task by assigning supplementary means to it. This reduction can only be made with certain limits. In general, it is more difficult and costly to gain the second day than the first, the third than the second, etc, which gives, between the cost and the duration, a decreasing relation as in Figure 4.18.

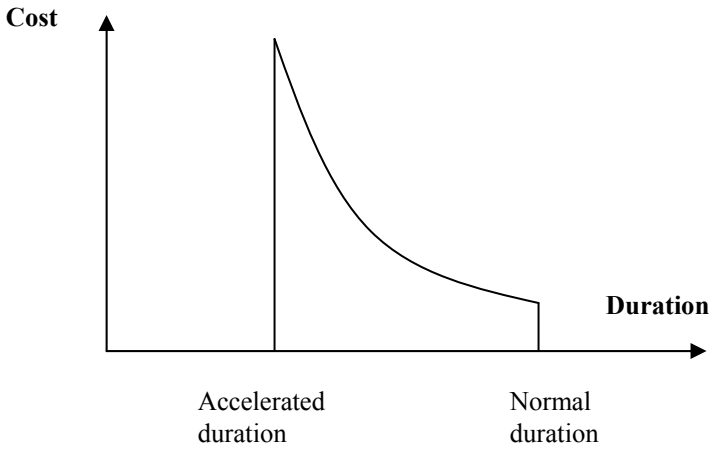


Figure 4.18. *Relation between cost and duration*

It is possible to solve this problem in an optimal manner. We will introduce this method in a small example that we will solve step by step.

EXAMPLE 4.5. The project is made up of seven tasks. We will, for once, represent the tasks in the form of an arc (see Figure 4.19).

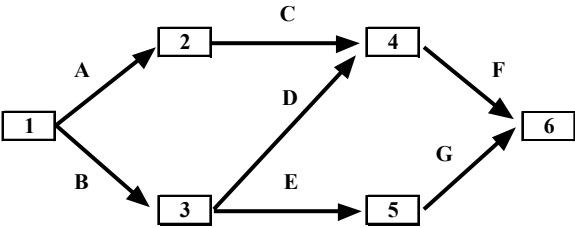


Figure 4.19. *Graph PERT/CPM for example 4.4*

We know the number of days normally predicted to execute a task and the shortest possible duration (accelerated duration). We will suppose here that the costs are proportional to the number of days gained (see Table 4.9).

Task	Arc	Normal duration	Accelerated duration	Cost by day gained
A	1, 2	3	2	5
B	1, 3	6	4	7
C	2, 4	4	2	4
D	3,4	4	2	2
E	3, 5	2	1	3
F	4, 6	6	5	3
G	5, 6	7	6	8

Table 4.9. Data for Example 4.4

Let us start the project with normal durations and with no extra cost (see Figure 4.20). This project is of duration 16 and the critical path (1, 3, 4, 6) is unique.

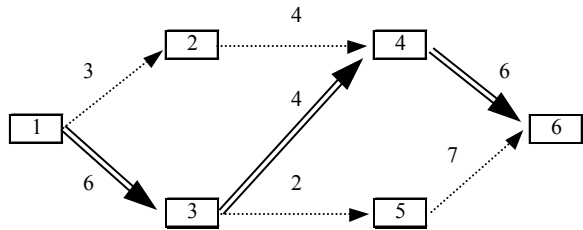


Figure 4.20. Results for duration 16

To reduce a non-critical task by one day will not reduce the duration of the project, even though it augments the cost. The choice is thus limited to (1, 3) of cost 7, (3, 4) of cost 2 or (4, 6) of cost 3. We reduce (3, 4) that has the lowest cost of a day. We now have the following project of duration 15 with a cost of 2 (see Figure 4.21).

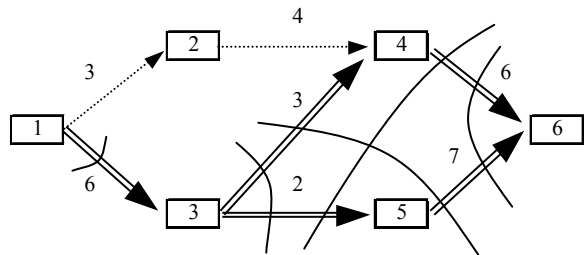


Figure 4.21. Results for duration 15

There are now two critical paths: (1, 3, 4, 6) and (1, 3, 5, 6). (3, 4) remains the task of least cost. But if we reduce only (3, 4), the path (1, 3, 5, 6) is still of duration 15 and we have not shortened the length of the project. In order to do that, we must cut the passage between 1 and 6. There are five cuts possible:

- (1, 3) of cost 7;
- (3, 4) and (3, 5) of cost $2 + 3 = 5$;
- (3, 4) and (5, 6) of cost $2 + 8 = 10$;
- (3, 5) and (4, 6) of cost $3 + 3 = 6$;
- (5, 6) and (4, 6) of cost $8 + 3 = 11$.

We choose the cut of minimum cost: (3, 4) and (3, 5). The project has a duration of 14 for a global cost of 7. We now have Figure 4.22.

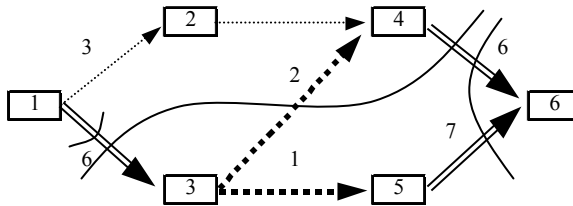


Figure 4.22. Results for duration 14

We have used heavy dotted lines for the arcs (3, 4) and (3, 5) that can no longer be reduced. To descend to 13 days, there are 3 possible cuts, two that existed before (1, 3) of cost 7 and (5, 6) and (4, 6) of cost 11. Let us look at the last cut that passes by (1, 3), (3, 4) and (4, 6). We can no longer reduce (3, 4), but to reduce (1, 3) and (4, 6) is sufficient to cut all the critical paths for a cost of 10. However, in this case, the path (1, 3, 4, 6) would be of length 12 ($5 + 2 + 5$). We can thus increase (3, 4) by one day, which would allow us to gain 2. The cost of the cut (1, 3), (3, 4) and (4, 6) is thus 8.

The *optimum* nevertheless remains the cut (1, 3) of cost 7 and the project has a duration of 13 days for a global cost of 14. We leave it to the reader to find the best cut to descend to 12 days (global cost of 22) and 11 days (global cost of 36). Having arrived at 11 days, there is a path entirely constituted of arcs with minimum duration (see Figure 4.23). It is no longer possible to reduce the duration of the project.

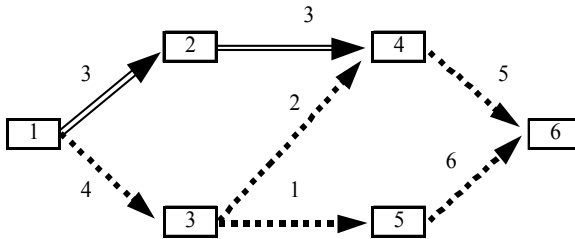


Figure 4.23. Results for minimum duration 11

The solution of PERT-cost is connected to the search for minimum cuts in the graph of critical tasks. The problem of the minimum cut is equivalent to the problem of maximum flow (Ford-Fulkerson's theorem). The possibility of increasing tasks previously reduced presents here a slight change in the classical algorithm of Ford and Fulkerson (see pages 151–162 of [FOR 62]). If we trace the cost/duration function of the project, we notice in this example that the function obtained is decreasingly convex. This property is true in the general case if the cost/duration functions of each task are decreasingly convex.

4.9. Conclusion

In this chapter we have presented the classical methods used in project scheduling. However, there remains work to do in this domain; for example, to ameliorate frequent constraints of the daily workload in construction (problem of equalizing the workloads from one day to the next). Let us also remember that project management is not only limited to its technical aspects, but it remains a management problem.

4.10. Bibliography

- [AFI 91] AFITEP, *Le management de projet: principes et pratiques*, AFNOR Gestion, 1991.
- [CHV 94] I. Chvidchenko and J. Chevallier, *Conduite et gestion de projets*, Cépaduès Editions, 1994.
- [DUP 98] L. Dupont, *La gestion industrielle*, Hermès, Paris, 1998.
- [FOR 77] L.R. Ford and D.R. Fulkerson, *Flow in Networks*, Princeton University Press, New Jersey, 1977.

[GIA 91] V. Giard, *Gestion de projets*, Economica, Paris, 1991.

[MAU 77] E. Maurel, D. Roux, and D. Dupont, *Techniques opérationnelles d'ordonnancement*, Collection des ouvrages de la Direction des Etudes et Recherches d'EDF, 1977.

Chapter 5

Operations Management in Transportation Networks

5.1. Introduction

5.1.1. *A bit of history*

Until the end of the 1970s, most researchers used approximate methods to solve the large-scale problems encountered in industrial transportation applications. By reading the survey article [BOD 83], we can appreciate the main concerns of this research area and notice the variety of mathematical models and the computational methods then used to solve them. At the beginning of the 1980s, a new class of problems became apparent, taking into consideration *time windows* that limit the beginning or the end of various activities. Our research thus led us to solve a school bus routing problem where, in the morning, for example, the school children had to arrive at school inside of a time interval before school started. This rather simple management problem enabled us to point out certain weaknesses in the current optimization methods. At the pace the scientific community was progressing, beginning with simplified cases and adding the difficulties one by one, it would have taken a century before solving to optimality the large industrial applications.

We took an original research direction, that of developing a solution approach taking advantage of the many constraints of real problems rather than trying to use ideas that are only efficient for problems with few constraints. The importance of

this research work was rapidly recognized, as we can see in the success of the methodology applied to school transportation [DES 84]. Thereafter, this approach was used to schedule bus drivers [DES 89] and to prepare itineraries for vehicles transporting handicapped people [DES 88c]. At the end of the 1980s, the first commercial version of the optimization software system GENCOL was available for these three applications. Later, the research was oriented toward other very important applications, notably in air and rail transportation.

The acronym GENCOL comes from the French expression GÉNÉration de COLonnes (that is, *column generation*), the optimization method used by researchers to solve many vehicle routing and crew scheduling problems. In the following pages, we describe the main aspects of this method and also look at some modeling considerations encountered in several contexts of operations management in large transportation networks.

5.1.2. *University–industry: a winning partnership*

Operations research is built on a balanced mix of mathematics and computer science applied to management, and several members of the scientific community are famous for the quality of their accomplishments, as well as for fundamental discoveries for the commercialization of optimization software systems. The research carried out for more than 25 years has thus enabled us to identify a common structure for vehicle routing and crew scheduling problems [SOL 88, DES 95, DES 98a], with numerous applications in the domain of urban, air and rail transportation. With this harmonious marriage of mathematics and computer science, it is now possible to solve many routing and scheduling problems. The diversity of applications, each with its own difficulties, is thus an open door to the future of operations research as it enables the development of many innovative optimization software systems.

Research is expensive and it is necessary to reach a critical mass to succeed in achieving the first rank. It is incontestable that the universities and government institutions have favored, and must still favor, the realization of large projects in the domain of transportation, the former to create large research centres, the latter to support their infrastructure. These large-scale projects consolidate university training in graduate schools, generate hundreds of jobs and increasingly accentuate the international leadership of commercial and industrial partners.

The role of these industrial partners is very important. Indeed, they often determine the relevance of the problems studied and maintain adequate industrial transfers of the new technologies. Therefore, young enterprises can benefit from exceptional research results, their technological advantage allowing them to start

and grow in markets that are sometimes already occupied by mature products developed by the world giants of computer science.

5.2. Fundamental notions

5.2.1. A common structure

A fundamental result of the work previously mentioned has been to identify a generic problem appearing in many applications [SOL 88, DES 95, DES 98a]: that is, to cover at the least cost a set of *tasks* by choosing in a graph a subset of *paths* satisfying the *operating constraints*.

This generic problem appears in *vehicle routing* problems. Tasks represent bus trips, plane flights, train segments, etc. Paths represent the itineraries that one must determine to do all these tasks. Operating constraints concentrate, for example, on the capacity of the equipment, the kilometers or the maximum duration before the maintenance, etc. They can also impose intervals or time windows during which the programmed tasks must begin.

The generic problem also appears in *crew scheduling* problems. Tasks then represent bus trips, plane flights and train segments between places where one can replace the crews. A path represents the sequence of tasks performed by a crew. For bus drivers, it is a question of workdays; for plane and train crews, it is a question of rotations leaving from a city, working for several days broken up by rest periods, and returning to the same city. Operating constraints on paths can be the minimum and maximum durations of rest and of workdays, as well as other rules arising from the labor agreements with the employees. Finally the establishment of *monthly schedules* occurs. In this case, tasks are workdays for the bus drivers or crew rotations in air transportation that it is now necessary to assign to the employees. A path is then defined by the sequence of tasks, days off and training days of an employee during the selected month. Operating constraints on these paths are the rules of the labor agreements, for example on the number, duration, and schedule of days off, on the maximum and minimum worked time, etc.

EXAMPLE 5.1. Let us consider the problem of assigning planes to flights so as to maximize the profits for the month to come. With several plane types to choose from, the largest planes would usually be assigned to the flights where the demand is the greatest, while the smallest planes would be assigned to flights with small demand. Cost often takes negative values, a result of the expenses incurred minus the revenues, a calculation based on the capacity of the plane used, on an estimate of the prices and on the number of tickets sold, etc. In addition, we suppose that the exact flight departure times have not yet been completely determined, but are rather

given in time windows of, for example, 30-minute duration. These time intervals leave flexibility in the optimization process.

Based on the current location of the planes, we construct itineraries aimed at covering each flight exactly once, while determining the departure times. For a given plane we must add the waiting time on the ground to the flight duration (a function of the type used) so as to validate its itinerary by verifying the departure time of each planned flight. If we take maintenance into account as well, we also keep track of the total flight time that must remain well within a threshold value, as required by the security rules; this flight time is set back to zero once the maintenance has been completed.

The notion of a path under operating constraints is thus omnipresent in all the applications mentioned above. If we can construct all the paths that satisfy the operating rules, the only thing that remains is to choose a certain subset of these to cover the programmed tasks. We present in the following section the first problem of this type found in the literature. It is a relatively simple case only involving a cost component and a single time characteristic. It is however at the origin of the development of optimization methods now used in this vast domain of applications.

5.2.2. *The shortest path problem with time windows*

The *shortest path problem with time windows* aims to determine the least costly path between two given vertices in a space-time network while respecting the time intervals associated with the visited vertices [DES 83]. Two components are present on each arc of the graph: cost and duration. By cumulating the durations, we can validate the paths by verifying if the time component satisfies the time interval imposed on each visited vertex.

When we consider only the arc cost, the shortest path between a source vertex and a destination vertex is relatively simple. To solve this problem, there are several iterative algorithms that all possess the following fundamental property: in each vertex, we keep only one cost label, the best one that gives an overestimation of the total cost from the source. This label is updated each time that it is improved. The left side of Figure 5.1 illustrates this aspect: four values are calculated for the lower vertex, but only that of the least cost (37) is retained. It is quite different in the presence of a supplementary time component. The elimination of a two-dimensional label by another is only possible if both the time and cost components of the best label are smaller than or equal to the components of the eliminated one. Of the four labels calculated on the right side of Figure 5.1, only one is eliminated. For example, the first and the last labels are not comparable. Indeed the last label has the lowest cost but the possibilities of extending the associated partial path are limited

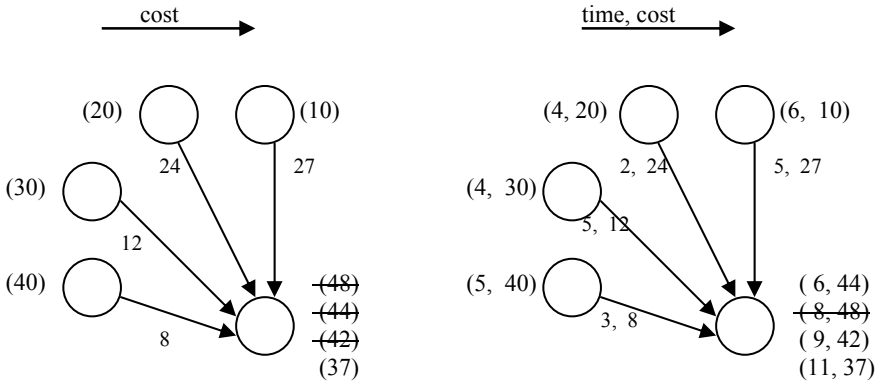


Figure 5.1. Labels at a vertex

by the high value of the time component. However, it would be easier to add new vertices to the path associated with the first label, thus increasing the chances of finding a better path.

Since they satisfy the time interval associated with a vertex, the labels retained constitute a set of *efficient labels*. The reader will notice that if the efficient labels are sorted in increasing order of time, the cost component appears in decreasing order. The solution algorithms exploit this important property.

5.2.3. Some mathematics

For the shortest path problem with time windows, let us consider $G = (V; A)$, a directed graph where A is the arc set while V indicates the vertex set. Moreover we have $V = N \cup \{o, d\}$, with N being the set of possible visited vertices on a path from the source o to the destination d . A time window $[a_i, b_i]$, $i \in V$ is associated with each of the vertices. A path in G is defined as a succession of vertices of V , connected two by two by an arc of A . All the paths begin at time a_o at the source vertex o and end at the destination vertex d at the latest at b_d . If the arrival time at a vertex i is smaller than the lower bound a_i , a waiting time is necessary, and this without penalty. With each arc $(i, j) \in A$ is an associated cost c_{ij} and a positive duration t_{ij} . An arc (i, j) is only defined in A if $a_i + t_{ij} \leq b_j$. According to the context, other conditions for the validation of the arcs can be imposed; we think, for example, of the connection between two flights, the departure airport of a flight being necessarily the arrival airport of the preceding flight.

A binary *flow variable* X_{ij} is defined on arc $(i, j) \in A$ and it takes value 1 if the optimal path goes through that arc, and 0 otherwise. A *time variable* T_i is defined on each vertex, in the interval $[a_i, b_i]$, $i \in V$. We suppose that the service time at vertex i is included in value t_{ij} , $(i, j) \in A$. A time variable thus represents the start of service at a vertex. With this notation, the mathematical formulation is the following:

$$\begin{aligned}
 & \text{Minimize} \quad \sum_{(i,j) \in A} c_{ij} X_{ij} \\
 & \text{s.t.} \\
 & \sum_{j \in V} X_{ij} - \sum_{j \in V} X_{ji} = \begin{cases} +1, & i = o \\ 0, & \forall i \in N \\ -1, & i = d \end{cases} \quad [5.1] \\
 & X_{ij} \text{ binary, } \forall (i, j) \in A \\
 & a_i \leq T_i \leq b_i, \quad \forall i \in V \\
 & X_{ij}(T_i + t_{ij} - T_j) \leq 0, \quad \forall (i, j) \in A
 \end{aligned}$$

The objective function minimizes the cost of a path that starts at vertex o and ends at vertex d . For each of the other vertices, we have a flow conservation constraint. A binary flow variable is defined for each arc while a time variable is defined for each vertex. The nonlinear relations establish the link between the flow and the time variables: if $X_{ij} = 1$, then $T_j \geq T_i + t_{ij}$. Since $T_j \geq a_j$, we have that $\max\{a_j, T_i + t_{ij}\} \leq T_j \leq b_j$, indicating the possibility of a waiting time if $T_i + t_{ij} < a_j$.

5.2.4. A generic algorithm

Several dynamic programming algorithms have been developed to solve the shortest path problem with time windows. Without going into all details, we present some important aspects.

We have already mentioned that for each vertex, we keep the set of efficient labels, that is the set of (time, cost)-vectors that do not dominate each other. In fact, for two labels (T_i^1, C_i^1) and (T_i^2, C_i^2) corresponding to two different paths leaving from source o and arriving at vertex i , the first label dominates (and eliminates) the second if and only if $(T_i^1, C_i^1) \leq (T_i^2, C_i^2)$. We denote by *Elim* this operation of elimination of the dominated labels. Let us define at vertex i , $\Gamma(i) = \{j: (i, j) \in A\}$ the set of successor vertices and $Q_i = \bigcup_e \{(T_i^e, C_i^e)\}$ the set of the current labels. The basic operation of dynamic programming algorithms applied to the shortest path problem with time windows is to transfer the vectors kept at vertex i to all the

successors of this vertex. For label (T_i^e, C_i^e) and arc $(i, j) \in A$, the new label created in vertex j is given by the following transformation:

$$f_{ij}(T_i^e, C_i^e) = \begin{cases} (\max\{a_j, T_i^e + t_{ij}\}, C_i^e + c_{ij}), & \text{if } T_i^e + t_{ij} \leq b_j \\ \phi, & \text{otherwise.} \end{cases}$$

Transformation f_{ij} , an *extension function*, creates a label from i to j if the time component does not exceed the upper bound b_j of the time window. This transformation must be applied to all the labels of Q_i . At vertex j , the new labels are given by $\bigcup_e f_{ij}(T_i^e, C_i^e)$.

Not all these labels are efficient, notably because those for which $T_i^e + t_{ij} < a_j$ necessitate a waiting time take the same time value a_j but with different costs. In addition, among the new labels, some can dominate or be dominated by the labels already present in Q_j . The new set of efficient labels at vertex j results from the application of the *Elim* operator to the set $\bigcup_e f_{ij}(T_i^e, C_i^e) \cup Q_j$. The treatment of a vertex i ends when all the successors $j \in \Gamma(i)$ of this vertex have been treated. The generic algorithm presented below (Algorithm 5.1) uses *LIST*, the set of vertices that must be considered again. In such a vertex, there is a modification of the set of efficient labels.

STEP 0	$Q_o = \{(T_o^1 = a_o, C_o^1 = 0)\}$ $Q_i = \{(T_i^1 = a_i, C_o^1 = \infty)\} \quad \forall i \in V \setminus \{o\}$ $LIST = \{o\}$
STEP 1	Choose a vertex $i \in LIST$ For all $j \in \Gamma(i)$ do $Q = Elim(\bigcup_e F_{ij}(T_i^e, C_i^e) \cup Q_j)$ if $Q_j \neq Q$ then $Q_j = Q$ and $LIST := LIST \cup \{j\}$
STEP 2	$LIST := LIST \setminus \{i\}$ if $LIST = \emptyset$ then STOP otherwise, go to STEP 1

Algorithm 5.1. *A generic dynamic programming algorithm for the shortest path problem with time windows*

With this generic algorithm we end our brief glimpse of fundamental notions. The interested reader can look at specialized articles on the subject that, of course, build on diverse properties of the shortest path problem with time windows [DES 86, DES 88a, DES 88b]. The most important is certainly the fact that the time component must always be non-decreasing, which enables us to immediately do a chronological treatment of the labels rather than the treatment of the vertices. Thus, the labels can still be ordered according to the time, which ensures the convergence of the algorithms presented in the literature, even in the presence of circuits of negative cost. This property induces the construction of an acyclic graph where the vertices are copies of the original vertices at different times, that is, all those that are possible in the given time interval. We come back later to the generalizations of this key problem. In particular, we consider the extensions to several dimensions as well as to other nonlinear extensions functions that can model many practical situations.

5.3. A mechanism of decomposition

5.3.1. Local restrictions and global constraints

The methodology described here also applies to problems encountered in strategic planning of vehicle routing and crew scheduling, as well as to the daily management of operations. These combinatorial problems deal with a great number of itineraries or schedules, the possible choices being given by binary decision variables. The models also include nonlinear constraints; we recall, in the shortest path problem with time windows, the constraints expressing the compatibility between the flow and the time variables, or again, the function $\max\{a_j, T_i + t_{ij}\}$ determining the start of service at vertex j coming from i . Economic issues may even result in discontinuous cost functions. For example, Figure 5.2 illustrates a minimum guarantee of 3 hours of work, then a regular hourly rate up to 7 hours followed by a bonus and a new higher hourly rate for the overtime work.

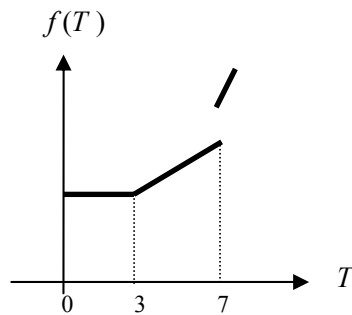


Figure 5.2. A discontinuous cost function

We now examine certain mathematical concepts capable of dealing with these complex but realistic aspects of applications encountered in operations management in transportation networks. The generic problem consists of covering at the least cost a set of tasks by choosing in a graph a subset of paths satisfying the operating constraints. Its solution is represented by paths where each corresponds either to the itinerary of a vehicle, or to the schedule of a crew member. To solve it, we break it into two parts: one part is in charge of all the *local restrictions*, that is, specific to a single path; the other part selects among the possible paths those that satisfy the *global constraints* for the covering of the tasks.

Examples of local restrictions on a single path are numerous and we have already evoked several: time window for the beginning of service of a client, vehicle capacity, minimum and maximum work time during a workday or a rotation, etc.

Most of the local restrictions are described with the help of extension functions. When these are non-decreasing, it is easy to manage them using algorithms similar to that of the shortest path problem with time windows [DES 98a]. If the cost function possesses the same properties, it is equally easy to treat it in an optimal fashion.

Global constraints ensure firstly the covering of the tasks. For example, we must use a single plane but several crew members to cover a Montreal-Paris flight. Other examples concern the number of planes of each type, the number of full-time and part-time workers, the number of work hours allotted to the personnel of a crew, etc.

The advantage of the proposed decomposition is that it keeps all the nonlinear aspects of the generic problem at the level of the local restrictions [DES 95, DES 98a]. This decomposition is only valid if the cost of a path is independent of the cost of the other paths. In this case, even if the cost of a path is a nonlinear function, once calculated it becomes a scalar, as illustrated in Figure 5.3. In this figure, 10 tasks must be performed (covered), twice for tasks 2 and 5 while all the others are only covered once. Each column corresponds to an admissible schedule whose cost is given in the first row. In a column, the value 1 appears for every covered task. We could interpret the last four lines as follows: we must select at most 2 schedules of the first type consisting of at least 25 hours of work, and at most 4 schedules of the second type totaling up to 40 hours of work.

From the matrix structure in Figure 5.3, we can consider a formulation for our generic optimization problem [DES 95, DES 98a]. Let K be the set of the types of paths, one type per crew member, vehicle or similar group. For $k \in K$, matrix B_k represents the admissible paths (itineraries or schedules), while D_k gives the contributions to the other global constraints, Y_k is the vector of path variables, and c_k is the corresponding cost vector.

		Costs																	
		39	43	42	40	38	38	...	43	44	34	41	34	33	38	58	41		
Tasks	1	1						...	1		1				1		...	= 1	
	2		1	1			1	...		1	1		1		1	1	...	= 2	
	3	1	1		1			...	1			1		1		1	...	= 1	
	4		1	1		1	1	...		1	1			1	1	1	...	= 1	
	5			1		1		...	1	1			1			1	...	= 2	
	6			1	1		1	...				1			1	1	1	...	= 1
	7				1			...	1		1		1					...	= 1
	8	1	1			1		...		1				1		1	1	...	= 1
	9					1	1	...	1			1	1		1	1		...	= 1
	10	1	1		1			...		1	1					1		...	= 1
		13	17	15	11	13	14	≥ 25
									12	18	13	12	9	10	14	24	13	...	≤ 40
		1	1	1	1	1	1	≤ 2
									1	1	1	1	1	1	1	1	1	...	≤ 4

Figure 5.3. Matrix structure of global constraints

The cover of the tasks is expressed by vector b ; the demand for the other global constraints by vector d . Using when needed slack and surplus variables to obtain constraints expressed in the form of equalities, the problem of covering a set of tasks at the least cost by choosing in a graph a subset of paths satisfying the operating constraints becomes:

$$\begin{aligned}
 & \text{Minimize } \sum_{k \in K} c_k Y_k \\
 & \text{s.t.} \\
 & \sum_{k \in K} B_k Y_k = b \\
 & \sum_{k \in K} D_k Y_k = d \\
 & Y_k \geq 0, \text{ integer}, \forall k \in K
 \end{aligned} \tag{5.2}$$

The preceding problem is a linear program with integer variables. To solve it, we go first through the solution of its linear relaxation; we remove the integrality restrictions on the decision variables and thus obtain a lower bound for the optimal value of the cost function. In a progressive branch and bound tree, additional constraints are then added so as to determine an optimal solution satisfying also the integrality of the decision variables. The number of admissible paths is generally too large for it to even be conceivable to efficiently solve the linear relaxation of [5.2]. In this case, we call upon an optimization technique called the column generation method; we describe it in the following sections. The reader who is less interested in the technical aspects of this solution method can go on to section 5.4.

5.3.2. The column generation method

The *column generation method* first considers the linear relaxation of problem [5.2] and separates it into two parts: a master problem and some sub-problems. The *master problem* coordinates the global or coupling constraints; it selects a subset of minimum cost paths to cover all the tasks. The *sub-problems*, one by type of vehicle, of crew member or of a similar group, generate, as needed, the paths satisfying the local restrictions. The optimal solution is obtained by solving alternatively the master problem and the sub-problems; by solving the master problem with the current paths, we obtain the best current solution, then we generate, by solving the sub-problems, new paths that enable us to eventually improve this solution of the master problem.

This approach treats most of the complex constraints (for example, labor agreements) at the level of the sub-problems. The master problem that contains only linear constraints and a linear cost function can be efficiently solved with the simplex algorithm. For the sub-problems, it was necessary to develop a whole family of dynamic programming algorithms, solving in integer numbers shortest path problems with nonlinear constraints and cost functions, which are not necessarily convex [IRN 05]. Without all the details of the mathematical aspects, it is impossible to correctly present the column generation method. However we will bring out the essential points and refer the interested reader to more specialized works.

We call a *restricted master problem* a master problem that is formed by a subset of all admissible paths. The solution of this problem supplies essentially three types of information: on the one hand, the value of the cost function and that of the current variables Y_k , and, on the other hand, the vectors of the dual variables π_b and π_d associated with the constraints of formulation [5.2]. In the simplex algorithm to solve linear programs, the choice of a new variable entering the basis is made from the calculation of the reduced cost of the variables. The basic variables have a zero

reduced cost value. If the reduced cost of a non-basic variable is negative, it can enter the basis; otherwise the current solution is optimal. In our context, the vector of reduced costs associated with type k is given by the expression:

$$c_k^\pi = c_k - (\pi_b B_k + \pi_d D_k).$$

We use exactly the same kind of information to evaluate the paths that are not yet present in the restricted master problem, but that could eventually improve the solution. Knowing the structure of the paths that we must construct, we have to solve an optimization problem that is a constrained shortest path problem, at the same time integrating the local restrictions and the current dual vectors. We have to solve a shortest path problem similar to the one presented in formulation [5.1] with the local restrictions introduced in the form of multidimensional labels while incorporating in the cost function the dual information (π_b, π_d) . For type k , the sub-problem is defined on the graph $G_k = (V_k; A_k)$, and the new cost function to minimize takes the form

$$\sum_{(i,j) \in A_k} c_{ijk}^\pi X_{ijk},$$

where, as before, X_{ijk} is a binary variable, while c_{ijk}^π adequately integrates into the arcs of graph G_k the dual information of the covering and the other global constraints. G_k is in fact a space-time network where the arcs represent various activities (such as task, waiting, rest, empty moving, etc) while the vertices are places at specific times (for example, airport, railroad station, storage area, etc).

To calculate the reduced cost c_{ijk}^π of an arc, we subtract from its original cost c_{ijk} the corresponding dual variables π_b and π_d , these last variables being multiplied by the contribution of this arc to the covering constraints in B_k and to the other global constraints in D_k .

If there is a sub-problem for which the optimal value of the modified cost function is negative, the path found can improve the solution of the current restricted master problem and the column corresponding to this non-basic variable is added to the matrix structure. Otherwise, when the value of the cost function of all the sub-problems is zero, the current solution to the restricted master problem is optimal for the linear relaxation of the master problem and the process of column generation ends. The solution of the sub-problems thus determines the detail of the

paths so as to construct matrices B_k and D_k of formulation [5.2]. It also provides the stopping criterion for the column generation method.

5.3.3. *Solutions satisfying the integrality constraints*

We have already mentioned that to obtain the solution in integer numbers for problem [5.2], it is necessary to develop progressive branch and bound methods, that is, branching decisions and cutting planes compatible with the column generation method [DES 95, DES 98a]. These mathematical developments have in fact solved an open problem known since the beginning of the 1960s when the column generation method had begun to be used in certain optimization problems, but techniques to obtain integer solutions were not yet well developed.

The main difficulty is the following. Let us suppose that the decision variables Y_k of problem [5.2] only take binary values: 1 if the corresponding path is used in the solution, and 0 otherwise. When the solution of the linear relaxation to the master problem is fractional, it is easy to fix by branching such a variable to 1. In fact, it is a matter of subtracting from the vectors b and d the contribution of this column and solving a new problem that is smaller than the preceding one. However, there is no simple way to fix a decision variable to 0. We cannot simply remove this column from the restricted master problem since, as it is part of the optimal solution of the linear relaxation, it will again be generated by one of the sub-problems.

The trick is to rewrite the generic problem [5.2] as a multi-commodity flow model, nonlinear and integer in G_k , $k \in K$, with coupling constraints and resource variables [VIL 05]. Each commodity $k \in K$ represents a type of path, by crew member or by vehicle. The covering constraints of the tasks are directly taken into account by the structure of the flow constraints on the arcs comprising the tasks while the coupling constraints determine the other global constraints. In addition, resource variables, such as the time worked, the load of a vehicle, the mileage since the last maintenance, etc, serve to model the local restrictions on a single path. They are accumulated on the arcs of a path, verified and updated on the vertices, in practice by extension functions that are often nonlinear.

We exploit the structure of this new model with a mathematical method: the Dantzig-Wolfe decomposition [DAN 60] developed at the beginning of the 1960s. This method explicitly creates the same structures of master problem and sub-problems introduced in the approach by column generation. Even if we add new constraints arising from branching decisions and cutting planes, this decomposition mechanism remains valid. These new constraints either appear in the structure of the global constraints D_k , $k \in K$ and the vector d , or they simply modify the shortest

path sub-problems. Branching decisions can be taken on the binary flow variables X_{ijk} or on the resource variables. Thus, we can, for example, fix a flow variable at 0 or 1, use the division of the time intervals, or utilize well-known constraints from the literature to determine integer solutions. For all nodes of the branch and bound search tree, the decomposition principle is applied each time in order to solve the linear relaxation of the new master problem. The solution of the master problem, given in terms of the path variables Y_k , fractional or not, is then carried forward in the formulation of the multi-commodity flow model to verify if the variables X_{ijk} are integer numbers or not. In practice, each application requires adapted strategies of branching and cutting that exploit its characteristics. Nevertheless, the theoretical difficulties raised by the column generation method applied to programs in integer numbers are completely set aside [LUB 05].

5.4. Diversity of the local restrictions

5.4.1. A few words on the extension functions

The solution approach by decomposition takes advantage of the constraints of real problems by keeping the local restrictions at the level of the construction of the feasible paths. As for the shortest path problem with time windows, these local restrictions are very often expressed with the help of resource variables cumulated on the arcs of a path, verified and updated on the vertices, in practice by nonlinear extension functions. Despite the diversity of the local restrictions, we have already mentioned that most of the extension functions from one vertex to another are non-decreasing functions. We now look more closely at this aspect.

In formulation [5.1] describing the shortest path problem with time windows, let us replace the last set of constraints

$$X_{ij}(T_i + t_{ij} - T_j) \leq 0, \quad \forall (i, j) \in A$$

with a more general form [DES 98a]:

$$X_{ij}(f_{ij}(T_i) - T_j) \leq 0, \quad \forall (i, j) \in A.$$

If variable X_{ij} , $(i, j) \in A$ takes value 1 on a path, we have $T_j \geq f_{ij}(T_i)$, that is, the value of resource T_j evaluated from vertex i to vertex j is a function of the resource variable T_i of the preceding vertex, given by $f_{ij}(T_i)$.

Figure 5.4 presents some of the most common extension functions. In the upper left part, $f_{ij}(T_i) = T_i + t_{ij}$ is the usual function for the total cost or the accumulated

transported load in a vehicle; more generally, it is the extension function utilized when there is no lower bound on the value of a linear resource. In the upper right part, we again find the piecewise linear function $f_{ij}(T_i) = \max\{a_j, T_i + t_{ij}\}$ that we have already analyzed during the study of the shortest path problem with time windows. This function is also used, but only at the destination vertex d , if the cost function imposes the payment for a minimum work time.

In the lower left part, the strictly increasing function $f_{ij}(T_i) = T_i - t_{ij}$ allows the value of a resource to be decreased. We can thus impose restrictions on the remaining capacity of a vehicle rather than on the load that it transports. For example, this function is used when the lower bound of a time window must be satisfied in a strict manner, without waiting. The restriction at vertex j given by $a_j \leq T_j \leq b_j$ is then treated with two resources and two extension functions. For the upper bound $T_j \leq b_j$, we use function $f_{ij}(T_i) = T_i + t_{ij}$. By setting $NegT_j = -T_j$ for each vertex $j \in N$, the lower bound $-T_j \leq -a_j$ becomes $NegT_j \leq -a_j$ and this time we use function $f_{ij}(NegT_i) = NegT_i - t_{ij}$. An example of application concerns the pilots of certain airlines to which we must attribute a minimum of 72 and a maximum of 78 flight hours per month.

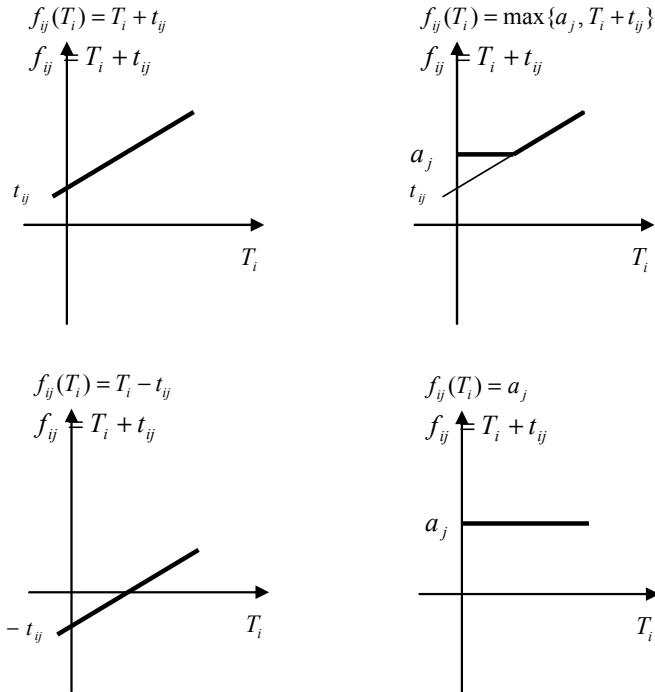


Figure 5.4. Some non-decreasing extension functions $f_{ij}(T_i)$

In the lower right part of Figure 5.4, we find the constant function $f_{ij}(T_i) = a_j$. If $a_j = 0$ for example, T_j can represent a counter adding up the number of flights made each day by a pilot, a counter that must be set back to zero at the beginning of each day, for example by using an arc whose associated activity takes place during the night. Similarly, the resource variable counting the time worked by a pilot is reinitialized at 60 minutes at the beginning of a workday; this value corresponds to the briefing time before the first flight.

Finally, we recall that the composition of non-decreasing functions is also a non-decreasing function. This important property allows us to model numerous practical situations even if the current functions are neither convex nor continuous. We can in this way combine the restrictions imposed by a sequence of several arcs with the help of a single function.

EXAMPLE 5.2. Let us consider Figure 5.5 where a resource T cumulates the work time of a pilot, this being limited to 7 hours a day (420 minutes). At the end of the last flight represented by vertex i , in order for the arc (i, j) to be valid, it is necessary that the cumulative time T_i plus the time for debriefing (30 minutes) is less than 7 hours, that is $f_{ij}(T_i) = T_i + 30 \leq 420$. During the night period on the arc (j, k) , the resource is set back to zero while it is reinitialized at 60 minutes (the duration of the briefing) before the first flight of the following day. These three functions on three successive arcs can be replaced by a single one if we create the arc (i, l) for which the composite extension function becomes $f_{il}(T_i) = 60$ if $T_i + 30 \leq 420$.

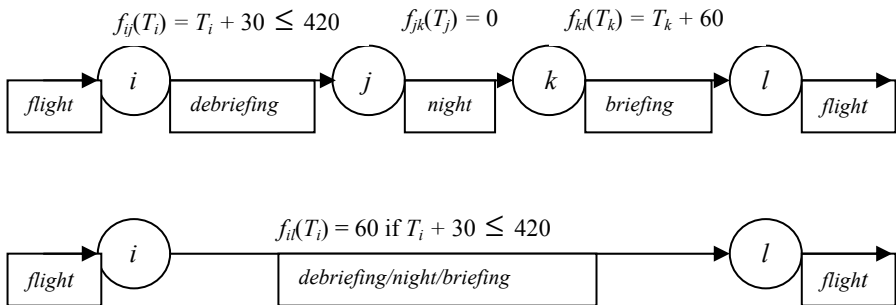


Figure 5.5. An example of the composition of extension functions

It sometimes happens that, in order to evaluate the value of a resource variable at vertex j by using the arc (i, j) , it is necessary to use several resource components calculated at vertex i . This is the case in air transportation where the cost of a pilot rotation can depend on the number of flight hours and the number of worked hours, the time of post-flight rest, the number of flights, the number of days, etc. Intuitively, we thus see that the extension function $f_{ij}(T_i)$ can be generalized to $f_{ij}(T_i)$, where the vector T_i corresponds to the set of all resource variables calculated at vertex i . Other examples have been given in the literature, notably in the carriage of goods when it is possible during a tour to allow at the same time both pickups and deliveries [DES 98a]. However, it is not within the scope of this work to go into the details of such situations.

5.4.2. Modeling examples

In the preceding sections, we have given some examples showing the possibilities of modeling complex situations by using the proposed approach. We have indicated how to manage the case of a cost function that guarantees the payment of a minimum work time and how to satisfy in a strict manner the lower bound of a time window. We present some additional examples to illustrate the versatility of the solution method discussed in these pages.

EXAMPLE 5.3. Let us consider the routing of vehicles of different types for which the operating costs from a single depot o are the same. Assume there are capacities of 85, 90 and 100 units. It is clear that three sub-problems can be used, one per vehicle type. However we can use a single sub-problem by creating an artificial vertex \tilde{o} at the exit of the depot. Three parallel arcs connect vertex o to vertex \tilde{o} : on the first arc, the load resource consumes 15 units; the consumption is 10 units on the second, while it is zero on the third. Therefore, the residual capacities from vertex \tilde{o} are 85, 90 and 100 units.

The preceding example shows that sometimes it suffices to modify the network structure to model a rule that *a priori* can appear quite complex. The following example is remarkable in this regard.

EXAMPLE 5.4. Most of the labor agreements allocate rest periods and meals during a workday. How can we model the attribution of a 90-minute lunch period in a time interval of 2 hours, at least 4 hours after the beginning of the workday? We present in Figure 5.6 a visual solution for this little brainteaser. The upper part gives a representation of a space-time network where are indicated in a very schematic manner the origin vertex o and the destination vertex d as well as N , the set of clients served. If there is no break allocated for lunchtime, an itinerary in this network begins at o , visits a subset of clients of N , and ends at d . The lower part of

the figure presents a network with the lunch period: between 4 and 6 hours after the beginning of the workday (a time bracket of 2 hours), the space-time network is duplicated and presented on two levels. These levels are connected by lunch arcs with a duration of 90 minutes; it then becomes impossible to construct an itinerary from o to d without going through these lunch arcs. Note that since the lunch duration is 90 minutes, the duplicated portion of the space-time network is reduced to only 30 minutes.

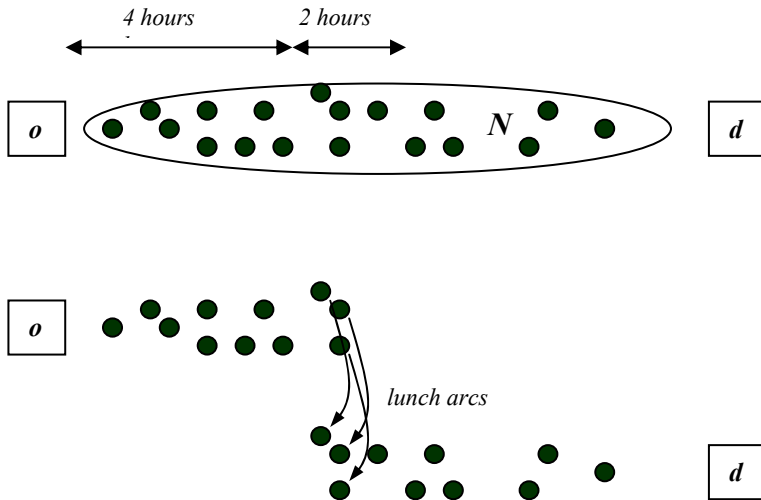


Figure 5.6. *A two-level space-time network*

The preceding example can be generalized to the case of several breaks and meals. We notice that in practice the network size is only increased a little. This modeling trick has been used with success for an air carrier to determine the assignment of monthly tasks to the pilots. The labor agreements imposed exactly 10 days off per month according to specific patterns; a space-time network requiring nine levels allows us to satisfy this unavoidable rule.

We first introduced at the beginning of the chapter the problem of the shortest path with time windows, a problem that requires only a single resource (the time accumulated since the start of the path). Following this, we mentioned the generalization with several resources [DES 86, IRN 05]. As an example, we can give the classical problem of vehicle routing with time windows that requires the

addition of a resource concerning the load transported. Most applications use resources for which the extension functions from one vertex to another are non-decreasing. Sometimes this is not the case, as the following example shows.

EXAMPLE 5.5. In some vehicle routing applications we can slightly modify the time window $[a_i, b_i]$ imposed by a client i by paying a penalty if we are either early or late for the start of service. As Figure 5.7 illustrates it, the penalty for the delay presents no difficulty since it is an increasing function of the time. On the other hand, the penalty is decreasing if we begin the service in advance. For this much more difficult case we developed a special algorithm for this new shortest path with time windows to be able to correctly treat this particularity [IOA 98].

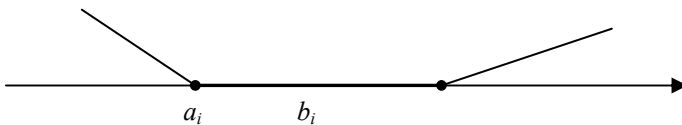


Figure 5.7. *Cost function on the start of service*

In practice, this type of algorithm that takes into account decreasing extension functions is, in CPU time, one order of magnitude slower than the preceding algorithms and we try to avoid their use by discretizing the corresponding time portion with only some time values. It is however necessary to know that the number of applications that interest us becomes more and more important. There are several of them in air transportation. One concerns the synchronization of a group of flights: for the flight from Montreal to Paris on Monday, Wednesday and Friday evenings we have to determine a departure schedule so that it is at exactly the same time each day. We thus have constraints connecting the departure time of the three flights. In the course of solving this, the dual information provided by the master problem penalizes either negatively or positively the departure time of a flight according to whether the current time at a vertex is too early or too late in comparison with the others [IOA 99]. Another application aims to sufficiently space the high frequency round-trip flights between two large cities [BEL 06]. Finally, we can measure the importance of this type of algorithm when we think about all the interdependent time components if, in the course of operations, we must redo the schedules in order to cope with a disruption of the scheduled hours [STO 01].

5.5. Applications in large transportation networks

5.5.1. Urban transportation

The first of these applications to be realized in large transportation networks was the scheduling of bus and metro drivers [DES 89]. This was in fact at the origin of the research done from 1984 to 1986 regarding algorithms for the shortest path problem with resource variables. However, it was necessary to wait until 1990 to see the first commercial installation in Lyon, France. In the following 15 years, other installations were added in more than 250 cities, including Tokyo, Singapore, Helsinki, Stockholm, Vienna, Barcelona, New York and Chicago. The GENCOL optimizer that solves mathematical model [5.2] by column generation is at the heart of the HASTUS software system distributed by GIRO, a Montreal consulting firm in operations research.

It was in order to overcome the difficulties encountered with the data in Tokyo in 1993 that many technical developments were made. The test problems consisted of 2,000 to 3,000 bus trips, and metro and suburban train trips, with work periods of one or two days, and very complex labor agreements. It was thus impossible for the Japanese schedulers to find solutions satisfying all the rules of their contract. Three months after the start of the research project, the time needed to correctly model the space-time network and all the rules, the GENCOL optimizer produced solutions with substantial savings of about 15% of the payroll.

Two other applications originated in urban transportation. The optimizer can also be used to allocate buses to routes: one wants to find the vehicle routes at the least cost by selecting the buses from the bus stations of a city [RIB 94]. Finally, GENCOL is also integrated in DARSY (*Dial-a-Ride System*) for the handicapped and elderly people [DES 88c]. The prototype was produced during the period 1985–1987. It was first necessary to develop a new algorithm to solve the shortest path problem integrating pickup and delivery [DUM 91]. We should also realize that every day in the large Canadian cities, more than 5,000 people need adapted door-to-door transportation, where they can specify the desired hours. In the past, those asking for transportation had to reserve a seat a week in advance. Optimized itineraries are now done in a single evening, and additional requests are inserted in the course of operations.

5.5.2. Air transportation

The research in air transportation began to take off in 1990 with work done for Air France [DES 97a]. Since the beginning of the 1960s, the problem of preparing crew rotations has excited the operations research community and it still provides

both theoretical and practical research projects. In short, we want to assign a crew to each flight in the form of a circuit of several days. The flying personnel (pilots, crew members, etc) must terminate their work period in the city of departure (base) and the rotations must satisfy all the rules imposed by the labor agreements. The mathematical formulation is again the one presented in model [5.2] with vector b taking unitary values. The number of admissible rotations is astronomical. For a small problem of an American company with only 300 flights, there were more than 190 million million variables! Even in generating all the good rotations, it is usually not possible to find solutions satisfying all the constraints of a problem. The best solutions generally combine good and less good rotations. The column generation method however allow us to implicitly have access to all these rotations.

Regarding computational results, let us consider the following example. It is a problem encountered at Air Canada for the crews on DC-9 and A320 planes counting, in one month, 5 bases and 11,914 flight segments. For this airline carrier, the variable costs represented 7.80% of the total cost of the rotations. The solution found by the GENCOL optimizer reduced this value to only 2.03%.

The GENCOL optimizer is integrated in the Altitude Suite [DES 00], a family of products distributed by the consulting firm AD OPT in Montreal, a division of Kronos Canadian Systems Inc. More than 20 airline carriers use it to prepare their crew rotations, to do the monthly planning [GAM 98, GAM 99] and to manage their fleet of planes [DES 97b]. As a percentage, the savings are of the same order of magnitude for all these applications. Tests realized for the monthly planning at Air France concerning the attribution of 3,000 rotations to 840 crew members based at Charles de Gaulle airport in Paris have given a coverage for regular crew members that is 7.6% more than the method used by the company. To manage air fleet, tests realized for Air Canada have shown that for a problem comprising nearly 3,000 flights per week and 91 planes of 9 different types, by authorizing time windows of more or less 20 minutes in relation to the proposed flight schedule, it was possible to reduce the operating costs by 8.9%.

For large air carriers, these optimization results represent savings of tens of millions of dollars per year and easily justify the money invested in research.

5.5.3. Rail transportation

We can also find some applications of the column generation method in rail transportation. In a general manner, making schedules for train drivers is similar to bus and metro drivers. The use of locomotives on freight trains is in several regards a problem similar to that of using several crew members on flights. Indeed, it is necessary to use up to six or seven locomotives to pull (cover) certain trains, just as

most of the flights require several crew members. However, for the locomotives, the coverage demand of the trains is also given in terms of horsepower units. The locomotives are assigned to a train and then, as needed, removed and used for other trains at certain stations according to whether the number of railway cars increases or decreases. The formulation once again corresponds to model [5.2] with vector b taking integer values while the number of coupling constraints, given by the dimension of the vector d , is much greater than for the applications encountered in air transportation. The main consequence is that we obtain very fractional solutions for the linear relaxation of the model, which demanded major research developments in order to obtain acceptable integer solutions [ZIA 99].

As an example of computational results, we can give those of the tests done for the Canadian National Railway [ZIA 97]. For a problem solved over a week and comprising nearly 2,000 trains and 26 locomotive types, the number of locomotives required to cover all the trains has been reduced to about 1,000, a reduction of 9.2% in relation to the solution of the schedulers. Once again, this represents savings of several hundred million dollars.

5.6. What does the future look like?

The solution method by column generation of the mathematical model [5.2] is of course not only limited to the applications presented in this text in urban, air and rail transportation. We also find some in the domain of maritime transportation for the management of chemical stocks [CHR 05]; in the domain of energy management by the control (power cuts) of electric water-heaters [LAU 95]; in production for flexible manufacturing [GEL 05], etc. In fact, every problem whose solution can be represented by a set of paths in a space-time network constitutes a field of potential applications for the presented model.

In large transportation networks, the process of operations management is broken down into several parts and the solution of the succession of problems is obtained in a sequential manner. In air transportation [DES 98b], for example, we first determine the flight schedules, then the type of plane for each flight. Next, we must determine the routing of the planes, the crew rotations, and finally the monthly planning. Recent applications aim at two-level problems, where we manage two components together so as to benefit from their interaction: flight schedules and plane itineraries [DES 97b], plane itineraries and crew rotations [COR 01b, MER 05], bus itineraries and driver schedules [HAA 01], assignment of engines and choice of railway cars in passenger transportation [COR 00, COR 01a], etc. It is also necessary to realize that during the operations, there are often disturbances and we must then simultaneously consider, in a short period, several components in order to repair the schedules [STO 98, STO 01, MER 07]. At Bombardier Flexjet, which

offers fractional aircraft services, we even combined within the same model the first four stages of the planning process, that is, the flight schedules in given time windows, the selection of the type of plane, the plane itineraries, and the crew rotations: total savings in 30 months exceeded \$50 million [HIC 05].

The constant augmentation of the size of the problems brings new challenges. Thus, several methods of mathematical decomposition are utilized jointly: the Dantzig-Wolfe decomposition, Benders' decomposition and the Lagrangian relaxation (see [COR 01b, HUI 05, LUB 05]). It also becomes necessary to dynamically manage in model [5.2] the number of constraints as well as the number of variables [ELH 05]. Finally, it is necessary to reduce the cpu time dramatically. For this last aspect, it is clear that we must make greater and greater use of approximations. On the other hand, it is through a better understanding of the theoretical mechanisms that we are able to choose those that are relevant [DUM 99, BEN 06]. In the course of the past few years, the decomposition method ACCPM (*Analytic Center Cutting Plane Method*) compatible with the solution by interior points has also been developed [GOF 92]. Although in theory it is much more competitive than the Dantzig-Wolfe decomposition presented in this text, nevertheless its use in an industrial context entails an acceleration of its diverse components.

Finally, the concern to preserve and even augment the quality of life of the personnel is increasingly present. Extreme optimization can have perverse effects and it is important that the mechanisms that favor the acceptance of solutions by the personnel involved be put forward. We thus speak of the regularity of airline routes so as to decrease the stress of flight crews. To monthly planning problems, we add *desiderata* and we explicitly take seniority into consideration [GAM 98]. We can even, from one month to another, balance the workloads. The information systems now being well implemented in the majority of the organizations, it is much easier to maintain and update the necessary information with the implementation of highly capable optimization software systems for operations management in transportation networks and elsewhere.

5.7. Bibliography

- [BEL 06] N. Bélanger, G. Desaulniers, F. Soumis, and J. Desrosiers, "Periodic airline fleet assignment with time windows, spacing constraints, and time dependent revenues", *European Journal of Operational Research*, vol. 175, no. 3, 1754–1766, 2006.
- [BEN 06] H. Ben Amor., J. Desrosiers and J.M. Valerio de Carvalho, "Dual-optimal inequalities for stabilizing column generation", *Operations Research*, vol. 54, no. 3, 454–463, 2006.

- [BOD 83] L. Bodin, B. Golden, A. Assad and M. Ball, "The state of the art in the routing and scheduling of vehicles and crews", *Computers & Operations Research*, vol. 10, 63–211, 1983.
- [CHR 05] M. Christiansen and B. Nygreen, "Robust inventory ship routing by column generation", in G. Desaulniers, J. Desrosiers, and M.M. Solomon (eds), *Column Generation*, Kluwer Academic Publishers, 197–224, 2005.
- [COR 00] J.-F. Cordeau, J. Desrosiers and F. Soumis, "A Benders' decomposition approach for the locomotive and car assignment problem", *Transportation Science*, vol. 34, no. 2, 133–149, 2000.
- [COR 01a] J.-F. Cordeau, J. Desrosiers and F. Soumis, "Simultaneous assignment of locomotives and cars to passenger trains", *Operations Research*, vol. 49, 531–548, 2001.
- [COR 01b] J.-F. Cordeau, J. Desrosiers and F. Soumis and G. Stojkococ, "Benders' decomposition for simultaneous aircraft routing and rescheduling", *Transportation Science*, vol. 35, no. 4, 375–388, 2001.
- [DAN 60] G.B. Dantzig and P. Wolfe, "Decomposition principle for linear programs", *Operations Research*, vol. 8, 101–111, 1960.
- [DES 83] J. Desrosiers, P. Pelletier and F. Soumis, "Plus court chemin avec contraintes d'horaires", *RAIRO – Recherche opérationnelle*, vol. 17, no. 4, 357–377, 1983.
- [DES 84] J. Desrosiers, F. Soumis and M. Desrochers, "Routing with time windows by column generation", *Networks*, vol. 14, no. 4, 545–565, 1984.
- [DES 86] M. Desrochers, "La fabrication d'horaires de travail pour les conducteurs d'autobus par une méthode de génération de colonnes", *Publication 470*, Centre de recherche sur les transports, University of Montreal, Canada, 1986.
- [DES 88a] M. Desrochers and F. Soumis, "A generalized permanent labeling algorithm for the shortest path problem with time windows", *INFOR*, vol. 26, no. 3, 191–212, 1988.
- [DES 88b] M. Desrochers and F. Soumis, "A reoptimization algorithm for the shortest path problem with time windows", *European Journal of Operational Research*, vol. 35, 1988, 242–254.
- [DES 88c] J. Desrosiers, Y. Dumas and F. Soumis, "The multiple vehicle dial-a-ride problem", in J.R. Daduna and A. Wren (eds), *Computer-Aided Transit Scheduling Lecture Notes in Economics and Mathematical System*, vol. 308, 15–27, 1983.
- [DES 89] M. Desrochers and F. Soumis, "A column generation approach to the urban transit crew scheduling problem", *Transportation Science*, vol. 23, no. 1, 1–13, 1989.
- [DES 95] J. Desrosiers, Y. Dumas, M.M. Solomon and F. Soumis, "Time constrained routing and scheduling", in M.O. Ball et al. (eds.) *Network Routing, Handbooks in Operations Research and Management Science*, vol. 8, 35–139, 1995.

- [DES 97a] G. Desaulniers, J. Desrosiers, Y. Dumas, S. Marc, B. Rioux, M.M. Solomon and F. Soumis, "Crew pairing at Air France", *European Journal of Operational Research*, vol. 97, no. 2, 245–259, 1997.
- [DES 97b] G. Desaulniers, J. Desrosiers, M.M. Solomon and F. Soumis, "Daily aircraft routing and scheduling", *Management Science*, vol. 43, no. 6, 841–855, 1997.
- [DES 98a] G. Desaulniers, J. Desrosiers, I. Ioachim, M.M. Solomon, F. Soumis and D. Villeneuve, "A unified framework for deterministic time constrained vehicle routing and crew scheduling problems", in T. G. Crainic and G. Laporte (eds.), *Fleet Management and Logistics*, Kluwer, Norwell, MA, 57–93, 1998.
- [DES 98b] G. Desaulniers, J. Desrosiers, M. Gamache and F. Soumis, "Crew scheduling in air transportation", in T. Crainic and G. Laporte (eds.), *Network Routing, Fleet Management and Logistics*, Kluwer, Norwell, MA, 169–185, 1998.
- [DES 00] J. Desrosiers, A. Lasry, D. McInnis, M.M. Solomon and F. Soumis, "Air Transat uses ALTITUDE to manage its aircraft routing, crew pairing, and work assignment", *Interfaces*, vol. 30, no. 2, 41–53, 2000.
- [DUM 91] Y. Dumas, J. Desrosiers and F. Soumis, "The pickup and delivery problem with time windows", *European Journal of Operations Research*, vol. 54, no. 1, 7–22, 1991.
- [DUM 99] O. du Merle, D. Villeneuve, J. Desrosiers and P. Hansen, "Stabilized column generation", *Discrete Mathematics*, vol. 194, 229–237, 1999.
- [ELH 05] H. Elhallaoui, D. Villeneuve, F. Soumis and G. Desaulniers, "Dynamic aggregation of set partitioning constraints in column generation", *Operations Research*, vol. 53, no. 4, 632–645, 2005.
- [GAM 98] M. Gamache, F. Soumis, D. Villeneuve, J. Desrosiers and E. G  linas., "The preferential bidding system at Air Canada", *Transportation Science*, vol. 32, no. 3, 246–255, 1998.
- [GAM 99] M. Gamache, F. Soumis, G. Marquis and J. Desrosiers, "A column generation approach for large scale aircrew rostering problems", *Operations Research* 47(2), 1998, 247–263.
- [GEL 05] S. G  linas and F. Soumis, "Dantzig-Wolfe decomposition for job shop scheduling", in G. Desaulniers, J. Desrosiers, and M.M. Solomon (eds.), *Column Generation*, Kluwer Academic Publishers, 271–301, 2005.
- [GOF 92] J.-L. Goffin, A. Haurie and J.-P. Vial, "Decomposition and nondifferentiable optimization with the projective algorithm", *Management Science*, vol. 38, 284–302, 1992..
- [HAA 01] K. Haase, G. Desaulniers and J. Desrosiers, "Simultaneous vehicle and crew scheduling in urban mass transit systems", *Transportation Science*, vol. 35, no. 3, 286–303, 2001,

- [HIC 05] R. Hicks, R. Madrid, C. Milligan, R. Pruneau, M. Kanaley, Y. Dumas, B. Lacroix, J. Desrosiers and F. Soumis, “Bombardier Flexjet significantly improves its fractional aircraft ownership operations”, Special Issue: 2004 Franz Edelman Award for Achievement in Operations Research and the Management Sciences, *Interfaces*, vol. 35, no. 1, 49–60, 2005,
- [HUI 05] D. Huisman, R. Jans, M. Peeters and P.M. Wagelmans, “Combining column generation and Lagrangian relaxation”, in *Column Generation*, G. Desaulniers, J. Desrosiers, and M.M. Solomon (eds.), Kluwer Academic Publishers, 247–270, 2005,
- [IOA 98] I. Ioachim, S. Gélinas, J. Desrosiers and F. Soumis, “A dynamic programming algorithm for the shortest path problem with time windows and linear node costs”, *Networks*, vol. 31, 93–204.
- [IOA 99] I. Ioachim, J. Desrosiers, F. Soumis and N. Belanger, “Fleet assignment and routing with schedule synchronisation constraints”, *European Journal of Operational Research*, vol. 119, no. 1, 75–90.
- [IRN 05] I. Irnich and G. Desaulniers, “Shortest path problems with resource constraints”, in G. Desaulniers, J. Desrosiers, and M.M. Solomon (eds.), *Column Generation*, Kluwer Academic Publishers, 33–67, 2005,
- [LAC 04] B. Lacroix and J. Desrosiers, “Altitude Manpower Planning – an integrated system that addresses the puzzle of planning a crew force”, *OR/MS Today*, April 2004.
- [LAU 95] J.-C. Laurent, G. Desaulniers, R.P. Malhamé and F. Soumis, “A column generation method for optimal load management via control of electric water heaters”, *IEEE Power Electric Systems*, vol. 10, no. 3, 1389–1400, 1995,
- [LUB 05] M.E. Lübbecke and J. Desrosiers, “Selected topics in column generation”, *Operations Research*, vol. 53, no. 6, 1007–1023, 2005,
- [MER 05] A. Mercier, J.-F. Cordeau and F. Soumis, “A computational study of Benders decomposition for the integrated Aircraft routing and crew scheduling problem”, *Computers & Operations Research*, vol. 31, 1451–1476, 2005,
- [MER 07] A. Mercier and F. Soumis, “An integrated aircraft routing, crew scheduling and flight retiming model”, *Computers & Operations Research*, vol. 34, no. 8, 2251–2265, 2007,
- [RIB 94] C. Ribeiro and F. Soumis, “A column generation approach to the multiple-depot vehicle scheduling problem”, *Operations Research*, vol. 42, no. 1, 41–53, 1994.
- [SOL 88] M.M. Solomon and J. Desrosiers, “Time window constrained routing and scheduling problems”, *Transportation Science*, vol. 22, no. 1, 1–13, 1988.
- [STO 98] M. Stojkovic, F. Soumis and J. Desrosiers, “The operational airline crew scheduling problem”, *Transportation Science*, vol. 32, no. 3, 232–245, 1998.

- [STO 01] M. Stojkovic and F. Soumis, “An optimization model for the simultaneous operational flight and pilot scheduling problem”, *Management Science*, vol. 47, no. 9, 1290–1305, 2001.
- [VIL 05] D. Villeneuve, J. Desrosiers, J. Lübbecke and F. Soumis, “On compact formulations for integer programs solved by column generation”, *Annals of Operations Research*, vol. 139, no. 1, 375–388, 2005.
- [ZIA 97] K. Ziarati, F. Soumis, J. Desrosiers, S. Gélinas and A. Saintonge, “Locomotive assignment with heterogeneous consists at CN North America”, *European Journal of Operational Research*, vol. 97, no. 2, 281–292, 1997.
- [ZIA 99] K. Ziarati, F. Soumis, J. Desrosiers and M.M. Solomon, “A branch-first, cut-second approach for locomotive assignment”, *Management Science*, vol. 45, no. 80, 1156–1168, 1999,

Chapter 6

Pickup and Delivery Problems with Services on Nodes or Arcs of a Network

6.1. Introduction

Pickup and delivery are activities that appear frequently in numerous companies and communities. They generally concern transporting merchandise from one depot to branch offices, delivering orders to clients, maintaining road networks, distributing mail, picking up rubbish, etc. These operations are often very costly and substantial savings can be made by reducing the length of the trips that must be made.

According to the nature of the service furnished by the vehicles and as a function of the geographical distribution of the clients, two main types of modeling emerge. The first consists of representing the clients by nodes of a network. The problem based on this model is called the *node routing problem* (NRP). Such a model is well adapted when the clients have precise distinct locations. The NRP has been and is still the object of many studies. Numerous variants of the NRP exist, going from the basic traveling salesman problem to problems with many more constraints, such as node routing problems with time windows.

Another type of modeling associates the clients with the arcs of a network. We then speak of an *arc routing problem* (ARP). An arc is a direct link between two nodes in a network. Less frequently present in the literature than the NRP, the ARP appears when the clients are located along the streets, or when the streets themselves need a service. Just as for the NRP, there are different variants of the ARP.

Most of the variants of the NRP and the ARP are *NP-hard*, that is, there exists today no algorithm capable of finding an optimal solution in computing time that does not explode when the size of the problem increases. To solve a NRP or an ARP of large size, we have to rely on heuristic methods that determine an approximation of the optimum of the problem. We present in this chapter a review of the principal heuristics that have been developed for the solution of the diverse variants of the NRP and the ARP.

6.2. Node routing problems

6.2.1. The traveling salesman problem

The traveling salesman problem was probably mentioned for the first time in a mathematical circle by A.W. Tucker in 1931. This problem has already been introduced in Chapter 3 in the general context of solution methods. Here, we present specific heuristics adapted to this problem. Let us remember that the formulation is particularly simple, which has captivated many researchers.

PROBLEM. A salesman, departing from home (or a depot), must visit a set of clients and then return home; the order in which he visits the clients should be such that the total distance he travels is as small as possible.

It is not necessary to be a particularly talented researcher to imagine algorithms producing good tours for the traveling salesman. However, the problem is NP-hard, and the best exact techniques (that is, guaranteeing optimality of the solution produced) can only treat instances with a few hundred clients. This explains why numerous researchers worked on developing efficient heuristics to solve the traveling salesman problem [LAW 85]. Of the most well-known constructive heuristics, we will mention only three here. First of all, the *Nearest Neighbor* algorithm is probably the simplest method to implement. It is described in Algorithm 6.1.

|

1. Choose a client v and consider a partial tour $T=(depot, v, depot)$.
2. Determine a client w who is not yet in T and who is the nearest to the last client v visited in T .
3. Add client w to the end of tour T . If all the clients have been visited, then STOP, if not return to 2.

Algorithm 6.1. *Nearest Neighbor algorithm*

Another algorithm, which is just as simple as the preceding one, is the *Cheapest Insertion* algorithm. It is described in Algorithm 6.2.

1. Choose two clients v and w and consider a partial tour T visiting only these clients, that is, $T=(depot, v, w, depot)$.
2. For each client z who is not yet in T , calculate the length $L(z)$ of the smallest detour brought about by inserting z between two consecutive clients on T .
3. Choose a client z who is not yet in T and who minimizes $L(z)$, and insert z in T (with the smallest detour). If all the clients have been visited, then STOP, if not return to 2.

Algorithm 6.2. *Cheapest Insertion algorithm*

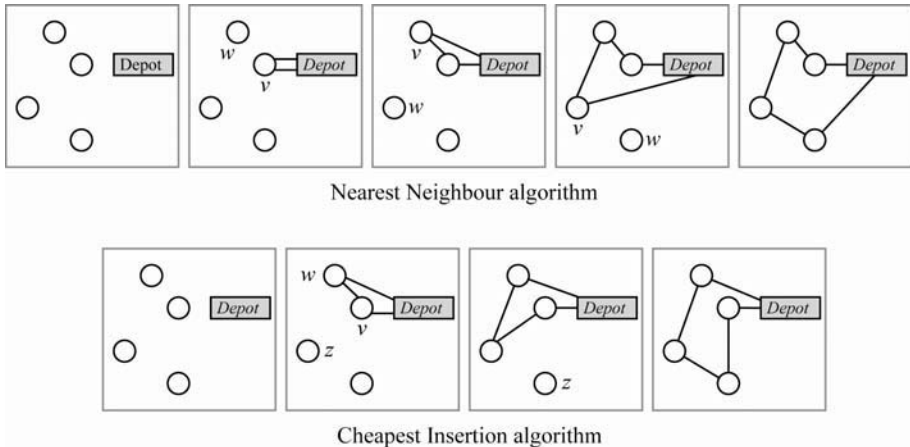


Figure 6.1. *Illustration of the Nearest Neighbor and the Cheapest Insertion algorithms*

These two heuristics for the traveling salesman problem generally produce solutions of reasonably good quality. They are illustrated in Figure 6.1. We can easily construct examples for which these heuristics are arbitrarily bad. More precisely, if we note $H(I)$ the length of the tour produced by one of these two heuristics for an instance I of the problem, and if $OPT(I)$ is the minimum length of a tour for instance I , then the ratio $H(I)/OPT(I)$ has no upper bound. This means that for each value $r > 1$ we can construct an instance I for which $H(I)/OPT(I)$ is greater than r .

By noting $D(x, y)$ the distance between two clients x and y , we say that the distances satisfy the *triangular inequality* if $D(x, y) \leq D(x, z) + D(z, y)$ for each triplet x, y, z of clients. In 1976, Christofides [CHR 76] proposed a heuristic guaranteeing that the length of a produced tour is in the worst case 50% more than the optimum, assuming that the distances satisfy the triangular inequality. This heuristic is described in Algorithm 6.3.

Determine a maximum tree of minimum cost A connecting the clients.

1. Let W be the set of clients who are the extremity of an odd number of arcs in A . This set necessarily contains an even number of nodes. Determine a maximum matching C of the elements of W of minimum cost.
2. The union of A and C induces a tour T that can eventually be shortened if we pass by the same client twice.

Algorithm 6.3. *Christofides' algorithm*

Determining a maximum tree of minimum cost at step 1, and a maximum matching of minimum cost at step 2, can be realized in polynomial time (see Chapter 2). This algorithm is illustrated in Figure 6.2.

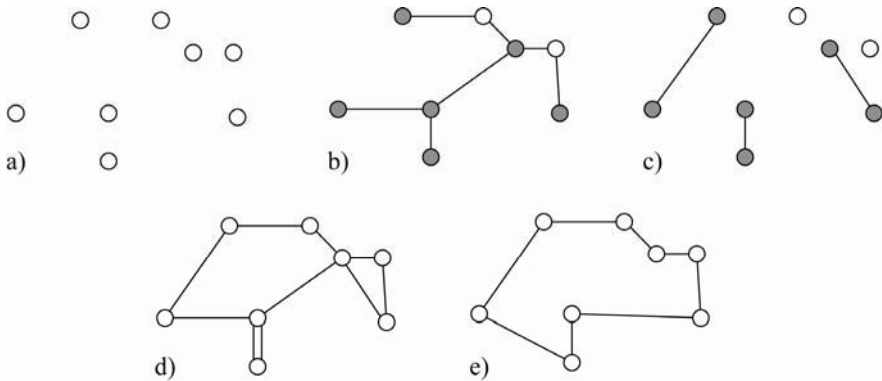


Figure 6.2. Illustration of Christofides' algorithm: a) set of clients to visit; b) maximum tree of minimum cost. The nodes in gray are the extremity of an odd number of arcs; c) maximum matching of minimum cost on the gray nodes; d) union of the tree with the matching; e) shortened tour that avoids passing by the same client twice

The algorithms described up to here were all proposed in the 1960s and 1970s. More efficient methods were proposed in the 1980s and new algorithms are regularly published in the scientific literature. The most efficient current methods are based on *Local Search techniques* (descent algorithm, simulated annealing, tabu search – see Chapter 3).

Numerous adaptations of Local Search techniques have been proposed for the solution to the traveling salesman problem. The set of solutions S to explore is simply the set of tours visiting all clients, and the function F to minimize is the length of these tours. The diverse adaptations proposed differ mainly in the definition of the notion of neighborhood $N(s)$ of a solution s . A very common technique consists of putting in $N(s)$ all the tours that can be obtained by replacing two arcs of s by two arcs that are not in s . Two neighbor solutions are represented in Figure 6.3. More refined neighborhoods have been proposed, and we refer the reader to [JOH 97] for more details on Local Search techniques to solve the traveling salesman problem. It clearly appears from the numerous experiments carried out that the Local Search techniques generally give tours whose length is less than 1% more than the optimal length of a tour.

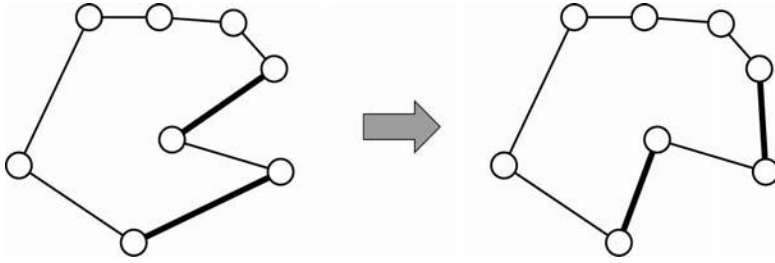


Figure 6.3. Two neighbor tours: the tour on the right is obtained from the tour on the left by replacing two arcs with two new arcs

6.2.2. Vehicle tours with capacity constraints

If merchandise must be delivered to clients, it is often necessary to make deliveries with the help of several vehicles, each vehicle having only a limited capacity. Delivery planning then necessitates the solution of two sub-problems:

- we must determine the set of clients that each vehicle must serve;
- we must determine the order in which each vehicle serves its clients.

We will suppose here that all the vehicles have the same load capacity, that each vehicle starts and finishes its tour at the depot, and that the deliveries to a client are made with the help of a single vehicle (that is, we forbid all solutions where several vehicles share delivery to the same client). The goal is to minimize the total distance.

To solve this problem, Clarke and Wright [CLA 64] proposed, in 1964, a constructive algorithm, which is still frequently used by companies today. To understand this algorithm, it is necessary to define the notion of *saving*.

DEFINITION. Given two clients v and w , the saving $s(v, w)$ is defined as the gain in length obtained by delivering v and w in the same tour (depot, v , w , depot) instead of using two tours (depot, v , depot) and (depot, w , depot).

By noting $D(x, y)$ the distance between two nodes x and y of the network, we thus have $s(v, w) = D(v, \text{depot}) + D(\text{depot}, w) - D(v, w)$. This concept is illustrated in Figure 6.4.

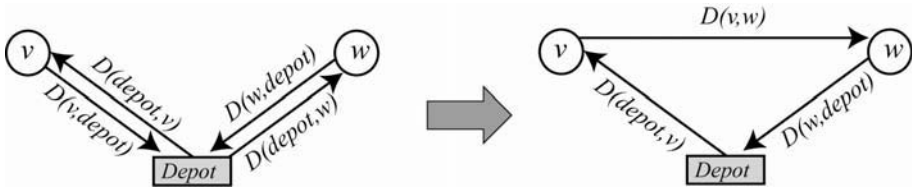


Figure 6.4. Illustration of the saving notion

Clarke and Wright's algorithm constructs a first route by inserting step by step, at the start or at the end of the tour, clients that bring about the greatest savings. When no other client can be added to this first tour without inducing a capacity overload, the algorithm constructs a second route, according to the same principle, with the remaining clients. This process is repeated until all the clients have been served. This algorithm is more precisely described in Algorithm 6.4.

1. Determine the savings $s(v, w)$ for all pairs of clients, and arrange these values in decreasing order.
2. Choose the first saving $s(v, w)$ from the list so that v and w are not yet served, and so that the sum of requests of clients v and w does not exceed the vehicle capacity. Create a tour $T = (\text{depot}, v, w, \text{depot})$.
3. Choose the first saving $s(v, w)$ of the list so that v is the first or the last client served in tour T , w is still not served by a vehicle, and the merchandise to deliver to client w can be added to tour T without inducing a vehicle capacity overload.
4. Add w to the end of T if v was the last client of the tour, and at the beginning of T if v was the first client of the tour.
5. If all the clients are served, STOP. If not, if no client can be added to T without causing overload of the vehicle capacity, then store this tour and return to step 2. If not return to step 3.

Algorithm 6.4. Clarke and Wright's algorithm

This algorithm has now been improved, mainly with the help of more adequate definitions of the notion of saving. More details on this type of technique can be obtained by consulting [GOL 85].

The two-phase methods described below generally give better results than a constructive algorithm. These methods are principally of two types.

– The *cluster first – route second* strategy first determines a section of the clients into clusters, each having a total demand not exceeding the vehicle capacity. In a second phase, a traveling salesman problem is solved on each cluster.

– The *route first – cluster second* strategy first solves a traveling salesman problem in order to obtain a single giant tour visiting all clients (without taking vehicle capacity into account). Then, in a second phase, the tour is partitioned into smaller tours satisfying all the vehicle capacity constraints.

These two strategies are illustrated in Figure 6.5. Examples of using these strategies are described in [CHR 85].

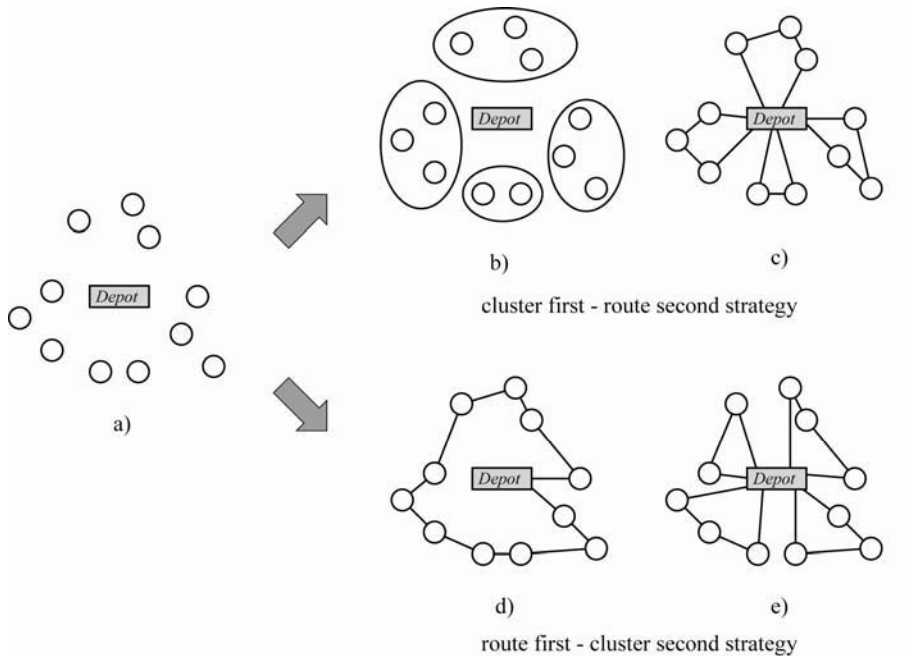


Figure 6.5. The two strategies of the two-phase methods: a) a set of clients to visit; b) partition of the clients into clusters; c) construction of a tour on each cluster; d) solution of the traveling salesman problem; e) partition of the tour into smaller tours satisfying all the vehicle capacity constraints

Local Search techniques can also be adapted to the routing problems with capacity constraints. In this case, a solution s is defined as being a set of tours satisfying all the client requests and respecting capacity constraints. The value $F(s)$ of a solution is the total distance covered by the set of vehicles, and the

neighborhood $N(s)$ of a solution s contains, for example, the set of solutions that can be obtained from s by moving the service of a client from one vehicle to another. Two neighbor solutions are represented in Figure 6.6.

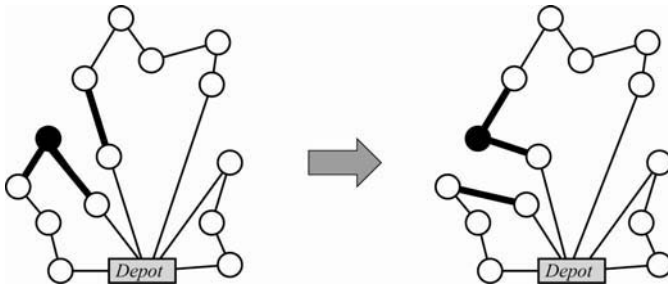


Figure 6.6. Two neighbor solution: transfer of a client from one tour to another

The adaptation of Local Search techniques described above can sometimes pose a problem because of limited vehicle capacities. Let us consider for example a solution constituted of two tours T_1 and T_2 , the first comprising four clients, each having a request for one unit of merchandise, and the second having only two clients with requests for two units of merchandise. If vehicle capacities are limited to four units of merchandise, it is not possible to move a client from one tour to another without causing a capacity overload. Indeed, if a client is transferred from T_1 to T_2 , the tour T_2 will have an overload of one unit, and if a client is transferred from T_2 to T_1 , it is an overload of two units that we will have in T_1 . Even a permutation between a client of T_1 and a client of T_2 does not solve the problem since that would induce an overload of one unit in T_1 . It is thus not possible to regroup the clients differently in the tours while the optimal solution can eventually consist of two tours, each having a client with a request of 2 units and two clients with a request for one unit.

To get around this problem, we can decide to accept violations of the capacity constraint. In other words, a solution s is a set of tours satisfying all the client requests, but not necessarily respecting capacity constraints. When a solution s violates capacity constraints, we define a penalty $P(s)$ proportional to the capacity overloads. The value $F(s)$ of solution s is then obtained by adding the total distance covered by vehicles with the penalty $P(s)$. Such an approach has been used with success in [GEN 94].

6.3. Arc routing problems

In this section we will describe some algorithms to solve vehicle routing problems in which clients are situated on the arcs of a network. If all the arcs in the network are oriented, we can easily transform an ARP into a NRP in the following manner. We create a complete network G where each node corresponds to an arc $a \rightarrow b$ to be serviced. The distance between two clients $a \rightarrow b$ and $c \rightarrow d$ is the length of the shortest path connecting the terminal extremity b of $a \rightarrow b$ to the initial extremity c of $c \rightarrow d$. By solving a NRP in G , we obtain a tour T that can then easily be transformed into a solution S of the ARP. The difference in length between T and S corresponds to the total length of the arcs to be serviced. This transformation is illustrated in Figure 6.7.

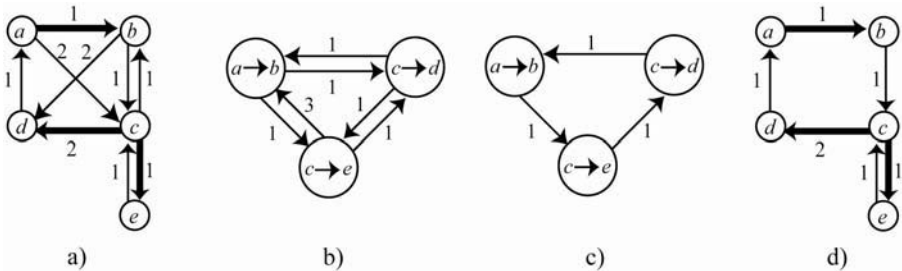


Figure 6.7. Transformation of an oriented ARP into a NRP: a) network in which one must solve an ARP. Clients are represented by heavy lines; b) associated graph in which one must solve a NRP; c) optimal tour T for the NRP. Its length is 3; d) optimal tour S of the ARP, deduced from tour T . Its length is $7=3+4$

6.3.1. The Chinese postman problem

The Chinese postman problem was introduced for the first time by the mathematician Meigu Guan in 1962 [GUA 62]. Given a graph, it is the problem of determining a circuit of minimum total length traversing each arc of the graph at least once. The optimal solution to this problem can be obtained in polynomial time when the graph is non-oriented (each arc can be traversed in any direction) or totally oriented (one direction is prohibited for each arc in the graph). On the other hand, if certain arcs in the graph have an orientation imposed while others do not, we then speak of a mixed graph and the problem to solve becomes NP-hard.

If each node of the considered network G has an even number of neighbors, and if the arcs are non-oriented, the Chinese postman problem consists of determining a

cycle passing exactly once through each arc in the graph. This problem can easily be solved with Algorithm 6.5. It is illustrated in Figure 6.8.

Determine a cycle C in the graph. If C covers all arcs, STOP.

1. Choose a node v belonging to C and incident to an arc that is not in C . Construct a second cycle C' containing v so that C and C' have no arc in common.
2. Join C and C' to form a cycle C'' . This merging is done by departing from node v , covering the arcs of cycle C , then being back in node v by visiting the arcs of cycle C' to finally terminate in v .
3. Set C equal to C'' . If C covers all the arcs, STOP; if not, return to 2.

Algorithm 6.5. Algorithm for the non-oriented Chinese postman problem, where each node has an even number of neighbors

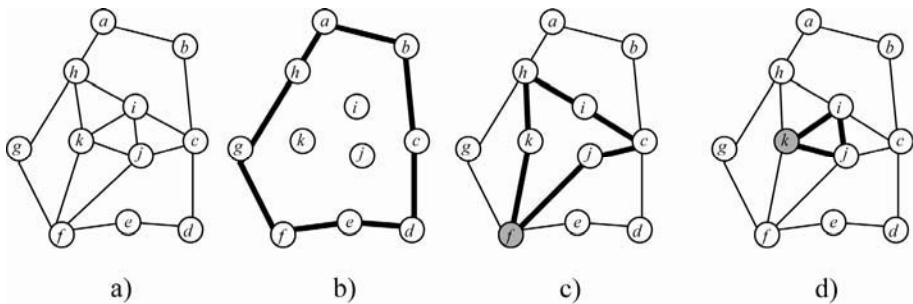


Figure 6.8. Solution of the Chinese postman problem in a network where all nodes have an even number of neighbors: a) network in which we want to construct a tour.

All the nodes have an even number of neighbors; b) detection of a first cycle ($a, b, c, d, e, f, g, h, a$); c) detection of a second cycle (f, k, h, i, c, j, f) joined with the first cycle by node f . We thus obtain a new cycle ($a, b, c, d, e, f, k, h, i, c, j, f, g, h, a$); d) detection of a third cycle (k, i, j, k) joined with the preceding cycle by node k . We thus obtain a new cycle ($a, b, c, d, e, f, k, i, j, k, h, i, c, j, f, g, h, a$)

If the nodes of the graph have an odd number of neighbors, we can duplicate a certain number of arcs so that the augmented graph has only nodes with an even number of neighbors. The choice of arcs to duplicate so that the Chinese postman tour is of minimum length can be done with the help of the search for a maximum matching of minimum cost in a complete auxiliary graph (that is, where all nodes are connected two by two). The algorithm is described more precisely in Algorithm 6.6.

1. If all the nodes in the graph G have an even number of neighbors, then determine a tour with the help of the preceding algorithm, **STOP**.
2. Determine the set W of nodes having an odd number of neighbors. This set necessarily contains an even number of nodes. Construct a complete auxiliary graph G' where the set of nodes is W and whose cost associated to each arc (v, w) is equal to the length of the shortest path between v and w in the original graph.
3. Determine a maximum matching of minimum cost in G' . For each arc (v, w) of this matching, add in the original graph a copy of all the arcs belonging to a shortest path from v to w . The nodes in the augmented graph now have an even number of neighbors.
4. Determine a tour in the augmented graph with the help of Algorithm 6.5.

Algorithm 6.6. *Algorithm for the non-oriented Chinese postman problem*

Steps 2 and 3 of this algorithm are illustrated in Figure 6.9. The search for a maximum matching of minimum cost can easily be done with the help of polynomial algorithms (see Chapter 2).

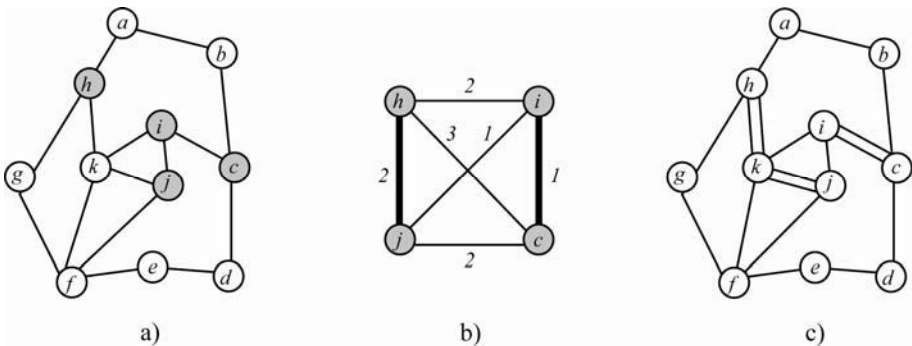


Figure 6.9. Transformation of a network so that all nodes are the extremity of an even number of arcs: a) network G in which we want to construct a tour. Nodes c , h , i and j are the extremity of an odd number of arcs. The arcs here all have a unitary length; b) auxiliary graph G' in which are indicated all the lengths of the shortest paths between c , h , i and j . The optimal matching is constituted by arcs (c, i) and (h, j) ; c) by adding the shortest path (c, i) between c and i , and (h, k, j) between h and j , we obtain a new network in which each node is the extremity of an even number of arcs

When all the arcs in the considered network G are oriented, we can solve the Chinese postman problem by using an approach similar to the one described above. More precisely, if every node v of the graph has as many arcs entering in v as arcs leaving from v , we can then easily determine an oriented circuit in the graph, passing exactly once (in the right direction) through each arc in the graph. If not, we must duplicate certain arcs so that each node v has as many arcs entering v as arcs leaving v . The choice of arcs to duplicate is made with the help of the solution to a minimum cost flow problem (see Chapter 2) in an auxiliary graph. More details on this algorithm are given in, for example, [EDM 73].

The algorithms presented up to here produce an optimal tour of the Chinese postman when no arc or all of them are oriented. In the mixed case, when certain arcs in the network are oriented while others are not, the Chinese postman problem becomes NP-hard, which means that there exists no algorithm to solve the problem in polynomial time. Solutions of reasonably good quality can be obtained, for example, by adequately orienting the non-oriented arcs and then duplicating certain arcs to obtain a totally oriented augmented graph where each node v has as many arcs entering v as arcs leaving v . We are thus returning to the preceding problem that we know how to solve. More details on the manner of orienting the arcs, as well as the choice of arcs to duplicate, can be obtained by consulting, for example, [MIN 79].

6.3.2. The rural postman problem

Let A denote the set of arcs of the considered network G . In the Chinese postman problem, one seeks a minimum cost tour that traverses all arcs of A at least once. In many contexts, however, it is not necessary to traverse all arcs of the network, but to *service* only a subset $R \subseteq A$ of *required* arcs (also called clients), traversing if necessary some arcs of $A \setminus R$. When a required arc is traversed on a tour T , we say that T *covers* this arc. A *covering tour* for R is a tour that covers all arcs of R , which means that all arcs of R are traversed at least once. When R is a proper subset of E , the problem of finding a minimum cost-covering tour for R is known as the *rural postman problem*. It was introduced by Orloff in 1974 [ORL 74].

If the sub-network of G , composed uniquely of required arcs, is connected (that is, one can go from one client to another by using only service arcs), then the rural postman problem can be solved in polynomial time in cases where G is non-oriented or totally oriented. The ingredients to use are identical to those used for the Chinese postman problem. As an example, we give below Algorithm 6.7 for the non-oriented case. This algorithm is illustrated in Figure 6.10.

1. Let G_R be the partial network G composed uniquely of required arcs. Determine the set W of nodes having an odd number of neighbors in G_R . This set W necessarily contains an even number of nodes. Construct a complete auxiliary graph G' whose set of nodes is W and whose cost associated with each arc (v, w) is equal to the length of the shortest path between v and w in the original graph G .
2. Determine a maximum matching of minimum cost in G' . For each arc (v, w) of the optimal matching, add in graph G_R a copy of all the arcs belonging to a shortest path from v to w . The nodes in the augmented graph now have an even number of neighbors.
3. Determine a tour by solving the Chinese postman problem in the augmented graph.

Algorithm 6.7. *Algorithm for the non-oriented rural postman problem where the clients form a connected sub-network*

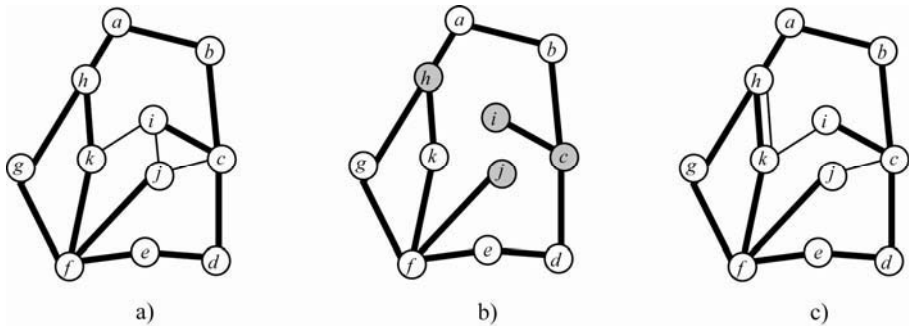


Figure 6.10. *Illustration of the algorithm for the non-oriented rural postman problem in the case where the clients form a connected sub-network: a) network in which we want to construct a tour. All arcs correspond to clients except (k, i) , (i, j) , and (j, c) . All arcs have unit length; b) partial network only containing required arcs. The nodes c , h , i , and j are the extremity of an odd number of arcs. The best matching among these four nodes consists of connecting h to i and j to c ; c) augmented graph containing the required arcs and the matching. A rural postman tour is thus, for example, $(a, b, c, i, k, h, k, f, j, c, d, e, f, g, h, a)$*

When the clients do not constitute a connected sub-network of the original graph, the problem is NP-hard (that is, no polynomial algorithm is known to solve this problem). Frederickson [FRE 79] proposed an algorithm for the non-oriented case. This algorithm guarantees that the length of the produced tour is in the worst case 50% more than the optimum, assuming that the distances satisfy the triangular inequality. Frederickson's algorithm is described in Algorithm 6.8 and illustrated in

A completely different approach is proposed in [HER 99] to solve the rural postman problem. We only describe here the non-oriented case. All the ingredients that we will present can, however, be easily adapted to oriented and mixed cases. Let us first of all consider a tour T covering a subset R' of required arcs. The procedure described in Algorithm 6.9 tries to reduce the length of T without modifying the set of covered arcs. It is illustrated in Figure 6.12.

Let T be a tour covering a subset R' of required arcs. Choose a direction for the tour T and choose a departure node v on T .

1. Determine a node w on T such that the length of the path linking v to w is as long as possible, while the path linking w to v covers all required arcs in R' . Let P denote the path going from v to w in T .
2. Let Q be the path connecting w to v in T , obtained by eliminating P from T . If there exists on Q a non-serviced arc (z, w) entering w , then the first arcs of Q to this arc (z, w) induce a circuit (w, \dots, z, w) ; in such a case, inverse the orientation of all arcs in this circuit and return to 2. If not, go to 4.
3. If the length of the shortest path from v to w in the original graph G is less than the length of P , then replace P by this shortest path.
4. Repeat steps 2 to 4 by considering the two possible orientations of T and each possible departure node v on T until no further improvement can be obtained.

Algorithm 6.9. *Shorten procedure that allows the shortening of a tour without modifying the set of serviced clients*

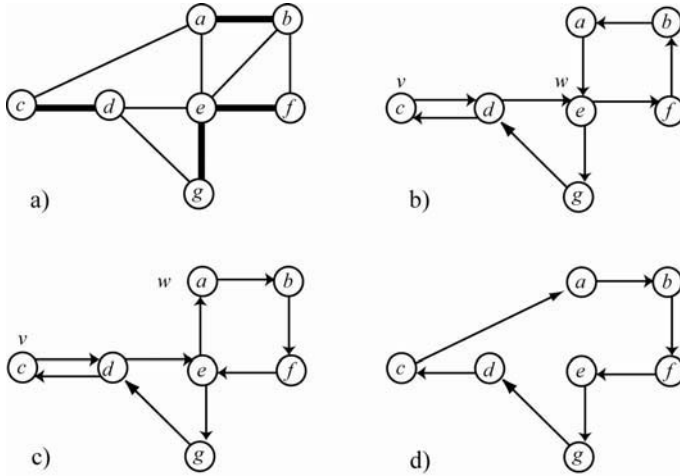


Figure 6.12. Illustration of the Shorten procedure: a) network with a set of four clients to serve in heavy lines. All arcs have unit length; b) an oriented tour T covering all required arcs. By choosing $v=c$ as departure node and by applying step 2 of Shorten procedure, we can reach $w=e$ with $P=(c, d, e)$. The path from e to c obtained by removing P from T is $Q=(e, f, b, a, e, g, e, d, c)$. The arc (a, e) of Q is a non-serviced arc entering $w=e$; c) new tour obtained by inverting the orientation of the circuit (e, f, b, a, e) on Q . We can now reach $w=a$ from $v=c$ with $P=(c, d, e, a)$; d) shorter covering tour obtained by replacing the path $P=(c, d, e, a)$ of length 3 with the arc (c, a) of length 1

The second procedure described in Algorithm 6.10 allows us to add a client to an existing tour. This algorithm is illustrated in Figure 6.13.

Let T be a tour covering a subset R' of clients. Let $(v, w) \notin R'$ be a client to add in T .

1. If neither v nor w appears in T , then determine the node z in T for which the sum of the distances to v and w is minimum, and add to T a cycle constituted of the shortest path from z to v , the required arc (v, w) , and the shortest path from w to z .
If not, if exactly one node among v and w appears in T (let us say v), or if the two nodes v and w appear in T but not consecutively, add to T the cycle (v, w, v) .
2. Add (v, w) to R' and try to diminish the length of the tour T with the help of the procedure *Shorten* described in Algorithm 6.9 (by imposing a service on all arcs in the new set R').

Algorithm 6.10. Add procedure that allows us to add a client to a tour

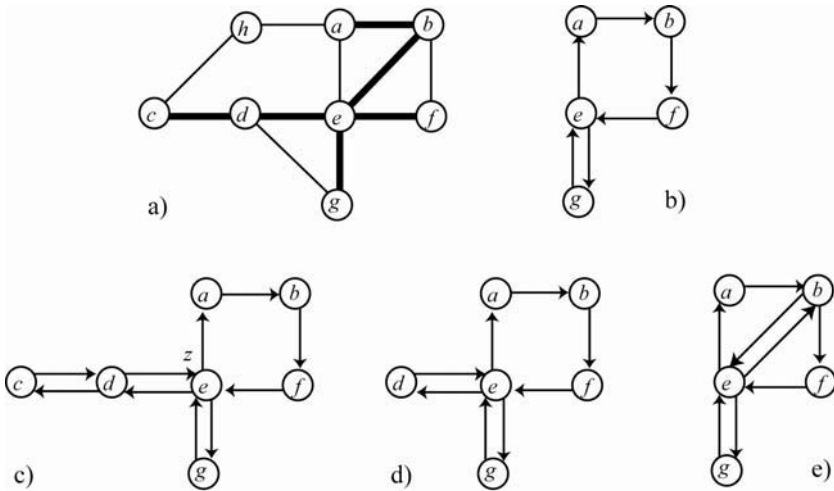


Figure 6.13. Illustration of the first step of the Add procedure: a) network with a set of clients to serve in heavy lines. All arcs have unit length; b) a tour $T = (a, b, f, e, g, e, a)$ that covers the subset $R' = \{(a, b), (e, f), (e, g)\}$ of clients; c) addition of client (c, d) . Neither c nor d appears in T . The node z in T for which the sum of the distances to c and d is minimum is node e . The new tour is $(a, b, f, e, g, e, d, c, d, e, a)$; d) addition of client (d, e) , with node e on T . The new tour is $(a, b, f, e, g, e, d, e, a)$; e) addition of client (b, e) with nodes b and e on T . The new tour is $(a, b, e, b, f, e, g, e, a)$

The third procedure is the inverse of the preceding one, and consists of removing the service from a client on an existing tour. This procedure is described in Algorithm 6.11 and is illustrated in Figure 6.14.

Let T be a tour covering a subset R' of clients. Let $(v, w) \in R'$ be a client to remove.

Remove (v, w) from R' and try to diminish the length of tour T with the help of the procedure *Shorten* described in Algorithm 6.9 (by imposing a service only on the arcs in the new set R').

Algorithm 6.11. Drop algorithm that allows us to remove a service on an arc of a tour

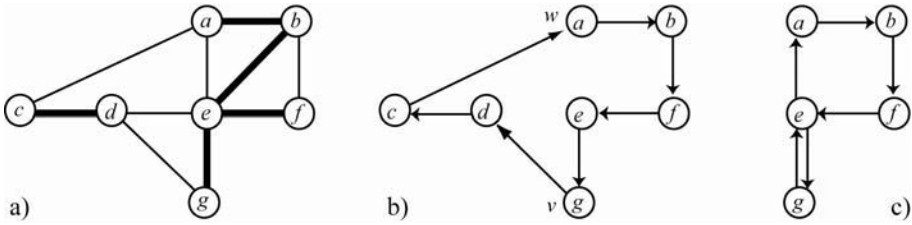


Figure 6.14. Illustration of the Drop procedure: a) network with a set of clients to serve in heavy lines. All arcs have unit length; b) a tour $T = (a, b, f, e, g, d, c)$ that covers the subset $R' = \{(a, b), (c, d), (e, f), (e, g)\}$ of clients; c) by removing (c, d) from R' and applying the procedure Shorten with $v=g$, one can reach $w=a$ and replace the path $P = (g, d, c, a)$ of length 3 with the path (g, e, a) of length 2

A solution to the rural postman problem (not necessarily optimal) can easily be obtained on the basis of the procedures described above. We can, for example, use the constructive Algorithm 6.12 proposed in [HER 99].

1. Choose a required arc (v, w) in R , and set $T = (v, w, v)$ and $R' = \{(v, w)\}$.
2. If $R' = R$, then STOP. If not, choose a client (v, w) belonging to R but not to R' , add the service of this client (v, w) with the help of the procedure *Add*, add (v, w) to the set R' of clients already serviced, and repeat step 2.

Algorithm 6.12. Algorithm for the non-oriented rural postman problem

6.3.3. Arc routing problems with capacity constraints

We again place ourselves in the context where the deliveries to make necessitate the use of several vehicles, each vehicle having only a limited capacity (see section 6.2.2). We then speak about an arc routing problem with capacity constraints. Most of the algorithms that have been developed to solve the ARP with capacity constraints have initially been described for the non-oriented Chinese postman (see section 6.3.1). In other words, these algorithms are applied to non-oriented graphs where each arc corresponds to a client. These algorithms can, however, be easily adapted to other ARPs, for example to the oriented case where certain arcs in the network require no service.

Among the constructive methods proposed for the solution of the Chinese postman problem with capacity constraints, we only describe the *Augment-Merge* algorithm proposed in 1981 by Golden and Wong [GOL 81]. This algorithm is inspired by the one developed by Clarke and Wright [CLA 64] for the NRP (see section 6.2.2). Starting with a set of tours, each one servicing only one client, the number of vehicles used is progressively reduced in two different ways. First of all, the service on certain arcs is transferred from one tour to another. Then, new tours are obtained by merging existing tours.

Let us consider two tours T_1 and T_2 so that the sum of the requests of their clients does not exceed the capacity of a vehicle. Let us note by v_1 and w_1 the first extremities of arcs to service encountered on T_1 and T_2 , respectively. Similarly, let w_1 and w_2 be the last extremities of arcs to service encountered on T_1 and T_2 , respectively. Merging T_1 with T_2 can be done in four different ways. We can, for example, traverse T_1 up to node w_1 , then go to v_2 by the shortest path, and finally traverse T_2 from v_2 to the depot. A second possibility consists of, for example, traversing T_1 up to w_1 , then going to w_2 with the help of the shortest path, and finally traversing T_2 in an inverse direction from w_2 to the depot. The two other possibilities are obtained by permuting the roles of v_1 and w_1 or of v_2 and w_2 . The merging of two tours is illustrated in Figure 6.15, and the *Augment-Merge* algorithm is described in Algorithm 6.13.

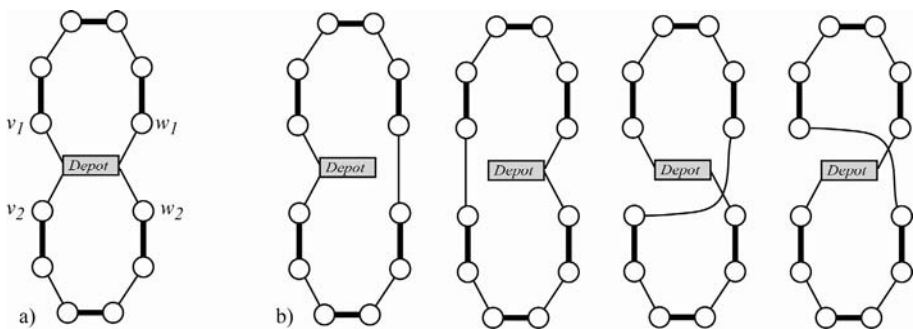


Figure 6.15. Illustration of the merging of two tours: a) two tours that we want to merge; b) the four possible merging

For each required arc (v, w) , construct a tour $T_{v,w}$ made of a shortest path of non-served arcs between the depot and v , the served arc (v, w) , and a shortest path of non-served arcs between w and the depot.

1. Starting with the longest tour $T_{v,w}$ and as long as vehicle capacity permits, change the status of traversed arcs from non-served to served, if these arcs are covered by shorter vehicle routes. Remove the shorter vehicle routes whose unique served arc is now covered by a longer route.
2. Subject to capacity constraints, evaluate all possible mergers of two tours. If the largest saving obtained by one of these mergers is positive, do the merger which yields the largest positive saving, and repeat step 3, if not STOP.

Algorithm 6.13. *Augment-Merge algorithm*

Other constructive algorithms have been proposed for the solution of the ARP with capacity constraints. A complete review of these algorithms is given in [HER 00b] and [MIT 00].

Currently, the most efficient algorithms for the solution of ARPs with capacity constraints are adaptations of Local Search techniques (see Chapter 3). Similar to what has been proposed for the NRP (see section 6.2.2), a solution s is defined as a set of tours satisfying all the client requests, but not necessarily respecting the capacity constraints. If a solution s violates the capacity constraints, a penalty $P(s)$ proportional to the excess demand with respect to vehicle capacity is calculated. The value $F(s)$ of a solution s is then obtained by adding the total distance covered by the vehicles with the penalty $P(s)$. The set $N(s)$ of neighbor solutions to s contains all the solutions that can be obtained by transferring the service of a required arc from one tour to another. To make such a transfer, it suffices to eliminate a service in a tour and add it in another. The elimination and addition of a service are done with the help of the procedures *Add* and *Drop* described in section 6.3.2. Such an approach has been used with success in [HER 00a].

6.4. Conclusion

To transport raw materials, people or information, to assure services at the clients, do maintenance work on the road networks or on the electricity lines, etc, all these activities are the daily work of numerous companies. They cannot be competitive without good management of the resources allotted to these types of problems. Operations research has the techniques to treat certain aspects of these problems. The objective of this chapter was to describe the principal heuristic methods used to solve vehicle routing problems where the clients are located either on the nodes (problem NRP) or on the arcs (problem ARP) of a network. Any reader who is interested in knowing more about the models and solution techniques of

vehicle routing problems is invited to consult the reference works [LAW 85] and [TOT 02] for the NRP and [DRO 00] for the ARP.

The variants of the NRP and the ARP that we have analyzed in this chapter are basic vehicle routing problems. When we must treat real routing problems, additional constraints must be taken into account (limit on length or duration of tours, different categories of routes, multiple depots, etc), and the algorithms that we have described must then be adapted or extended since they can not be directly applied to such problems. The algorithms described in this chapter must be considered as skeletons of more specialized algorithms to be designed for each particular real life problem.

6.5. Bibliography

- [AAR 97] E. Aarts and J.K. Lenstra, *Local Search in Combinatorial Optimization*, Wiley, West Sussex, 1997.
- [BAL 88] M.O. Ball and M.J. Magazine, "Sequencing of insertions in printed circuit board assembly", *Operations Research*, vol. 36, 192–201, 1988.
- [CHR 76] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem" Research report no. 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, 1976.
- [CHR 85] N. Christofides, "Vehicle routing", Chapter 12 in [LAW 85], 431–448, 1985.
- [CHR 86] N. Christofides, V. Campos, A. Corberán and E. Mota, "Algorithms for the rural postman problem on a directed graph", *Mathematical Programming Study*, vol. 26, 155–166, 1986.
- [CLA 64] G. Clarke and J. Wright, "Scheduling of vehicles from a central depot to a number of delivery points", *Operations Research*, vol. 12, 568–581, 1964.
- [DRO 00] M. Dror, *ARC Routing: Theory, Solutions and Applications*, Kluwer Academic Publishers, 2000.
- [EDM 73] J. Edmonds and E.L. Johnson, "Matching, Euler tours and the Chinese postman", *Mathematical Programming*, vol. 5, 88–124, 1973.
- [FRE 79] G.N. Frederickson, "Approximation algorithms for some postman problems", *Journal of the ACM*, vol. 26, 538–554, 1979.
- [GEN 94] M. Gendreau, A. Hertz and G. Laporte, "A tabu search heuristic for the vehicle routing problem", *Management Science*, vol. 40, 1276–1290, 1994.
- [GOL 81] B.L. Golden and R.T. Wong, "Capacitated arc routing problems", *Networks*, vol. 11, 305–315, 1981.

- [GOL 85] B.L. Golden and W.R. Stewart, “Empirical analysis of heuristics”, Chapter 7 in [LAW 85], 207–249, 1985.
- [GUA 62] M. Guan, “Graphic programming using odd and even points”, *Chinese Mathematics*, vol. 1, 273–277, 1962.
- [HER 99] A. Hertz, G. Laporte and P. Nanchen, “Improvement procedure for the undirected rural postman problem”, *INFORMS Journal on Computing*, vol. 11, 53–62, 1999.
- [HER 00a] Hertz A., Laporte G., Mittaz M., “A Tabu search heuristic for the capacitated arc routing problem”, *Operations Research*, vol. 48, p. 129–135, 2000.
- [HER 00b] A. Hertz and M. Mittaz, “Heuristic algorithms”, Chapter 9 in [DRO 00], 327–386, 2000.
- [JOH 97] D.S. Johnson and L.A. McGeogh, “The traveling salesman problem: a case study”, Chapter 8 in [AAR 97], 216–310, 1997.
- [LAW 85] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester, 1985.
- [MIN 79] E. Minieka, “The Chinese postman problem for mixed networks”, *Management Science*, vol. 25, 643–648, 1979.
- [MIT 00] M. Mittaz., Problèmes de Cheminements Optimaux dans les Réseaux avec Contraintes Associées aux Arcs, PhD Thesis no. 2103, Department of Mathematics, Ecole Polytechnique Fédérale de Lausanne, Switzerland, 2000.
- [ORL 74] C.S Orloff, “A fundamental problem in vehicle routing”, *Networks*, vol. 4, 35–64, 1974.
- [TOT 02] P. Toth and D. Vigo , *The Vehicle Routing Problem*, SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002.

Chapter 7

Telecommunication Networks

7.1. Introduction

Telecommunication services take on more and more importance in our society driven by information technology (IT). Numerous actors are entering this sector and mastery of costs and service quality are crucial. To satisfy client needs, an operator must offer telecommunication services of excellent quality at the right price. The main element of the quality of services in a network is its availability. Through the transport and treatment of information that they bring to our disposal, networks have become a must in people's lives and in company economics. Damages caused by disruption in telephony or data services can be very important, and can have disastrous consequences for the brand image of an operator. In this context, the construction of safe and reliable networks under all circumstances is an indispensable criterion for the credibility of operators offers. In this chapter we will show how to build high-quality networks in the case of both fixed and wireless networks.

To ensure the availability in large telecommunications networks, one must maintain service continuity even in case of an element breakdown. The main causes of breakdowns are cable cuts due to civil engineering work, malfunctioning of optical transmission components or multiplexing equipment failure. Even if the probability of these events remains low, their consequences are not acceptable. Interruption of the access to telecommunication services can be very damaging for some private or public entities (banks, hospitals, etc). Their tolerance for a cutoff is

no longer than one second for the telephone service, and only a fraction of second (some tens of milliseconds) for data transfer. Modeling and optimizing the dimensioning and the routing in fixed networks in case of breakdown will be the subject of the first part of this chapter.

In mobile communications systems, availability essentially consists of getting rid of spatial constraints to communication. The flow of people, goods and services is accompanied by an increased need for nomad communications. However, wireless has for a long time carried a negative image due to covering defaults or bad quality transmission. Since the emergence of cellular numeric systems, customers (whether professional or individual) are increasingly reluctant to accept not having a fully available network no matter where they are located. Management of coverage and overload of the spectrum then become key elements in the success of wireless mobile operators. Modeling and optimization problems to obtain quality wireless communication will be the object of the second part of this chapter, which is dedicated to the conception of cellular networks.

7.2. Optimal synthesis of backbone networks

To maintain the continuity of services even when equipment fails, the operator must build reliable networks. Moreover, to stay competitive it must build them at the lowest cost. To do that, operators must put in place in their networks search procedures for alternative routes in order to reroute traffic demands in every potential failure. For economic reasons and also because of the low probability of breakdowns, the operator will build and dimension networks to protect them only against a single breakdown: for example a cable cut, a transmission system or multiplexing equipment failure. In effect, the increase of the cost of the network to protect it against multiple breakdowns would not be justified compared to the risks involved. It already reaches 30% to 50% of the cost of the network when protection is against a single breakdown. Some operators, wanting to make aggressive commercial offers, neglect to secure their network and it is the client who must then protect himself against potential service unavailability through multiple connections to different operators. Usually that is a double connection.

Optimal network synthesis consists of determining the topology, the dimensioning of the capacities for links and nodes, and defining the routings and reroutings of demands. It has become an important application in the field of operations research. Its strategic stake has generated a large number of academic and practical works on the subject. Numerous references are available in international journals of optimization and operations research. For the classical optimal synthesis problem of a backbone network, we will show how different optimization

techniques, such as greedy heuristics, metaheuristics, linear programming and the decomposition method, can be advantageously used.

Backbone networks are at the core hierarchical level of a telecommunication network. They carry all telecommunication services for operators, and also for large companies or governmental institutions. They are partially meshed and they connect among them traffic sources and traffic destinations. Company backbone networks are small or medium size: typically around ten nodes. Those built by operators are often much larger: hundreds up to a few thousand nodes. Figure 7.1a gives the example of a network with 13 vertices and 16 links. Point-to-point traffic demands between vertex pairs are deterministic. We do not take into account the probabilistic nature of the traffic with all its consequences on the quality of services: blocking probability in case of circuit mode networks or transfer delay in case of packet mode networks. A traffic demand flows on a path connecting its origin and destination. Such a path is called a routing. Figure 7.1a shows two routings between vertex pairs (A, B) and (D, C). If a network element (a vertex or a link) breaks down, the traffic demands whose routing used this failed element are interrupted. To maintain reliability and service, it is then necessary to use alternate paths called reroutings for these demands. Figure 7.1b shows two examples of rerouting for the failure of link (I, J): a rerouting path for the demand (A, B) and two for the demand (D, C). Rerouting of demands for different breakdown situations necessitates additional capacities, also called spare capacities, on links and vertices. Capacity needs to route traffic demands in the nominal network (without breakdown) are called nominal capacities.

The best way to plan an *optimum* backbone network would be to be able to simultaneously optimize the topology and dimension the nominal and spare capacities. In that case, routing of traffic demands and rerouting in case of breakdown would be deduced from these dimensions. In fact this problem is extremely difficult to solve and few works consider this global approach. In [POP 96], the authors consider the problem for small size networks (6 and 9 vertices). Some studies [IRA 98, MUR 98, LUT 98] simultaneously dimension the nominal network and the spare capacities, but suppose the network topology to be known. We note that certain studies [GRÖ 95] only take into account topology and connectivity aspects while neglecting the traffic aspect.

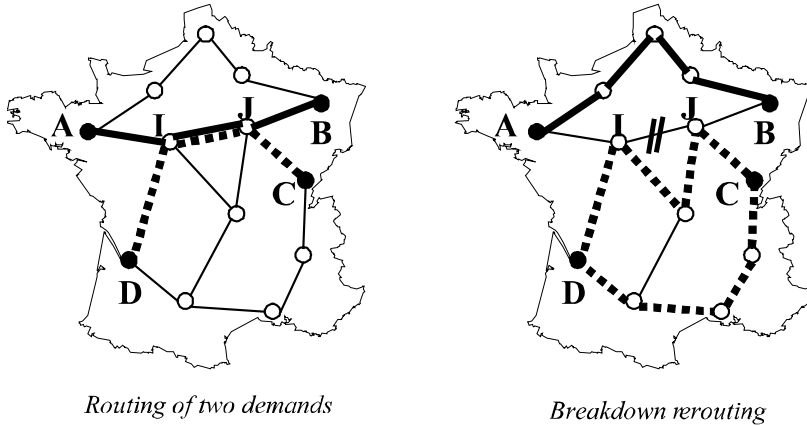


Figure 7.1a and b. *Example of a backbone network*

To obtain operational results on networks of significant size (several dozen vertices), optimization is addressed in two steps. The first step determines the topology, the dimensioning and the routing of the demands that optimize the cost of the nominal network. The nominal network is then known and definitively fixed; the second step defines the optimal dimensioning of the spare capacities necessary to reroute all the traffic demands disturbed in the case of single breakdown of a network element. This approach presents some disadvantages. The optimization of the nominal network in the first step often leads to few meshed networks because of a strong concavity of the cost function of network equipment as optical transmission systems.¹ Without multiple connectivity constraints on their topology, optimal networks obtained by this first step would very often be trees or quasi-trees. The second step, on the other hand, needs a strong diversity of paths in the graph in order to minimize the additional quantity of resources to be installed. The graph of the network must also be 2-connected without an articulation point to guarantee the existence of at least one alternate path for each demand disturbed by a single breakdown of a link or vertex. The opposition appears obviously between these two optimization processes and mostly explains the difficulty of the global approach. This conflict of interest between the two steps can lead to suboptimal solutions. It is what motivates and makes research work necessary to develop efficient solution methods for the global problem in only one step.

To solve the first step, we present two approaches. In section 7.3.1, we describe a “demeshing” greedy heuristic proposed in [MIN 76] that judiciously exploits the theoretical properties of the problem if the cost function of links are concave (sub-modular to be exact). In section 7.3.2, we will show that a metaheuristic like

¹ We can roughly assume that the cost of a link only doubles when its capacity quadruples.

simulated annealing can also exploit this property. Moreover, we will see that the flexibility of a metaheuristic lets us take into account numerous additional constraints encountered in practical problems. Other methods are of interest to us, but that would make us exceed the scope of this chapter. We can, for example, note a primal-dual solution method proposed in [BAL 89a], as well as Lagrangian relaxation techniques in [BAL 89b]. We can refer to [MIN 89] for a more complete panorama of the different possible approaches.

For the second step, optimal dimensioning of spare capacities is generally addressed by linear programming. In section 7.4.1, we introduce non-simultaneous multiflow models. We will show how generalized linear programming methods with path generation can be used to solve them. However, linear programs engendered by these models are soon too big to be solved directly when the network exceeds about 30 vertices. In section 7.4.2, we explain the Benders' decomposition technique [BEN 62] that lets us manage network instance size up to 100 vertices or more without difficulty. In section 7.4.3, an extension of this decomposition technique will be introduced for the case of modular spare capacities. In practice, the treatment of modular capacities is primordial. In effect the network equipment never has continuously defined capacity values. Capacities of optical transmission systems can be 1x155Mbit/s, 4x155Mbit/s, 16x155Mbit/s or 64x155Mbit/s and more. In addition, we can of course multiply the number of systems on each network link. Often the staircase function for the cost of this equipment also has a concave trend (their average unitary cost decreases) as illustrated in Figure 7.2.

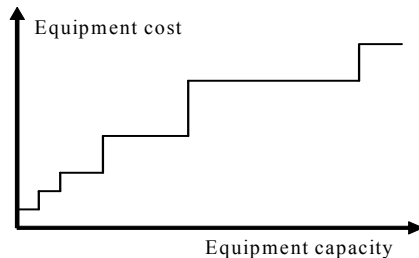


Figure 7.2. *Illustration of a rising modular cost function of equipment*

7.3. Topology and dimensioning of the nominal network

The solution to this problem has been the object of substantial literature. For general cost functions, it is particularly difficult. Solving it exactly raises interesting theoretical questions, but it remains limited to small networks: at most a dozen links. The exact approaches [MAG 95, GAB 99, SRI 00] generally use branch and bound

or branch and cut methods. To manage operational instances comprising more than 50 vertices and several hundred or even thousand potential links, we must use heuristics. For the case of general cost functions, the conception and development of an efficient and rapid heuristic is difficult. In fact, even in knowing the topology and dimensioning of a network, the computation of good routings for all traffic demands requires solving a large size linear program [AHU 93]. It is then difficult to repeat this calculation in an iterative heuristic for a large number of configurations of different topology and dimensioning without seeing the calculation time explode.

Some specific instances of this problem are less difficult and it is possible to develop efficient methods to solve them. These are instances without capacity limit for which the cost functions on the links are concave. In [YAG 71, ZAD 73] some of the first developed methods are proposed. In fact, these instances have a particularly interesting property. For a given topology, the network of minimum cost is such that traffic demands are mono-routed on the shortest paths in the sense of the marginal costs of the links. In the case of cost functions including a fixed fee plus a linear cost depending on the volume, this marginal cost is the linear variable cost. This very strong property constitutes the foundation of numerous heuristics: for example the greedy demeshing algorithm (a descent algorithm) and the simulated annealing (a metaheuristic) that we will now present.

7.3.1. Descent algorithm

The problem considered consists of determining the topology of a network and the dimension of its links while minimizing its cost when the cost of a link (i, j) between two vertices i and j is decomposed in a fixed installation cost F_{ij} and a usage cost c_{ij} linear with the volume of traffic passing on the link as illustrated in Figure 7.3. The underlying graph to construct the network will be noted $G = (X; E)$, where X is the set of vertices and E the set of possible links (i, j) . We will call D the set of point-to-point traffic demands that must flow through the network.

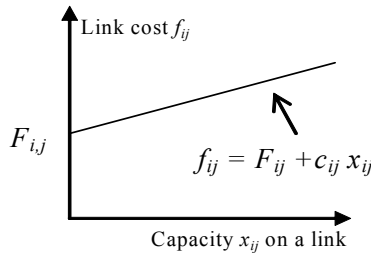


Figure 7.3. Cost function on a link

From a complete or partial supporting graph (some links can be forbidden *a priori*), the greedy demeshing heuristic successively suppresses links while the value of the objective function decreases. At each iteration, the objective function of the current configuration is compared to all different configurations in which a link is deleted. This comparison is made by a computation of the set of shortest paths that define the routings of demands. Routing the set of demands on these shortest paths is sufficient to obtain the dimensioning of the links and consequently the cost of the network.

By using the notations previously introduced, the cost function $f_{i,j}$ for each link $(i, j) \in E$ is defined in the following manner:

$$\begin{cases} f_{i,j}(0) = 0 & \text{if } x_{i,j} = 0 \\ f_{i,j}(x_{i,j}) = c_{i,j}x_{i,j} + F_{i,j} & \text{if } x_{i,j} > 0 \end{cases}$$

where $x_{i,j}$ is the volume of traffic flowing on the link (i,j) . If $f(x)$ is the value of the objective function associated with a flow vector x on the set of possible links of E , then:

$$f(x) = \sum_{(i,j) \in E} f_{i,j}(x_{i,j})$$

At an iteration t of the heuristic, E^t gives the set of links still available and x^t represents the flow vector on links. Let us consider a link (k, l) of E^t joining the vertices k and l . We can determine the chain of minimum length $L'_{k,l}(x^t)$ joining the extremities k and l in the graph $(X, E^t \setminus \{(k, l)\})$ (graph without link (k, l)). This length is calculated by valuing the set of links (i, j) of the graph with the unitary linear cost values $c_{i,j}$. It can be interpreted as the cost induced by the rerouting of one flow unit previously passed on the link (k, l) in the graph $(X, E^t \setminus \{(k, l)\})$. The deletion of the link (k, l) also induces a saving of $f_{k,l}(x^t_{k,l})$. It is then interesting to suppress the link (k,l) if:

$$\Delta_{k,l} = x^t_{k,l} L'_{k,l}(x^t) - f_{k,l}(x^t_{k,l}) < 0.$$

If $\Delta_{k,l}$ becomes positive or zero for all the links (k, l) of E^t , the demeshing algorithm stops. For the first step ($t=0$), all links are possible ($E^0 = E$). The initialization of flows x^0 consists of routing traffic demands on the shortest paths calculated in the sense of unit costs c_{ij} .

Initial solution: (x^0, E^0)

At iteration t , the current solution is (x^t, E^t)

1. for all links $(k, l) \in E^t$ so that $x_{k,l}^t > 0$, calculate $L_{k,l}^t(x^t)$, the shortest path between k and l on the partial graph G induced by $E^t - \{(k, l)\}$ supplied with the costs $c_{k,l}$ on the links. Then calculate $\Delta_{k,l} = x_{k,l}^t L_{k,l}^t(x^t) - f_{k,l}(x_{k,l}^t)$
2. determine the link (k^*, l^*) so that $\Delta_{k^*, l^*} = \underset{(k,l) / x_{k,l}^t > 0}{\text{Min}} \{\Delta_{k,l}\}$
3. if $\Delta_{k^*, l^*} \geq 0$ then **Stop**, the current configuration cannot be improved
4. let Λ^* be the chain associated with the value $L_{k^*, l^*}^t(x^t)$,

$$\begin{cases} x_{i,j}^{t+1} = x_{i,j}^t & \text{if } (i,j) \notin \Lambda^* \\ x_{i,j}^{t+1} = x_{i,j}^t + x_{k^*, l^*}^t & \text{if } (i,j) \in \Lambda^* \\ x_{k^*, l^*}^{t+1} = 0 \end{cases}$$
5. $E^{t+1} = E^t - \{(k^*, l^*)\}$
6. $t = t+1$ and return to step 1.

Algorithm 7.1. Demeshing algorithm

In the worst case this demeshing algorithm has a complexity in the order of $O(n^5)$, n being the number of vertices in the network. For each potential link (k, l) ($\sim n^2$ links exist in the complete graph), it is necessary to calculate the shortest path between the vertices k and l . In practice, the implementation of the algorithm is a little bit more complex than this short description. In effect it is necessary to take into account possible cycles appearing in the network after the reroutings of the traffic demands that follow a link deletion. It is also possible to improve the speed of the algorithm by avoiding the recalculation of all the shortest paths between all link deletion attempts.

This demeshing heuristic gives excellent results: the solutions obtained are very close to the optimum. When the size of the network increases, it can however

become relatively slow because many links must be suppressed. In addition, like for all greedy algorithms, it cannot backtrack after a bad choice. While running, the algorithm can wrongly decide to take off a link. That is why it can be judicious to iterate between trials of adding and subtracting links at the end of the procedure to improve this descent heuristic and avoid being caught in suboptimal local minima.

7.3.2. *Simulated annealing*

Each iteration of a descent heuristic like the one previously described takes for a new current configuration, that of the lowest cost in its neighborhood. This procedure leads to a local minimum, that is a configuration whose cost is less than all of its neighboring configurations. To avoid being caught in a local minimum too far from the optimum, some methods called metaheuristics exist. Simulated annealing, tabu search, the GRASP and genetic algorithms are the most well-known metaheuristics. Each of these methods uses specific mechanisms to escape from local minima. We will stay for a moment with one of these methods: the simulated annealing [KIR 83] (we refer the reader to Chapter 3 for a more precise description of this method). With simulated annealing we escape from local minima by allowing transitions to higher cost configurations. This intermediate degradation of the objective function is controlled by a control parameter called temperature.

Like descent heuristics, metaheuristics use transformation operators of the current configuration; in fact they often use the same ones. These operators determine the neighborhood of the current configuration: the set of configurations accessible by all possible applications of the operator. In a metaheuristic it is sometimes useful or even imperative to put several operators into action. For the optimal synthesis of a backbone network the above-described greedy search uses the suppression of a link as operator (the deletion operator). For simulated annealing we will again use this operator and we will also use the addition of a link as second operator. We can also use the link exchange operator that is none other than the concatenation of the two previous operators: the link addition and the link deletion.

Even if it is theoretically possible to reach the optimal solution of a problem, the simulated annealing algorithm in practice converges toward a suboptimal configuration. In order to guarantee a convergence toward the absolute minimum would need an exponential number of iterations [HAJ 88], which is of course not conceivable. We use the classical annealing scheme that foresees a geometric decrease of the temperature. The simulation proceeds in temperature steps for which a certain number of transformation trials is carried out. The simulation is stopped when hopes of improvement become too weak; for example, if the rate of acceptance for higher cost configurations becomes too small (in the order of $1/1,000$).

To obtain good results the number of temperature steps necessary in an annealing scheme is between 20 and 50 with a geometric decrease of the temperature by a multiplicative coefficient α of about 0.9 between each step. How to choose the initial temperature was the object of numerous discussions. Some prefer a very high initial temperature in order to accept almost all the transformation trials at the beginning of the search. From a theoretical point of view that is correct. However, as the objective is to obtain the best possible solution in a limited CPU time, it is useless to begin with too high an initial temperature. The iterations done at high temperatures are in fact spoiled and would be much more useful at a lower temperature at the end of the simulated annealing process. As a general rule, an initial temperature so that 30% to 50% of the transformation trials are accepted is largely sufficient to avoid the annealing getting caught around the initial configuration and not being able to explore the set of pertinent configurations. If the tuning of parameters, number of steps and initial temperature pose no difficulty, the number of iterations by temperature step is more delicate to define. It essentially depends on the CPU time that we want to allow for the optimization process or on the quality of the solution that we wish to obtain. Indeed, the greater the number of iterations, the greater the hope of obtaining the best solution. However we must pay careful attention to the fact that this is only true on average: for a given simulation we can obtain a best solution with fewer iterations and vice versa. Let us note that the sharing of iterations among the set of temperature steps also affects the quality of the solution. An equitable dispatching between all steps gives satisfactory results. It is however better to increase the number of iterations when the temperature decreases. Thus for the same total number of iterations, a progression from 5% to 10% of the number of iterations between steps improves (on average) the results obtained with a uniform dispatching.

A simulated annealing algorithm performs a great number of transformations of the current configuration. We must consequently calculate very many times the cost difference between the trial configuration and the current one. In the case of optimal network synthesis, the transformation operators used are the suppression and the addition of a link. The choice of the link to suppress (or to add) is made by chance among the links present (or absent) in the current configuration. The deletion operator must also watch to maintain network connectivity. Determining the cost difference brought about by the addition (respectively the deletion) of a link can be roughly calculated as the difference between the cost of the trial solution and the one of the current configuration. The complexity of this computation is essentially defined by the computation of the shortest paths that give the routing of the demands in the network. It results in a complexity in $O(n^4)$. To determine this cost difference, it is in fact more efficient to compute these shortest paths incrementally. The calculation of the set of shortest paths is realized once for the initial configuration. Then we only recalculate changes in the shortest paths brought about by the addition or the suppression of a link. Thus, we can define the set of demands which change

their routing, and consequently we recalculate the difference of cost only for the links concerned by these changes. For weakly meshed networks, the number of shortest paths modified by the addition or the suppression of a link remains low, especially for large networks ($\gg 50$ vertices). For a network with 100 links, the number of links that see a change in their passed traffic is at most one or two dozen and the number of modified shortest paths remains extremely low. The determination of the cost difference by an incremental calculation of the shortest paths is thus extremely rapid. This implies that we have the possibility of doing a large number of iterations with the simulated annealing in a reasonable computation time.

The main advantage that explains the success of metaheuristics like simulated annealing is their flexibility to handle practical problems. In relation to other optimization techniques, such as linear programming, Lagrangian relaxation, etc, it is in effect easy to modify the objective function and the constraints of the problem. We can even envisage adding new ones. For optimal network synthesis, that means numerous possibilities for extensions. We can for example guarantee the diameter or the degree of the network connectivity. For that it suffices to forbid transformations (addition or suppression of a link) that would lead to configurations that do not verify the diameter or connectivity required. Using an identical mechanism we can force or forbid the presence of some links in the solution. It is also possible to use this flexibility to take into account more complex cost functions, for instance staircase functions. For that we only have to consider two cost systems while running the algorithm: on one hand, the real link cost function that is used to calculate the cost difference between the trial and current configurations in the annealing decision process and, on the other hand, a set of linearized costs with which the shortest routing paths are determined. These linearized costs are naturally deducted from the real costs and readjusted periodically. This approach is of course heuristic. Strictly speaking, the property of mono-routing on the shortest paths in the sense of the marginal costs no longer holds for step cost functions. However, from an operational point of view, maintaining the suboptimality property of routings (any sub-routing of an optimal routing is also an optimal routing) can be useful. Let us remember that this suboptimality is guaranteed if the routings use shortest paths. In effect, the routing command protocol running on vertex equipment does not necessarily know how to achieve routings that would not respect suboptimality. It is notably the case when they use routing tables that only depend on the destination and not on the origin of the traffic demands; IP routers are a perfect example of this.

In the same way, we may also consider vertices as costly elements. Of course the more constraints and supplementary options we add, the farther we are from the rigorous academic framework that overextends the initial algorithm. Consequently we take risks to obtain a less efficient heuristic. For example, for link costs with a strong economy of scale and where we force the 2-connectivity of the network

topology (by forbidding deletion of links which would lead to non-2-connected networks), we have a good chance of obtaining an “optimal” network with unused links which only serve to guarantee the 2-connectivity. The optimal network configurations obtained in these situations are no longer necessarily pertinent. As they are unused, the algorithm has added to a tree-like configuration a set of least cost links leading to the satisfaction of the connectivity constraints. Unfortunately, these links artificially added by the algorithm could be totally inadequate!

7.4. Dimensioning of spare capacities

After optimizing the nominal network, we will now attempt to dimension the reserve network. This problem is frequently considered in the literature when the topology and the dimensioning of the nominal network are known. Numerous variants of this problem are possible depending on the strategies and options of rerouting used in case of breakdown. The rerouting of traffic demands can be local between the origin and destination vertices of the link that fails or from end-to-end between the origin and destination vertices of the interrupted demands. Other options can intervene: maximum length of reroutings, the possibility (or not) of reusing the capacities liberated by traffic demands interrupted by the failure, etc.

We can also consider cost functions for the links that are continuous or of the staircase type. In [GRO 91, SAK 90], the authors solve the problem of dimensioning spare capacities for a local rerouting strategy and continuous capacity variables. [HER 94, LUT 00, POP 98] handle the problem in continuous variables for local and end-to-end rerouting. In [GOE 93] and [LEE 95], the local rerouting problem is addressed with modular capacities. A model with integer variables and with the rerouting of all the demands in case of breakdown is presented in [STO 94]. Heuristics are also presented in [GRO 91] to solve the problem in integer variables.

7.4.1. Non-simultaneous multiflow model and linear programming

Usually we model the spare capacities dimensioning problem as a non-simultaneous multiflow in a graph, which can be precisely solved with linear programming. These linear programs consider two classes of constraints. The first expresses capacity constraints on network resources. To pass traffic demands, we cannot use more resources on the links (resp. vertices) than the resource capacities available on these links (resp. vertices). The second handles the necessity of passing all or part of the interrupted demands.

There are two possible formulations to describe a multiflow model. They differ in the definition of decision variables and in the notation of traffic constraints that follow. In the first formulation, called edge-vertex, the variables of the model are the quantities of traffic flowing on a link for a given demand. The flow for the demand is thus represented by Kirchoff's constraints of flow conservation. For a vertex and a demand, the sum of entering traffic quantities is equal to the sum of departing traffic quantities. It is of course necessary to take into account the sources and sinks of the traffics. For a link, the capacity constraint is expressed according to the sum of all flows passing through this link.

The other formulation, called edge-path, uses variables that represent the flow of traffic passed on a path used to route the whole or part of a traffic demand. The flow constraints indicate that the sum of traffic flow passed on the set of all possible paths for a given traffic demand must be equal to the whole or part of this demand. The second formulation has the advantage of comprising far fewer flow constraints. There is one constraint per traffic demand, while in edge-vertex formalism the number of Kirchoff's constraints is equal to the product of the number of vertices multiplied by the number of traffic demands. The number of flow variables in edge-vertex formalism is equal to the product of the number of traffic demands multiplied by the number of links, whereas it is much larger in the edge-path formalism. The number of possible paths to pass a traffic demand can be very large and, as a general rule, we cannot formulate all paths in a linear multiflow program. Generalized linear programming allows us to bypass this disadvantage. To solve the multiflow linear program (LP), at first we only consider at a subset of possible paths, then, as we go along, we insert new paths that prove to be useful.

Notwithstanding that edge-path models do not give a fully defined description of the multiflow, they are much easier to solve. Moreover, they are much better adapted to consider additional constraints on the nature and propriety of routing paths: maximum length in number of sections, forbidden path, etc.

We will now use a multiflow edge-path model to optimize the dimensioning of spare capacities in a backbone network. We will only consider single link breakdowns, and the rerouting strategy of interrupted demands will be end-to-end between the origin and destination of the interrupted demand. Let us point out that the nominal capacities released by interrupted demands in the breakdown cannot be used for the reroutings. We will first formulate the model with continuous capacity variables.

First of all, let us introduce the notations that will be useful for us:

- let $G^0 = (X^0; E^0)$ be the graph of the nominal network and L the set of possible breakdowns. We will note by $G^l = (X^0; E^l)$ the graph of the network after the

breakdown l . For simplification reasons, here we will only consider single link breakdowns. With these conditions we have $E^l = E^0 \setminus \{l\}$;

– $R \in \mathfrak{R}^{|E^0|}$ defines the vector of spare capacities on links installed in the network and $p \in \mathfrak{R}^{|E^0|}$ is the vector of unit costs for these capacities;

– D^l is the set of traffic demands perturbed by the breakdown l (T_k is the volume of the demand k) and $H^l(k)$ gives the set of possible rerouting paths for the traffic demand k in graph G^l ;

– for a demand k and for the breakdown l , the variable $x_{k,j}^l$ gives the value of traffic passed on the rerouting path j .

The dimensioning of spare capacities is in fact modeled like a non-simultaneous multiflow. Each multiflow corresponds to the reroutings of demands disrupted by each breakdown l . The mathematical model of the problem is expressed in the form of the following linear program LP :

$$(LP) \left\{ \begin{array}{ll} \min \sum_{e \in E^0} p_e R_e & \\ \forall l \in L & \sum_{j \in H^l(k)} x_{k,j}^l = T_k \quad (\pi_{l,k}) \quad (I) \\ \forall k \in D^l & \\ \forall l \in L & - \sum_{\substack{k \in D^l, j \in H^l(k) \\ \text{with } e \in j}} x_{k,j}^l + R_e \geq 0 \quad (\omega_{l,e}) \quad (II) \\ \forall e \in E^l & \\ x_{k,j}^l \geq 0, R_e \geq 0 & \end{array} \right.$$

For each breakdown l , constraints (I) force the flow of the disrupted demands $k \in D^l$ on the set of possible rerouting paths. Constraints (II) represent capacity constraints available on links for reroutings. For each breakdown l and for each link $e \in E^l$, the sum of all the traffic flows passed on rerouting paths that use link e must be less than or equal to the value of the spare capacity installed on link e . We will note by $\pi_{l,k}$ and $\omega_{l,e}$ the dual variables that correspond respectively to the flow constraints (I) and to the capacity constraints (II). Let us note that the multiflows for different breakdowns are only coupled by the spare capacity variables R_e . Of course, the objective function expresses the total cost of spare capacities that must be installed.

It is possible to solve LP directly using generalized linear programming. As all possible rerouting paths $j \in H^l(k)$ for a demand $k \in D^l$ cannot be included, we will

solve LP iteratively. At an iteration we only take into account a subset of possible paths. After solving this restricted size linear program LP , we determine for all the demands $k \in D^l$ if there is a better candidate path for their rerouting. That consists of verifying whether there exists or not a path with a negative reduced cost for this demand among all the paths $j \in H^l(k)$ not already present in LP . Let us remember that the reduced cost of a rerouting path j for a demand $k \in D^l$ takes the form:

$$\delta_{k,j}^l = \sum_{e \in j} \omega_{l,e} - \pi_{l,k}$$

It is very easy to determine the path j that minimizes this reduced cost. It suffices to provide the links e of the graph E^l with the weights $\omega_{l,e}$ and to calculate the shortest path between the origin and the destination of the demand $k \in D^l$. If the length of this shortest path is strictly smaller than the value of the dual variable $\pi_{l,k}$ then this shortest path has a negative reduced cost and must be retained for rerouting demand k . If it is longer, there is no best path candidate for rerouting demand k for this iteration. It then suffices to remove the rerouting demands for all the breakdowns, to insert in LP the set of negative reduced cost paths retained, and then solve LP again. We obtain a new intermediary solution for which we can start again the process. When there is no longer any demand for a path with negative reduced cost, the solution of the current iteration of LP is optimal. This path generation technique lets us solve LP directly. We can however only solve instances of small size. As soon as it exceeds 40 vertices and 100 links with about 500 traffic demands, getting the optimal solution becomes much more difficult. We note a slower and slower convergence of the process and the number of generated paths explodes.

For instances of larger size networks, we must use another more competitive optimization technique, for example a decomposition technique that makes the most of the particular structures of linear programs that model non-simultaneous multiflows.

7.4.2. Solution of non-simultaneous multiflows by decomposition

Non-simultaneous multifold linear programs have very particular structures. For what concerns traffic flow constraints (I), there is complete independence between the different breakdowns considered. On the other hand, capacity constraints (II) are coupled by capacity variables R_e . For such a structure of the matrix of constraints, Benders' decomposition is applicable.

The objective of Benders' decomposition consists of separating the optimization of coupling variables R_e from that of other variables describing the linear independent sub-programs. The decomposition of LP leads to two levels of linear

programs: a master program that determines the spare capacities on the links and satellites that define the rerouting paths for all disrupted demands for each breakdown. In fact, Benders' decomposition is an iterative method. The master program calculates a vector of spare capacities contingent on the information available at the current iteration. Each satellite program will then verify that this capacity vector allows the rerouting of interrupted demands for the breakdown it handles. If these capacities do not allow the reroutings of all perturbed demands, the multiflow is non-admissible; then an admissibility condition on the vector of spare capacities is calculated. This condition is then provided to the master program in the form of an additional constraint so that it takes it into account in its later optimizations of the capacity vector. The optimal solution is reached when a capacity vector supplied by the master is admissible for all the satellites considering the different breakdowns. Each satellite then gives the rerouting paths used and the values of traffic passed on each of them. From a mathematical point of view, each of the $|L|$ satellite programs is determined by fixing the values of capacity vector R in LP . We obtain for a breakdown l , the following program P^l :

$$(P^l) \begin{cases} \text{Min } \varepsilon_l \\ \forall k \in D^l & \sum_{j \in H^l(k)} x_{k,j}^l = T_k & (\pi_{l,k}) \\ \forall e \in E^l & - \sum_{\substack{k \in D^l, j \in H^l(k) \\ \text{and } e \subset j}} x_{k,j}^l + \varepsilon_l \geq -R_e & (\omega_{l,e}) \\ \varepsilon \geq 0, x_{k,j}^l \geq 0 \end{cases}$$

Here we face a multiflow admissibility program. The first constraints force the passage of the perturbed demands $k \in D^l$. The seconds correspond to the capacity constraints. As for the optimization of the global multiflow program LP , we use a technique of generalized linear programming by generating paths to solve each of the satellite programs P^l . We will use the dual variables $\pi_{l,k}$ and $\omega_{l,e}$ to calculate the pertinent rerouting paths for the demands $k \in D^l$.

The variable and objective function ε_l determines the admissibility or not of the multiflow for a capacity vector R . If ε_l^r is negative or zero for the optimal solution of P^l , this indicates that the capacities on the links are sufficient to reroute the perturbed demands. However, if ε_l^r is strictly positive, it is necessary to add capacity to the links. The condition of admissibility that we will give to the master

program consists of forcing a negative or zero value for ε_l^r . To do that, it suffices to notice that due to the duality, the optimal solution of P_l is such that:

$$\varepsilon_l^r = \sum_{k \in D^l} \pi_{l,k}^r T_k - \sum_{e \in E^l} \omega_{l,e}^r R_e$$

where $\omega_{l,e}^r$ and $\pi_{l,k}^r$ are the values of dual variables for the optimum of P_l . To force a negative value of ε_l^r , it suffices then to add to the master program the following constraint on the capacity vector:

$$\sum_{e \in E^l} \omega_{l,e}^r R_e \geq \sum_{k \in D^l} \pi_{l,k}^r T_k .$$

Under these conditions the master program PM is written as:

$$(PM) \begin{cases} \text{Min} \sum_{e \in E^0} p_e R_e \\ \forall l \in L & \sum_{e \in E^l} \omega_{l,e}^r R_e \geq \sum_{k \in D^l} \pi_{l,k}^r T_k \\ \forall r \in \Gamma^l \\ R_e \geq 0 \end{cases} .$$

The space of admissible capacity vectors is then defined by a finite but potentially exponential number of constraints. Γ^l gives the set of possible constraints for a breakdown l . Experiments show, however, that the iterative process between the master program and the satellites converges rapidly (a few dozen minutes CPU on a PC) to the optimal solution even when the network size attains or goes over 100 vertices with 200 or 300 links and two or three thousand traffic demands. The number of constraints generated in PM remains limited and does not exceed a few hundred (much less for medium size networks).

Let us note that many of the constraints of PM have equal coefficients $\omega_{l,e}^r = \omega$ and can then be written as

$$\sum_{e \in E^l} R_e \geq \frac{\sum_{k \in D^l} \pi_{l,k}^r T_k}{\omega}.$$

We immediately notice that these constraints are nothing other than the bipartitioning cuts of the network. They simply express that the sum of the capacities that one must install on the set of links crossed by a bipartition cut of the network must be greater than the sum of the traffic demands whose origin and destination are in each bipartition. Some authors, however, use only these bipartition cuts to solve the multiflow problems and thus avoid the solution of admissible multiflow linear programs. In Γ^l there are other families of more complex constraints and are, in essence, more difficult to define *a priori*. That is why the approach that consists of generating cuts of Γ^l , without recourse to the implicit solution of the linear program of admissibility of the flow previously described, does not necessarily seem more efficient. We can refer to the literature if we wish to have more information on the different families of possible cuts of Γ^l .

7.4.3. Extension of the decomposition model for modular capacities

In the preceding section, the model of dimensioning considered continuous capacity variables. It is easy to extend this model in order to take into account modular capacities. Let us note that it is also possible to handle cases where several modularities cohabit. To facilitate the explanation we will only consider the case with a single module of capacity C . Extension of the continuous capacity variables model is done by replacing the continuous variables of capacity R_e with the integer variables Y_e giving the number of capacity modules C on a link e . For the satellite programs, that entails no change. It suffices to replace R_e by $C \cdot Y_e$ in the right-hand side of the capacity constraints. The master program undergoes more evolutions. It becomes:

$$(PMI) \left\{ \begin{array}{l} \text{Min} \sum_{e \in E^0} p_e Y_e \\ \forall l \in L \quad \sum_{e \in E^l} \omega_{l,e}^r Y_e \geq \frac{\sum_{k \in D^l} \pi_{l,k}^r T_k}{C} \\ Y_e \text{ positive integers} \end{array} \right.$$

and p_e is here the price of a capacity module C for link e . The master program PMI has become an integer linear program that can be solved with a classical solver of linear programs with mixed variables, such as CPLEX.

We can considerably improve the efficiency of this solution by reinforcing the knapsack type constraints on the variables Y . The dual variables $\omega_{l,e}^r$ that we obtain during the solution of the satellites are very often equal. We can then report this coefficient to the denominator of the right-hand side of the constraint. The terms of the left-hand side of this constraint then do not take more than the values 0 or 1. We can then without loss of information reinforce the right-hand side of the constraint up to its upper integer part (we denote with $\lceil \cdot \rceil$ this reinforcement). In fact this reinforcement operation can be done even if dual variables $\omega_{l,e}^r$ are not all equal. Thanks to properties of the constraint matrix of the dual of the multiflow satellite programs (all its coefficients are integers), it is possible to express in a constant term the set of coefficients $\omega_{l,e}^r$ for a constraint r given as the product of a constant term v by the integer coefficients $\alpha_{l,e}^r$ ($\omega_{l,e}^r = v\alpha_{l,e}^r$). The constraints of PMI become:

$$\sum_{e \in E^l} \alpha_{l,e}^r Y_{l,e} \geq \left\lceil \frac{\sum_{k \in D^l} \pi_{l,k}^r T_k}{vC} \right\rceil.$$

Without this reinforcement, solving PMI would doubtless be illusory! It is still possible to improve the efficiency of Benders' iterative procedure by noticing that the succession of optimal solutions of PMI is increasing; by adding an admissibility constraint to PMI , the value of its objective function can only increase. The value of the optimal solution obtained for an iteration can be used as the minimum bound for the solution of the following iteration. Then the solution of this can be truncated when the best admissible solution found reaches this bound.

While the solution of the master program now calls for the solution of an integer linear program reputed to be much more difficult, we can notice with curiosity that the search for the optimal dimensioning of modular spare capacities does not necessarily have a longer calculation time: it is sometimes even the contrary! Of course the computation time to optimize the master program PMI is much longer. On the other hand, the computation time to optimize the satellites is shorter. As the capacities on the links are modular, the decision to add capacity in the network brings much more freedom for the search for an admissible multiflow for each breakdown. The number of optimized satellite programs is thus decreased as well as

the calculation time for the solution of each of these satellites: many fewer paths are computed.

7.5. Conception of cellular networks

Cellular radiotelephony is a communication system that enables clients to telephone while freely moving in a territory. Communications are established with mobile terminals or mobile stations by radioelectric waves. The territory covered by the radio network is cut into cells (see Figure 7.4), each cell being the result of radio coverage of an emitting station or a base station that is a component of the network. This cutting gives the network the faculty to reply to a very large number of communications in spite of spectral resources limited by reuse of the frequencies from one cell to another (for example 62 frequencies for the GSM (Global System for Mobile Communications) in France).

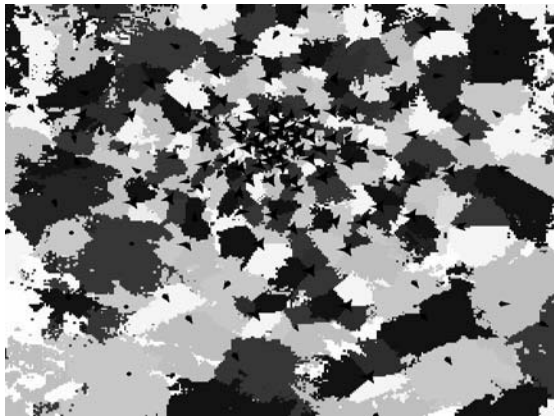


Figure 7.4. *Cellular network*

The quality of the service offered on the entire covered zone is complex to calculate; it depends on several factors tied to individual properties of each cell (station location, coverage, capacity, etc) and to the collective properties emerging from the whole system (mobility, radio quality, etc). Knowing that the real urban networks today exceed 1,000 base stations, the complexity of the problems and the very large economic stakes (the number of mobile telephone subscribers is greater than the number of fixed telephone subscribers) have generated numerous studies on the optimization of cellular networks: localization of sites, connection to the fixed network, radio resource assignment, etc.

The general issue of the conception of cellular networks globally consists of optimizing the choice of the number, location and parameters of the base stations to attain the objectives of service quality fixed by the operator at each point of the covered zone. To fulfill these objectives, optimization and simulation are indispensable tools because the combinatory is very high, the volume of data is large and evaluation functions are complex. A base station can be installed on any roof or building facade and no less than a dozen parameters determine its adjustment: height, type of antenna, orientation, power, frequencies, etc. Also, the evaluation of the service quality can be done from any point in the zone that can receive a mobile phone signal with precision from a few dozen meters to several dozen kilometers. This evaluation is made complex because the presence of a service of quality comes from several network properties, such as its coverage, its capacity to pass traffic or its tolerance to mobility. This last factor, peculiar to mobile radio networks, is certainly the most difficult to manage.

As shown in the preceding section, in fixed network planning, networks are often modeled by graphs whose vertices are transmission points or commutation equipment, and the edges represent means of transport (cables, fibers, etc.) or the flow of data. For these problems the graph model constitutes a natural representation, so that optimization tools may be based on algorithms from operational research. On the radio access network, communications are established via radio waves between a base station that supports a fixed transmitter connected to the network and a mobile station used by the client and situated somewhere in the geographic zone and capable of moving while communicating. Radio network planning thus uses a meshed or discretized zone on which each point of the zone is a potential emission point (in the direction mobile→base) or reception point (in the direction base→mobile) for a mobile station. Optimization of the network must be done by taking into account the service quality on each of the points in this zone. The use of graphs and associated methods to handle the problems is possible, but particular attention must be given to modeling and solution methods that are able to support large size problems. The number of variables and constraints to consider is very large; a radio network covering a zone represented by more than a million points are frequently optimized. Because of this, questions concerning the adaptation of tools guaranteeing optimal solutions to these optimization problems remain open. For the moment, in the absence of adapted theoretical tools, an important place is given to approximate approaches based on metaheuristics. These let us propose quality solutions in acceptable times and use heuristics tested by engineers to solve small size problems.

The following sections present research work that lead to the development of evolutionary algorithms and tabu algorithms to consider planning problems in radio networks. Two main subjects are addressed, the search for positioning and configuring radio sites, which implements the construction of cells, and the

assignment of frequencies at the base stations of the network that furnishes a frequency plan capable of delivering the traffic demand. The two subjects addressed are related to the GSM system that is the most widely used in the world today [LAG 99]. In conclusion, we will mention a few elements of the evolution of problems in view of the deployment of the UMTS (Universal Mobile Telecommunications System), a relatively new norm of digital communication, which started in 2004. Simultaneous optimization of cellular topology of the network, induced by the positioning and configuration of the antennas, and quality radio management after the assignment of frequencies, is the ideal method to design a network. But the two problems are in themselves very difficult, and they are almost always considered separately. Some tentative trials regrouping the two problems to envisage them as a single unique optimization problem have been presented in literature, but in making many concessions to antenna positioning and configuration, with a very restrained number of parameters or quality metrics, the results are made impracticable [CAL 96, FLO 97, FRU 97]. The two problems are considered separately here, with frequency assignment being seen as a result of antenna positioning and configuration. In addition to these two problems, the definition of mobile localization zones and connection of base stations to the fixed network constitute two other problems that complete the range of difficult problems in planning mobile radio networks. These two problems are not addressed here.

7.6. Antenna positioning and configuration

Antenna positioning and configuration is a difficult problem because it cumulates several components whose integration is antagonistic. From the angle of optimization, we must simultaneously treat several objectives that lead to solutions that cannot coexist; for example, maximizing the cover for a number of given sites while minimizing the overlap among the cells. However, the real cells are never perfectly contiguous, contrary to theoretical studies that fix the number of sites. From the angle of exploitation, we must approach a good solution in a reasonable calculation time, for example in less than 24 hours, whereas most of the models of field-strength prediction that furnish the input data to determine the quality of assigned parameters do not allow us to test more than a dozen of coverage solutions in 24 hours. This problem thus necessitates great reflection in terms of mathematical modeling and algorithmic efficiency. This is the reason why the literature offers only very few references on this subject, except in indoor network design (antennas located inside buildings) where the territories to cover are very restricted, which makes acceptable the necessary computational time for combinatorial optimization [SHE 94, FOR 95]. In practice, the design of cellular topology is still done empirically by experts assisted by simulation tools to evaluate the quality of their solutions.

In general, the problem of antenna positioning and configuration needs several resources whose allocation is sequentially done by the network operators: positioning of the base stations in a set of candidate sites, station configuration notably in number and type of antennas, and antenna configuration settings. Allocation of the set of resources organizes the cellular topology. A cell is a geographic zone whose localization, coverage and supported quality of service depend on the configuration of a quadruplet (site, station, antenna, frequency). Even if the set of cells forms the cellular network, their juxtaposition is not sufficient to construct a satisfactory network. The network is a complex system whose design necessitates a systematic approach. The control of interactions among the cells that compose it is a key element of the service quality given to clients. By supposing that the set of cells covers the territory on which the network operator wishes to offer a wireless communication service, there remains the issue that mobility management is difficult to tackle. To succeed, each cell must be constructed and configured according to the others, its neighbors. For this reason, all the resources must be allocated simultaneously to allow for the need for coverage to satisfy the fundamentals of the mobile radio system, that is, the service quality, the capacity and, above all, the mobility.

The objectives of service, capacity or mobility change considerably according to the considered territory: indoor and outdoor deployments, and rural and urban environments do not have the same needs. However, it is possible to establish general optimization objectives. In the same way, the constraints to respect can change from one environment to the other, but generally requirements on cover and traffic must be satisfied. On the other hand, available necessary information to design the network is very different according to the environment: building maps in 3D for an indoor network (commercial center, company, etc), contour of building and open spaces for an urban network, nature of the above-ground terrain and precise layout of rural roads and highways, etc. This variety of data describing the environment is justified by the singularity of the phenomenon of radioelectric wave propagation according to the surroundings. The waves react differently according to the obstacles that they meet. For this reason, several radio wave propagation models, which combine differently the phenomena of reflection, diffraction, penetration, etc, are used to simulate cellular coverage, and each of them needs particular information [BOI 83]. It follows that it cannot be directly envisaged to propose a global mathematical model furnishing all the elements necessary for combinatorial optimization of a cellular network for all environments. The model explained below, and used later, is related to the design of the cellular network in external surroundings with macro-cell type (antennas installed above the roofs).

7.6.1. Multicriteria model

The model that we propose comes from the ESPRIT IV ARN research project (IT 23243 financed in part by the European Commission) for *Algorithms for Radio Network Optimization*. It concerns, in particular, the GSM system, but its principle can be extended to optimization of networks using other cellular systems. In addition to the modeling of a large number of data and constraints, among which are those that come in the design of a cellular network, this model presents some originalities in the definition of objectives of network conception. It deals with a concentrator link model that aims to simultaneously maximize objectives of cost, capacity and radio quality. It is thus a multicriteria model. The objectives of capacity and radio quality are the basic elements needed in order to offer a guarantee of service quality to network clients. The cost objective is the indispensable complement that assures the operator the design of the best network, at the best price, for the expected service quality.

The concentrator link approach seems perfectly adapted to the situation since to establish a communication network we indeed imagine that a mobile, wherever it may be, must be attached to a base station by radio. The organization model must then allow us to be assured that at every point of the territory it is possible to establish a radio connection with at least one base station in the network, and thus guarantee the cover of the territory. This approach has been adopted in several studies [SHE 94, FRU 96, FLO 97]. The sections that follow successively introduce the description of data, constraints and the objectives of the problem on the basis of a multicriteria concentration model. A more detailed description of the presented model is found in [REI 98]. Although incomplete, this model is a good base to understand real problems.

Data of the problem

The data of the problem come essentially from the dimensioning phase before the phase of network design. This phase also depends on the resources and objectives of the design in terms of cost, capacity and expected radio quality. Let \mathcal{P} be the deployment zone of the regularly discretized network in which all the points are marked by their coordinates (x, y) ; the data from the dimensioning are:

- \mathcal{RT} and \mathcal{ST} , sets of predefined points RTP_i and STP_i of \mathcal{P} , so that $\mathcal{ST} \subseteq \mathcal{RT}$, on which the quality of the radio signal must be tested and eventually compared to a threshold Sq_i for a given service. The threshold of quality essentially depends on the type of mobile station expected on each point.

- T , a matrix describing a demand in traffic e_i at each point of \mathcal{P} (often approximated by Erlang's distribution) [LAG 99]).
- L , a set of points L_i , or sites, predefined from Π on which base stations could be deployed. The sites are described by their height z . The sites classify rather diverse locations: high buildings, negotiated buildings and operator buildings, etc.
- \mathcal{F} , a set of costs f_i per site L_i . f_i is a function that estimates the cost of interventions carried out on L_i . The interventions considered are the opening of a site, the addition and the modification of a base station on the site (for example, an antenna change) and the closing of a site (base stations are put out of service). The intervention cost varies according to the site access, the rent, etc.

Engineering parameters

Engineering parameters are the set of given rules relative to the design of the cellular networks. Some are directly furnished by engineering rules on network equipment, while others are more complex and come from digital models:

- Parameters on the base stations (Table 7.1): localization, height, antenna, horizontal (azimuth) and vertical (tilt) orientations of the antenna, power at emission, amount of transmission/reception equipment (TRX). These parameters are the unknowns of the problem. The table below gives their domain of validity for a base station k of site j , noted as $B_{j,k}$. The set of base stations is noted as \mathcal{B} .
- Parameters on the antennas: vertical diagram DGV , horizontal diagram DGH , gains, etc. The diagrams of the antenna describe the loss of signal power on horizontal and vertical planes. These parameters are fixed for each antenna.
- Model of radio-wave propagation: it is a digital model simulating the loss of the radio wave on the path between any two points representing one transmitter (base station) and one receiver (mobile station). This model is rather complex; it takes at entry a cartography as precise as possible, including a digital model of ground and above-ground terrain. Its output is $AFF_{i,j,k}$, the loss of the signal in dB between a base station $B_{j,k}$ and any point RTP_i of \mathcal{RT} .

Configuration of base station $B_{j,k}$	Domains
Antenna	Database
Azimuth	0° to 360°
EIRP (emitted power)	33 dBm to 55 dBm
Height	0 to 150 meters
TRX	1 to 7
Tilt	0° to 15°
Localization	L

Table 7.1. *Parameters on base stations*

Constraints

The design of the network is strongly conditioned by engineering constraints. They guarantee the deployment of a cellular network satisfying a minimum service quality. Within the optimization problem they define the set of feasible solutions. The three major constraints of the model are defined below:

- cover constraint: each point ST_i in the deployment zone \mathcal{ST} must be covered with a quality signal verifying the threshold of quality Sq_i on this point;
- handover constraint: the cell of each station $B_{j,k}$ must have at least one cover zone with that of another station $B_{u,v}$ in order to guarantee the continuity of a mobile communication;
- connectivity constraint: the set of points from \mathcal{ST} covered by a station $B_{j,k}$ must be connected.

Objectives

At last the objectives of the problem are fixed. Network optimization is realized by taking into account numerous quality metrics. The importance and utility of these metrics depend notably on the considered step (newly deployed or evolving network) and on the investment strategy of the operator. The four objectives below constitute a departure point to be adapted according to the case. They are interdependent and antagonistic, and their simultaneous management is thus difficult. The problem of the design of cellular networks is a multicriteria optimization problem with constraints to satisfy:

- minimize the cost of the network; this objective is directly related to investment cost;

$$\min \sum_{j=1}^{|\mathcal{I}|} f_j y_j$$

where $y_j=1$ if L_j locates a base station; if not, $y_j=0$. f_j represents the cost (aggregation of economic factors) of the site L_j ;

– maximize the traffic flow: this objective aims at the best response to traffic demands;

$$\max \sum_{j=1}^{|\mathcal{L}|} \sum_{k=1}^3 \tau_{j,k}$$

where $\tau_{j,k} = \min \left\{ n, \sum_{i=1}^{|\mathcal{ST}|} e_i \cdot x_{i,j,k} \right\}$, with n given, knowing that $x_{i,j,k}=1$ if the point

ST_i is covered by the station $B_{j,k}$, and 0 if not;

– maximize the yield: this objective aims to avoid the overdimensioning of a network by taking into account its capacity;

$$\max \sum_{j=1}^{|\mathcal{L}|} \sum_{k=1}^3 \frac{\tau_{j,k}}{E(B_{j,k}^{TRX})}$$

where E is a given Erlang table, and $B_{j,k}^{TRX}$ is the number of TRX of the station $B_{j,k}$;

– minimize the interference: this objective aims to manage signals that are useless for cover and mobility, and that penalize frequency management;

$$\min \sum_{i=1}^{|\mathcal{ST}|} \Upsilon_i$$

where Υ_i is a set of interfering signals.

The following section introduces a solution method for the problem of positioning and configuring antennas on the base of the above model.

7.6.2. A heuristic for positioning antennas

Complexity of the problem

The complexity of the antenna-positioning problem is immense. The first factor of this complexity is linked to a very large number of possible combinations to configure the base stations of a network. Indeed, with a realistic discretization for the different antenna parameters, we obtain several thousand possible configurations for a single site. Let us call this number A . A will be multiplied by B if we have B candidate sites (typically several hundred). To construct a network is to find a configuration among these $A \times B$ possible choices that verifies the constraints and optimizes the objectives, and which involves a large size search space of $2^{A \times B}$. Thus,

for a realistic network of 400 candidate sites, the size of the space is of the order of $2^{3,000,000}$.

The second factor of this complexity concerns a very large number of non-trivial calculations. Indeed, at each step of the solution process it is necessary to verify the satisfaction of the constraints and the evaluation of the objectives. Thus, for an average of 100 selected stations in a typical network, one has to realize more than a million non-elementary calculations to evaluate a network configuration.

These two characteristics make an exact solution impractical. An approximate solution is thus inevitable. Nevertheless, it is to be noted that a single metaheuristic would not be sufficient to address such a complex problem as antenna positioning and configuration. It would doubtless be more judicious to combine the metaheuristic approach with more specific techniques (heuristic or other). To illustrate this principle we will present a heuristic approach that combines a pre-processing technique, an optimization step with tabu search and a post-optimization step [VAS 00]. Two other heuristic solution approaches following this combination principle are presented in [BER 98, CHA 98].

Reduction of search space by pre-processing

Pre-processing is a well-known technique in optimization that aims to diminish the size of the initial search space and the calculation time. For the problem considered, an interesting pre-processing consists of using certain constraints of the problem to suppress bad configurations. The fundamental idea is the following: for each candidate site, we calculate all the possible cells generated by a complete configuration of a base station (power, tilt, azimuth, etc). According to the individual characteristics of these cells, we retain or refuse the stations that engender them.

Out of about 3,000,000 possible configurations for a typical network, pre-processing retains on average only about 200,000 to 300,000 configurations that are likely to participate in a good solution. If today it is practically impossible to realize an efficient search in a space with $2^{3,000,000}$ points, we can hope to find a good solution in a greatly reduced space of $2^{300,000}$.

This pre-processing also favorably prepares the following optimization step by the realisation of a large number of costly calculations. Indeed, the value change of an antenna parameter has effects on numerous cells. To evaluate these effects in the network, we must realize a large number of non-trivial calculations (arc-tangent, real divisions, separation of the signals, calculation of connected components, etc). These calculations being realized and stored by pre-processing, the values of these calculations will be directly available during the optimization step by tabu search.

After this pre-processing step, we have a reduced set of cells that individually verify certain desired properties. These cells constitute the point of departure for the network construction. This means finding a subset of stations among the stations chosen by pre-processing so as to cover the geographical zone while respecting the constraints and optimizing the objectives of the problem. This task is assured by a tabu algorithm.

Optimization by tabu search

Tabu search (see Chapter 3) is a very powerful metaheuristic founded on the notion of *neighborhood* [GLO 97]. As with every neighborhood method, it follows a rather simple general scheme. The search begins with an initial configuration and then realizes an iterative process that, at each iteration, chooses a neighbor of the current configuration according to a neighborhood N and then replaces the current configuration by the neighbor selected. This process stops and returns the best configuration found when the stopping condition is realized. This stopping condition generally concerns a limit for the number of iterations or an objective to be realized.

From this general scheme, we note certain similarities between tabu search and simulated annealing. However, unlike simulated annealing that randomly considers a *single* neighbor solution $s' \in N(s)$ at each iteration, tabu search examines a *set* of solutions, often the totality of solutions of $N(s)$, and retains the best s' even if s' is worse than s . A tabu search thus does not stop at the first local optimum met. However, this strategy can lead to cycles, for example a cycle of length 2: $s \rightarrow s' \rightarrow s \rightarrow s'$. To prevent this type of cycle, we store the last k configurations visited in a short-term memory and forbid every movement that leads to one of these configurations. This memory is called the *tabu list*; it is one of the essential components of this method. The tabu list allows us to avoid all the cycles of length inferior or equal to k . The value of k , called the length of the tabu list, depends on the problem to be solved, and must be determined empirically. This length can possibly evolve in the course of the search. That is often realized with the help of a simple function defined on a significant parameter of the problem, for example the number of non-satisfied constraints. Let us note that a universal optimal length does not exist.

The storage of entire configurations in a tabu list would be too costly in calculation time and in memory space and doubtless would not be a very efficient option. In practice, the tabu list stores only *characteristics* (for example, pair variable-value) of the configurations in place of complete configurations. More precisely, when a movement is made from the current configuration to one neighboring configuration, it is generally the characteristic lost by the current configuration that becomes tabu. Thus, if the movement has modified the value of a variable, we forbid the return to the former value of this variable for the duration of

the tabu list. When a tabu list records characteristics, the interdictions that they engender can be very strong and restrain the set of admitted solutions at each iteration in a too brutal manner. A particular mechanism, called *aspiration*, is put in place in order to overcome this disadvantage. This mechanism allows us to revoke the tabu status of a configuration without, however, introducing a risk of cycles in the search process. The aspiration function can be defined in several ways. The simplest function consists of revoking the tabu status of a movement if the tabu status allows us to attain a quality solution superior to the best solution found up until then.

There exist other interesting techniques to improve the power of this basic search schema of tabu search, in particular by *intensification* and *diversification*. Both are based on the use of long-term memory and are different according to the means of exploiting the data in this memory. Intensification consists of realizing a detailed search in a zone judged as interesting. Diversification has an inverse objective from intensification: it directs the search toward unexplored zones. Intensification and diversification thus play a complementary role.

Thus, to develop a tabu algorithm for a given problem, it is necessary to develop a set of base elements as the configuration, the evaluation function, the neighborhood, the tabu list, etc. We will now present the principal ingredients of a tabu algorithm to treat the application of antenna positioning and configuration.

Configuration and evaluation

A solution to the problem will be composed of a subset of the stations furnished by the pre-processing stage. A configuration will then be naturally defined by a binary vector, each component representing the presence or not of a station. More formally, let β be the set of stations selected by pre-processing; a configuration is a vector $s = (b_1, \dots, b_{|\beta|})$ of $X \subset S = \{0, 1\}^{|\beta|}$ where X is the partially constrained space defined by the set of binary vectors verifying certain constraints of the initial problem. Such a configuration only verifies a part of the initial constraints. The non-satisfied constraints (relaxed) are integrated in the evaluation function and treated by a penalty approach.

To evaluate the quality of such a configuration in relation to the set of objectives and relaxed constraints, we adopt a prioritized approach. More precisely, we use a vector function:

$$\xi(s) = \langle c_1(s) \dots c_p(s), f_1(s) \dots f_q(s) \rangle$$

where each c_i ($1 \leq i \leq p$) is a function of evaluation of a relaxed constraint and each f_j ($1 \leq j \leq q$) an evaluation function of an objective of the problem. We accord a greater importance to a constraint than an objective to favor satisfaction of relaxed

constraints. By imposing an order of priority on these functions we can compare any two configurations.

Neighborhood and tabu list

A possible neighborhood relation for a binary problem is founded on the *add/drop* principle: from a configuration we change the value of a variable from 0 to 1 and another from 1 to 0. In this case, where the configurations are subject to constraints and held in a restrained search space X , this simple neighborhood *add/drop* can be extended to *add/repair*. Here, when a variable is set at 1, we modify as many variables (transition $0 \rightarrow 1$) as necessary to bring the configuration back into the space X .

For antenna positioning we can define the neighborhood N as follows. Let $s = (b_1, b_2, b_3, \dots, b_{|\beta|}) \in X$ be a valid configuration, a neighboring configuration $s' = (b'_1, b'_2, \dots, b'_{|\beta|})$ is obtained by a series of flips: $(b_i: 0 \rightarrow 1, b_{i_1} \dots b_{i_m}: 1 \rightarrow 0)$ where $b_{i_1} \dots b_{i_m}$ are variables connected to b_i by a constraint. This neighborhood consists of adding to the current network a station β , and of dropping a certain number of stations so that the new configuration remains in the partially constrained space X . Such a neighborhood implies for each configuration s of X , $|N(s)| = |\beta| - |s|$ neighbors with $|s| = \sum_{1 \leq i \leq |\beta|} b_i$.

The tabu search algorithm navigates in search space X to visit configurations there. To prevent the search from being trapped by the cycles, the tabu list stores recently realized movements. Its use necessitates a vector T of length $|\beta|$ of which each component corresponds to a possible movement of the current configuration. Thus, each time that a variable b_i is set at 1 (addition of a station in the network), setting this variable back to 0 (suppression of the station) is forbidden for the next $T[i]$ iterations. The value $T[i]$, the length of the tabu list, is determined by a function $freq(i)$ that corresponds to the number of times where b_i is changed from 1 to 0. In practice, $T[i]$ stores $freq(i) + j$ where j is the current iteration. Thus, to know if a movement is classified tabu at an ulterior iteration t , it suffices to simply compare t and $T[i]$.

Strategy of movement and aspiration

At each iteration, the tabu algorithm examines the set of neighbor configurations $N(s)$ that are not forbidden by the tabu list and then selects the best neighbor s' . The selected neighboring configuration s' then replaces the configuration s and becomes the current configuration. It is important to note that the neighbor retained is not necessarily of better quality than the configuration that has just been replaced. Indeed, it is this characteristic that gives the tabu search the possibility not to stop at

a local optimum. It should equally be noted that the tabu search recommends the evaluation of all the neighbors at each iteration of the search. That could be costly *a priori*. In practice, two solutions are possible. The first uses a rapid incremental evaluation with the help of a data structure Δ that stores the cost differential for each possible movement. In the case where such a data structure is not available, we then adopt the second solution that consists of examining only a subset of $N(s)$. Very often, this subset is either randomly taken from $N(s)$, or limited to the configurations verifying certain properties.

Post-optimization

Post-optimization is a technique widely used in combinatorial optimization and constitutes the last stage of our global heuristic approach. Starting from a solution furnished by the tabu algorithm, this step tries to improve the quality of this solution even more by focusing on a particular objective (diminution of the potential noise level, traffic lost, etc). It can also serve to satisfy constraints (cover, etc) that remained violated during the optimization by the tabu search. Post-optimization is essentially assured by some simple deterministic rules acting on the antenna parameters. In general, modifications in this step are rather limited.

This heuristic approach has been tested in two types of situations. The first consists of adjusting certain network parameters to optimize network quality. The second consists of constructing an entire network. In both cases, different networks are envisaged, corresponding to typical environments: route axis, rural environment, urban environment or even a dense urban environment. Experimental results show that this heuristic approach is easily adaptable to supply optimized solutions to construct a network and to optimize an existing network.

Similar results have been obtained by other heuristic approaches [BER 98, CHA 98]. Like the approach that has just been presented, these approaches combine a general heuristic such as simulated annealing, evolutionary algorithms or neural networks with problem-specific heuristics. All these studies clearly show us the interest of combinatorial optimization by heuristics to position and configure antennas.

7.7. Frequency assignment

Frequency assignment varies according to the application (FM diffusion, TV diffusion, mobile phones, etc) and to the radio communication system used. In the case of mobile telephony by terrestrial cellular networks, there are several systems that have their own standards of frequency management and, in particular, of interference and transmission rates. Spectral resources, usage constraints and distinct quality and capacity objectives illustrate these particularities. However, in all these

cases, the success of the cellular network makes it necessary to optimize spectrum use. The need for operators to satisfy their clients in a context of strong competition emphasizes the importance of constructing good frequency plans. In addition, the very large size of these networks justifies the use of optimization algorithms to manage the complexity of the problem.

Concerning the GSM system, the problem of frequency assignment consists of allocating a finite number of frequencies to each emitting station (BS) in the network. These frequencies will be used by the station to send the signal for each individual connection established with clients' mobile phones. The number of frequencies to allocate is defined according to the functions of the GSM system present in the network. With activation of the frequency hopping, the number of frequencies to assign is also a parameter of the problem. In the simplest case, without activation of the frequency hopping, d_i the number of frequencies to assign per base station s_i is firstly fixed according to the traffic demand in the zone covered by the base station, the cell. The following model is dedicated to this type of case. There is currently no widely used public model that integrates the frequency hopping in a satisfactory manner and can be used as a reference.

Generally, no matter what system options are activated, the frequency assignment is a constrained optimization problem whose objective is to minimize the interference to satisfy a level of jamming or quality of service. Radio interference is produced in an area when several emitting stations cover this area, and when these stations use the same frequency channels (i.e. co-channel interference) or frequency channels that are close on the spectrum (i.e. interference in adjacent channels). The interference with frequency $f_{i,l}$ of station s_i in any area (point) m is evaluated by the following function:

$$\sigma_{i,l}^m = \frac{C_i^m}{\sum_{s_j \in BS - \{s_i\}} \sum_{k=1}^{d_j} \frac{C_j^m}{K_{abs(f_{i,l} - f_{j,k})}}}$$

where m is an element of \mathcal{P} the set of points, C_i^m and C_j^m are the respective fields of the base stations s_i and s_j in m , $K_{abs(f_{i,l} - f_{j,k})}$ is the level of protection among the frequencies $f_{i,l}$ of s_i and $f_{j,k}$ of s_j and BS is the set of network base stations.

The frequency $f_{i,l}$ of the station s_i is jammed at a point m if: $\sigma_{i,l}^m \leq t_{jam}$, where the jamming threshold t_{jam} is dependent on system functions (activated or not) on the network. Minimization of interference can be treated in diverse ways: minimize the

number of jammed points, minimize the volume of jamming relative to the interference threshold considered for all the jammed points, minimize the absolute interference independently of a threshold, etc. [KOS 99] gives a rather large list of the problems that can be formulated. The most classical consists of minimizing the volume of interference relative to a threshold for a number of jammed points determined in advance. In this case, by making an abstraction of multiple jamming, i.e. by limiting oneself to the interference of a station s_i by a unique station s_j , the problem can be modeled in the form of a problem of constraint satisfaction. An engineering rule lets us determine the separation of frequencies to respect between the channels of s_i and the channels of s_j to satisfy the interference relative to the given threshold; the problem then consists of assigning the frequencies to the stations so that the constraints of frequency separation between station pairs are satisfied. It is this model that has been retained for this study. This model has been much used during the last ten years and has led to a very large number of studies published in the domain of radio networks and operations research including [BOX 77, GAM 86, DUQ 93, TIO 95, DOR 95, 98b, CAS 96, HAO 96, 98] which are basic references.

In all the models, the problem is made complex by a large number of basic constraints to be satisfied. The constraints are traffic demand, initial assignments, forbidden assignments, intra-station separation of frequencies and separation of frequencies between stations on the same site. In GSM, the number of available frequencies varies according to world regions. In no case can an operator use frequencies outside the radio spectrum attributed to him by license; for example, French operators have 62 frequencies in the 900 MHz band. However, the demand for frequencies goes way over this resource. A medium size urban network frequently has from 3,000 to 4,000 channels; a large size network has more than 10,000 channels, which is the case for London, Paris and Rome. This restriction thus imposes a very great rate of spectrum reuse that must be satisfied; it is unacceptable for an operator not be able to totally satisfy the demand – it is a question of survival. However, reuse is a source of radio interference, which makes interference minimization difficult. The other constraints are directly concerned with the conditions of frequency reuse. The initial or forbidden assignments suppose that certain frequencies are already assigned to certain channels or that certain frequencies cannot be assigned to certain channels. It is the same for one station or among a number of base stations on the same site. The channels cannot use the same frequencies or neighboring frequencies in the spectrum; a minimum separation of 3 frequencies is required for intra-station or (co-station) channels, and 2 frequencies for the channels among co-site stations.

7.7.1. Modeling by set T-coloring

Set T-coloring

The problem of frequency assignment can be appropriately formalized with a mathematical model: set T-coloring [HAL 80]. Set T-coloring is a generalization of the classical graph coloring model, which is widely studied in the literature.

Let $G=(V;E)$ be a non-oriented graph where $V=\{v_1...v_n\}$ is a set of vertices and E a set of edges $E=\{\{v_i, v_j\} \mid v_i, v_j \in V\}$. A k -coloring (k positive integer) is a function $c : V \rightarrow \{1...k\}$ so that for each edge $\{v_i, v_j\} \in E$, $|c(v_i) - c(v_j)| \neq 0$.

In a k -coloring of G , each vertex of G is colored with one of the k colors $\{1...k\}$ so that the colors of the two vertices are *different*. The *problem of graph coloring* consists of determining the chromatic number $\chi(G)$ of a given graph G , i.e. the smallest k for which there is a k -coloring.

We can generalize this coloring problem to introduce the problem of set T-coloring (let us note ST-coloring) in the following manner. We associate with each vertex in the graph a specific number of colors. For each pair of colors (from one or two adjacent vertices), we specify a separation constraint defined by a set of forbidden values. To determine a ST-coloring with k given colors consists of coloring the vertices in the graph so that each vertex receives the number of colors demanded and the separation constraints are respected.

Modeling frequency assignment

We can now formalize the problem of frequency assignment with ST-coloring [HAL 80, DOR 98a, 98b]. To do this, we represent the available network frequencies by colors, and we construct a ST-coloring graph $G=(V;E;T;D)$ where:

- V is the set of mobile radio stations;
- E is the set of edges representing neighboring stations;
- D is the set of frequency needs per station (number of colors per vertex)
- T is the set of minimum frequency separations for intra-station channels and for channels between neighboring stations.

The following example shows the model of a network of four stations with the number of frequencies per station and the necessary frequency separations. For example, station S3 demands two frequencies that have a minimum separation of 3 in the radio spectrum. In the same way, a minimum distance of 2 must be respected between frequencies S1 and S3 and frequencies S1 and S4.

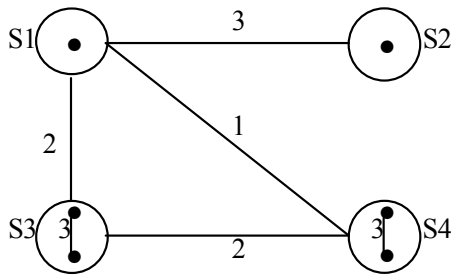


Figure 7.5. Frequency assignment and T -coloring with sets

Thus the search for a frequency plan without interference with k frequencies consists of determining a ST-coloring with k colors.

7.7.2. Solution by evolutionary algorithms

Evolutionary algorithms, more particularly genetic algorithms (see Chapter 3), are heuristic optimization methods whose principle is inspired by the process of natural evolution [GOL 89, MIC 92]. A typical evolutionary algorithm is composed of three essential elements:

- a *population* constituted of several individuals representing potential solutions (configurations) of the given problem;
- an *evaluation mechanism* of the adaptation of each individual in the population in regard to its exterior environment;
- an *evolutionary mechanism* composed of operators making it possible to select certain individuals and to produce new individuals from selected individuals.

From an operational point of view, a typical evolutionary algorithm begins with an initial population and then repeats an evolutionary cycle composed of three sequential steps:

- measure the quality of each individual in the population by the evaluation mechanism;
- select a part of the individuals;
- produce new individuals by recombination of selected individuals.

This process ends when the stopping condition is verified, for example when a maximum number of cycles (generations) or a maximum number of evaluations is

attained. According to the analogy of natural evolution, the quality of individuals in the population should tend to improve during the process.

The objective of *selection* is to choose individuals able to survive and/or reproduce to transmit their characteristics to the following generation. The selection is generally based on the principle of conservation of the best-adapted individuals and the elimination of those less well-adapted. *Crossover* or recombination tries to combine the characteristics of parent individuals to create offspring individuals with new potentialities in the future generation. *Mutation* causes slight modifications of some individuals.

The theory of classical genetic algorithms advocates random operators (random mutation, random crossovers, etc). However, such an algorithm for optimization is rarely competitive without adaptation. In effect, to be efficient, a genetic algorithm must be carefully adapted to the problem treated by a specialization of its operators and by integration of specific knowledge of the problem. In the same way, genetic algorithms provide a general framework that, combined with other methods of specific or general solution, can lead to extremely powerful hybrid methods.

By adapting the principle of genetic algorithms, an evolutionary approach has been developed for the problem of frequency assignment [HAO 96]. This approach is made up of an iterative cycle of evolutions operating on a population of frequency plans with specific evolution operators. By associating a rate of application with each evolution operator, this approach allows the reproduction of different evolution cycles, for example without crossover or mutation.

This approach lends itself well to an efficient distributed implementation. Thus in [REN 97], the authors present such an implementation on a model of islands. In this distributed algorithm, each individual of the population is distributed on an island and is managed by a processor. The individuals evolve on the island by a mutation mechanism with the local search. Periodically, a *control operator* is triggered and decides on the exchange of information between islands. Contrary to the model of classical islands where the individuals transit from one island to another, here the exchanges are realized by a crossover mechanism between two individuals on the two islands. Thus, this distributed algorithm furnishes at the same time a great diversification of the population and an efficient regeneration of individuals offering new potentialities. It should be emphasized that the control operator plays an important role in such an algorithm. Experimentation shows a better regeneration of the population when the exchange operations are realized regularly, after fixed periods. The exchange period must then be determined according to the problems, and the islands must have sufficient time to intensify the search and discover the local minima.

These evolutionary algorithms are founded on a set of basic elements, such as configuration, the function and evaluation mechanism of individuals, the operators of evolution (mutation or hybridization) [DOR 95, 98b, HAO 96, REN97], which we present below.

Configuration and search space

Let a network with n stations be $\{s_1 \dots s_n\}$, with a demand d_i for station s_i and k available frequencies numbered from 1 to k . A configuration $s = \langle f_{1,1} \dots f_{1,d_1}, f_{2,1} \dots f_{2,d_2} \dots f_{n,1} \dots f_{n,d_n} \rangle$ is a complete assignment plan of k frequencies in the set of stations verifying the demand constraint. The search space S of a problem is made up of the set of configurations s of the problem: $S = \{s \mid s \text{ is a configuration}\}$.

Evaluation function

Let $s \in S$ be a configuration, the evaluation function $\text{eval}(s)$ gives the weighted sum of the set of non-satisfied interference constraints by s : the smaller the value, the better the quality of the configuration. The value zero corresponds to an assignment plan without interference.

Mutation by local search [DOR 95]

A mutation by local search corresponds to a neighborhood method adapted to the frequency assignment problem. We will first introduce the neighborhood below.

Let $s = \langle f_{1,1} \dots f_{1,d_1}, f_{2,1} \dots f_{2,d_2} \dots f_{n,1} \dots f_{n,d_n} \rangle$, then $s' = \langle f'_{1,1} \dots f'_{1,d_1}, f'_{2,1} \dots f'_{2,d_2} \dots f'_{n,1} \dots f'_{n,d_n} \rangle$ is a neighboring configuration of s if and only if there exists a unique couple $(i,m) \in \{1..n\} \times \{1..d_i\}$ so that $f_{i,m} \neq f'_{i,m}$. In other words, a neighbor of a configuration is obtained by the unique modification of a frequency of a station in this configuration. A movement is then characterized by a pair (station, frequency). This neighborhood function implies a great number of neighbors for large networks (where the number of stations is greater than 100). We can reduce the size of this neighborhood by focusing on the conflicting frequencies. In effect, to improve a frequency plan, it would be judicious to modify the frequency of a station *in conflict* (a station is in conflict if one of its current frequencies invalidates an interference constraint). That gives us a reduced neighborhood.

Once the neighborhoods are defined, we can introduce multiple mutation operators by local search: descent, descent plus *random-walk*, simulated annealing or even tabu search. Thus, a mutation by descent with the reduced neighborhood replaces the frequency of a station in conflict by another frequency so that the number of non-satisfied interference constraints in the new configuration is diminished. In the same way, the mutation *descent + random walk* realizes the

descent with a rate p , and a random movement with a rate $1-p$. Finally, the mutation can be assured with the strategy of simulated annealing or tabu search.

Geographic crossover [REN 97]

In general, specific crossovers use knowledge of the problem to transmit the good genes of the parents to the offspring. Geographic crossover uses spatial properties of the problem of frequency assignment. The network base stations, and thus the supporting channels, are localized and fixed in space. It follows that for each frequency allocated to a given station, the radio interferences generated or supported by it are subjected to or are the fact of other frequencies that are found in a region delimited in space. Geographic crossover exploits this property of problem localization (see Algorithm 7.2).

To set up this crossover, a set of cells is identified from a reference cell called C_i . The cells of the set have particular interference properties with C_i : they can interfere or be jammed by the frequencies used by C_i . The set thus constitutes a point of local interference centered on C_i that can be more or less well solved for each individual of the population. From a cell C_i randomly taken in the network, the function of geographic crossover is to identify this set and to create from two parents two new solutions resulting from the exchange of their information concerning the solution of this local point. Algorithm 7.2 defines this crossover.

```

Select a jammed cell  $C_i$ 
Construct the reference set  $SetC_i$  for  $C_i$ 
Let  $p_1$  and  $p_2$  be the parent individuals, and  $e_1$  and  $e_2$  the offspring individuals
Do for  $j=1$  to  $n$ , the number of network stations:
    If  $C_j$  is in  $SetC_i$  then
         $e_1$  inherits the frequencies of  $C_j$  of  $p_2$ 
         $e_2$  inherits the frequencies of  $C_j$  of  $p_1$ 
    Else
         $e_1$  inherits the frequencies of  $C_j$  of  $p_1$ 
         $e_2$  inherits the frequencies of  $C_j$  of  $p_2$ 
End If

```

Algorithm 7.2. *Crossover algorithm*

Crossover has a rather large impact on the quality of the two solutions produced. If the local exchanged zone is intrinsically coherent in the two individual offspring, the border between this zone and the rest of the network is very disrupted. It gives

the systematic production of interference on the border of the zone between the set of cells $SetC_i$ and the cells of the rest of the network on the periphery of the local zone. If the interference criteria for the reference set are relatively simple to establish in our case, the size of this set is an element that must be parameterized according to the problems to be treated. Indeed a set that is of small size in comparison to the complete network will only have a slight impact on the crossover; the initial solutions risk being rapidly reconstituted by blocking perturbations brought to the cells in the rest of the network. In this case we must consider a recursive procedure on $SetC_i$ and select jammed cells of the second rank in relation to C_i . According to this principle, an appropriate solution consists of extending the recursive selection until a significant proportion of cells in relation to the set of network cells is obtained, thus giving to the imported zone a chance to impose an external redevelopment if it has a good intrinsic quality.

The evolutionary algorithms presented have been applied on very large networks (more than 3,000 stations). They give a quality solution in a few minutes that, although suboptimal, is much better than custom-made engineering methods. In the last ten years the problem of frequency assignment has generated numerous academic studies. Today optimization algorithms founded on metaheuristics are the basic element of decision-making in the services of frequency planning for the majority of GSM mobile radio network operators in the world.

7.8. Conclusion

We have presented in this chapter some important optimization problems in telecommunications networks:

- concerning fixed networks, determination of the topology and the optimal capacities of the principal network, and the dimensioning of reserve capacities,
- concerning mobile radio networks, antenna positioning and configuration to define cellular topology and frequency assignment.

We have seen that even for the academic problems presented here (the models presented do not fully reply to operational needs), the algorithms used can be particularly varied and complex. For difficult problems like determination of network topology, for fixed networks as well as for cellular networks for which the problematic is quite distinct, we have seen that the use of greedy heuristics and metaheuristics, such as simulated annealing, evolutionary algorithms or tabu search, allows us to rapidly obtain efficient solutions. For another class of problems, like the calculation of minimum cost reserve capacities that is perhaps a little less difficult, the evolution of solution techniques (today we have good linear programming solvers in continuous or mixed variables) and means of computing (low-cost

powerful desk computers) enable the exact solving of instances of large sizes that are representative for operational situations. The same goes for the problem of frequency assignment; several models are applicable offering different degrees of precision in relation to the real problem, from the model based on restricted set T-coloring to the one that maximizes the quality of service and remains to be invented. For this problem, several solving methods have been tested and compared on diverse benchmarks that are more or less close to reality.

By limiting ourselves to network optimization in this chapter, we are still quite far from a completely satisfying outcome. Numerous points remain open; we can mention and non-exhaustively cite, for example:

- simultaneous optimization of dependent problems which are currently treated separately because of their complexity; for example, optimization of topology and dimensioning of nominal and reserve capacities on fixed networks or optimization of cellular topology and frequency assignment;
- optimization of multicriteria problems under constraints, such as antenna positioning, that are still very widely treated by aggregation of the criteria to optimize, and because of this do not enable interactive decision-making which is, however, indispensable in order to experimentally determine the domains of validity of the acceptable solutions.

Finally, many other network optimization models have not yet succeeded in a satisfactory manner so as to be usable in practice. That is particularly the case concerning the models for optimization of multi-wavelength optical networks that can use particular rings structures, or the optimization models for data networks with service quality constraints and/or with uncertain traffic demands. It is still the case for UMTS mobile radio networks whose deployment is made on the existing GSM network, thus posing cooperation/competition problems between the communication layers that could be mutually transferred according to the load and the services. Further, future network optimization models should integrate pricing and competition aspects as well as dimensioning aspects. That will certainly add a supplementary level of difficulty to telecommunication network optimization, which is a problem in perpetual evolution.

7.9. Bibliography

- [AHU 93] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Networks Flows – Theory, Algorithms and Applications*, Prentice Hall Inc., 1993.
- [BAL 89a] A. Balakrishnan, T.L. Magnanti and R.T. Wong, “A dual-ascent procedure for large-scale uncapacitated network design”, *Operations Research*, vol. 37, 716–740, 1989.

- [BAL 89b] A. Balakrishnan and S.C. Graves, "A composite algorithm for a concave-cost network flow problem", *Networks*, vol. 19, 175–202, 1989.
- [BEN 62] J.F. Benders, "Partitioning procedure for solving mixed variables programming problems", *Numerische Mathematik*, Vol. 4, 438–452, 1962.
- [BER 98] A. Berny and O. Sarzeaud, "Radio network optimization by means of neural networks", *Proceedings of the 36th Annual Allerton Conference on Communication, Control and Computing*, 177–186, September 1998.
- [BOI 83] L. Boithias, *Propagation des ondes radioélectriques dans l'environnement terrestre*, Dunod, Collection technique et scientifique des télécommunications, 1983.
- [BOX 77] Box F., "A heuristic technique for assignment of frequencies to mobile radio nets", *IEEE Transaction on Vehicular Technology* 27: 57–64, 1977.
- [CAL 96] Calegarie P., Guidéc F., Kuone P., Chamaret B., "Radio network planning with combinatorial optimisation algorithms", *Proc. of ACTS Mobile Communications Conference*, p. 707–713, 1996.
- [CAS 96] Castelino D.J., Hurley S., Stephens N.M., "A tabu search algorithm for frequency assignment", *Annals of Operations Research* 63: 301–319, 1996.
- [CHA 98] Chapman S.J., Hurley S., Kapp-Rawnsley R., "Optimising Radio Network Design", *NATO Symposium Frequency Assignment, Sharing and Conservation*, Aalborg, Denmark, Oct. 1998.
- [DOR 95] Dorne R., Hao J.K., "An evolutionary approach for frequency assignment in cellular radio networks", *Proc. of IEEE Intl. Conf. on Evolutionary Computation*, IEEE CS Press, p. 539–544, Perth, Australia, 1995.
- [DOR 98a] Dorne R., Hao J.K., Tabu search for graph coloring, "T-colorings and set T-colorings", Chapter 6, *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, et al. (Eds.), Kluwer Academic Publishers, 1998.
- [DOR 98b] Dorne R., Etudes des méthodes heuristiques pour la coloration, la T-coloration et l'affectation de fréquences, PhD Thesis, Université Montpellier II, 1998.
- [DUQ 93] D. Duque-Anton, B. Kunz and B. Ruber, "Channel assignment for cellular radio using simulated annealing", *IEEE Transaction on Vehicular Technology*, vol. 42, 14–21, 1993.
- [FLO 97] L. Floriani and G. Mateus, "An optimization model for the BST location problem in outdoor cellular and PCS systems", *Proceedings of International Teletraffic Congress*, 527–537, 1997.
- [FOR 95] S. Fortune, D. Gay, B. Kernighan, O. Landron and R. Valenzuela, "Wise design of indoor wireless systems", *IEEE Computational Science Engineering*, vol. 2 (1), 58–69, 1995.

- [FRU 96] T. Fruhwirth, J. Molwitz and P. Brisset, "Planning cordless business communication systems", *IEEE Expert Magazine*, vol. 11 (1), 662-673, 1996.
- [FRU 97] M. Frullone, G. Riva and P. Grazioso, "Advanced cell planning criteria for personal communication systems", *Proceedings of COST 259 TD*, 1997.
- [GAB 99] V. Gabrel, A. Knippel and M. Minoux, "Exact solution of multicommodity network optimization problem with general step cost functions", *Operation Research Letters*, vol. 25, 15-23, 1999.
- [GAM 86] A. Gamst, "Some lower bounds for a class of frequency assignment problem", *IEEE Transaction on Vehicular Technology*, vol. 35, 8-14, February 1986.
- [GLO 97] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, 1997.
- [GOE 93] X.M. Goemans and D.J. Bertsimas, "Survivable networks, linear programming, relaxations and the parsimonious property", *Mathematical Programming*, vol. 60, 145-166, 1993.
- [GOL 89] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, New York, 1989.
- [GRO 91] W.D. Grover, T.D. Bilodeau and D.D. Vanables, "Near optimal spare capacity planning in a mesh restorable network", *Proceedings of IEEE GLOBECOM, 2007-2012*, 1991.
- [GRÖ 95] M. Grötschel, C.L. Monma and M. Stoer, "Polyhedral and computational investigations for designing communication networks with high survivability requirements" *Operations Research*, vol. 43, 1012-1024, 1995.
- [HAJ 88] B. Hajek, "Cooling schedules for optimal annealing", *Mathematics of Operations Research*, vol. 13, 311-329, 1988.
- [HAL 80] W.K. Hale, "Frequency assignment: theory and application" *Proceedings of the IEEE*, vol. 68. no. 12, 1498-1573, 1980.
- [HAO 96] J.K. Hao and R. Dorne, "Study of genetic search for the frequency assignment problem", *Lecture Notes in Computer Science*, vol. 1063, 333-344, Springer-Verlag, 1996.
- [HAO 98] J.K. Hao, R. Dorne and P. Galinier, "Tabu search for frequency assignment in mobile radio networks", *Journal of Heuristics*, vol. 4, no. 1, 47-62, 1998.
- [HER 94] M. Herzberg and S.J. Bye, "Spare-capacity assignment in survivable networks for multi-link and node failures with hop limits", *Proceedings of the Sixth International Network Planning Symposium*, 381-386, 1994.
- [IRA 98] R.R. Iraschko, M.H. Macgregor and W.D. Grover, "Optimal capacity placement for path restoration in STM or ATM mesh-survivable networks", *IEEE/ACM Transactions on Networking*, vol. 6, 325-336, 1998.

- [KIR 83] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, "Optimization by simulated annealing", *Science*, vol. 22, 671–680, 1983.
- [KOS 99] A. Koster, Frequency assignment – models and algorithms, PhD thesis, Université de Maastricht, 1999.
- [LAG 99] X. Lagrange, P. Godlewski and S. Tabbane, *Réseaux GSM-DCS – Des principes à la norme.*, 4th edition, Réseaux et Télécommunications series, Hermes Science, 1999.
- [LEE 95] K. Lee, H. Lee, K. Park and S. Park, "Spare channel assignment for DCS mesh-restorable networks", *Proceedings of the 3rd Conference on Telecommunications Systems*, Nashville, 1995.
- [LUT 98] J.L. Lutton and J. Geffard, "Dimensionnement de réseaux sécurisés", *Actes du congrès FRANCO II*, 1998.
- [LUT 00] J.L. Lutton, D. Nace and J. Carlier, "Assigning spare capacities in mesh survivable networks" *Telecommunications Systems*, vol. 13, 441–451, 2000.
- [MAG 95] T.L. Magnanti, P. Mirchandani and R. Vachani, "Modeling and solving the two-facility network loading problem", *Operations Research*, vol. 43, 142–157, 1995.
- [MIC 92] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer Verlag, Berlin, 1992.
- [MIN 76] M. Minoux, "Multiflots de coût minimal avec fonctions de coût concaves", *Annales des Télécommunications*, vol. 31, 77–92, 1976.
- [MIN 89] M. Minoux, "Network synthesis and optimum network design problems: Models, solution methods and application", *Networks*, vol. 19, 313–360, 1989.
- [MUR 98] K. Murakami and H.S. Kim, "Optimal capacity and flow assignment for self-healing ATM networks based on line and end-to-end restoration", *IEEE/ACM Transactions on Networking*, vol. 6, 207–221, 1998.
- [POP 96] F. Poppe and P. Demeester, "An integrated approach to a capacited survivable network design problem", *Proceedings of the 4th Conference on Telecommunications Systems*, Nashville, 1996.
- [POP 98] F. Poppe and P. Demeester, "Economic allocation of spare capacity in mesh-restorable networks: Model and algorithm", *Proceedings of the 6th Conference on Telecommunications Systems*, Nashville, 1998.
- [REI 98] R. Reininger and A. Caminada, "Model for GSM radio network optimisation", *2nd ACM/IEEE MOBICOM International Workshop on Discrete Algorithms & Methods for Mobile Computing*, 1998.
- [REN 97] D. Renaud and A. Caminada, "Evolutionary methods and operators for frequency assignment problem", *Speedup Journal*, vol. 11, no. 2, 27–32, 1997.

- [SAK 90] H. Sakauchi, Y. Nishimura and S. Hasegawa, “A self-healing network with an economical spare-channel assignment”, *GLOBECOM'90*, 438–443, 1990.
- [SHE 94] H. Sherali, C. Pendyala and T. Rappaport, “Optimal location of transmitter for micro cellular radio communication network system design”, *IEEE Journal of Selected Areas in Communications*, vol. 4, 858–862, 1994.
- [SRI 00] V. Sridhar and J.S. Park, “Benders-and-cut algorithm for fixed-charge capacitated network design problem”, *European Journal of Operational Research*, vol. 125, no. 3, 622–632, 2000.
- [STO 94] M. Stoer and G. Dahl, “A polyhedral approach to multicommodity survivable network design”, *Numerische Mathematik*, vol. 68, 149–167, 1994.
- [TIO 95] S. Tiourine, C. Hurkens and J.K. Lenstra, “An overview of algorithmic approaches to frequency assignment problems”, *CALMA Symposium Final Report*, 53–62, 1995.
- [VAS 00] M. Vasquez and J.K. Hao, “A heuristic approach for antenna positioning in cellular networks” *Journal of Heuristics*, vol. 7, no. 5, 443–472, 2001.
- [YAG 71] B. Yaged, “Minimum cost routing for static network models”, *Networks*, vol. 1, 139–172, 1971.
- [ZAD 73] N. Zadeh, “On building minimum cost communication networks”, *Networks*, vol. 3, 315–331, 1973.

Chapter 8

Mission Planning for Observation Satellites

8.1. Introduction

We will study the planning problem for image acquisition made by a satellite in orbit for observation of the Earth. Satellite users emit *image requests*. To satisfy these, the satellite takes photographs of strips of ground with the help of a high resolution visible optical camera. As it turns around the Earth (a complete circuit is an *orbit*), and because of the rotation of the Earth, it is capable of photographing every point of the globe at different instants. Furthermore, it can photograph any strip from one or several orbits. The time necessary for a photograph of a strip is known since it depends on the speed of the orbiting satellite (which is constant) and the length of the strip. The *start date* of a strip photograph thus depends on the orbit from which it is realized and, given an orbit, this date must be chosen in an interval.

As it often happens that the totality of the images requested cannot be realized, planning a set of photographs comes down to choosing a start date for a subset of photographs, so as to satisfy a maximum number of requests while respecting the two following constraints: the camera takes at most one photograph at each instant, and a transition delay must be respected between any two consecutive pictures (this delay depends on the dates the pictures are taken).

From this general planning problem, we can distinguish two cases:

- (i) the images can be photographed in several time windows;
- (ii) each image can be realized in at most one time window.

In case (ii) each image can be realized from at most one orbit which, in our context, is verified when the planning horizon chosen is very short (approximately equal to a day). Since each orbit realizes distinct images, the problem of *short-term planning* comes down to a subset of problems independent of planning in an orbit. The problem of determining an orbital plan maximizing the number of images realized is called the problem of selecting the Maximum Shot number in an Orbit (MSO). On the other hand, case (i) corresponds to the choice of a more distant planning horizon. In this context, each image can be realized from several orbits. The problem of determining a *middle- or long-term* plan that maximizes the number of images realized is called the problem of selecting the number of Maximum Shots (MS). The objectives for MS and MSO are not similar: for the first, it is a case of arbitrating in time the realization of images, which means assigning an orbit to each image chosen; for the second, it is more the fine and precise optimization of the capacities of the satellite that is sought. These two problems have been studied in various contexts by numerous authors [BEN 96, FAB 92, FUC 90, GAS 89, MOO 75, TES 88, VAS 01].

The originality of the proposed approach is to study simultaneously MSO and MS by proposing a unique model for these two problems. Our model is based on a *discretization* of the set of possible start dates to realize each image. Given an instance of MSO or MS, the model first consists of defining a discrete set of shots for each image: one shot for an image represents the possibility of taking a photograph of the corresponding strip at a given instant. Let us now note that for MSO the discretization must be “fine”, while for MS a “rough” discretization suffices. In the set of the shots retained, three binary relations are then defined. These relations describe the possibilities for a pair of shots both belonging to the same plan. From the set of shots and binary relations, three graphs are constructed. On this base, MSO and MS can be formulated by using concepts from graph theory. The properties of these graphs define efficient algorithms to solve MS and MSO.

Concerning MS, it is shown that the resulting combinatorial problem cannot be solved precisely in an acceptable time. We must thus content ourselves with *approximate solutions* (at least for instances of realistic size). In consequence, we first propose several approximate algorithms to obtain these solutions, and then an algorithm to calculate an upper bound of the optimal value of the problem in order to evaluate the quality of the approximate solutions found. Concerning MSO, we propose an algorithm that operates on one of the graphs mentioned previously. The shots plan thus rapidly obtained is not necessarily optimal because of discretization. To determine the impact of discretization and more generally the relevance of our model, we must compare these approximate solutions with the solutions exhibited without a preliminary phase of discretization. Two other algorithms, taking into account the totality of the set of continuous start dates for each image, are constructed as follows: one is exact but can only solve in an acceptable time

instances of small size; the other is approximate and can treat instances of realistic size.

This chapter is organized in the following manner: in section 8.2, we describe the production capacities of the satellite considered and we present the preceding works dealing with MS and MSO; in section 8.3, we present the model proposed for MS and MSO; in section 8.4, we define our approach and algorithms for MS and MSO; finally, in section 8.5, we describe the numerical experiments. To facilitate reading this chapter, a certain number of technical proofs have voluntarily been omitted. The interested reader can find them in [GAB 01].

8.2. Description of the problem of planning satellite shots

8.2.1. Context: image acquisitions – constraints of the satellite used

We consider a satellite circling the Earth in quasi-circular orbits (north-south) at an altitude of 800 kilometers with a speed of 7 km/s. The duration of an orbit is about 100 minutes. The camera on the satellite takes photographs of strips, at most one each instant. Each band has a width of 60 km and a length between 70 and 700 km. This camera is visible optical, that is, unusable at night or in a cloud cover. An orbit is composed of a half-orbit in the daytime when the satellite is “active”, followed by a half-orbit at night when it is not. The camera can change its imaging axis in two independent directions, one “north-south”, called *pitching*, and the other “east-west” called *rolling* (see Figure 8.1).

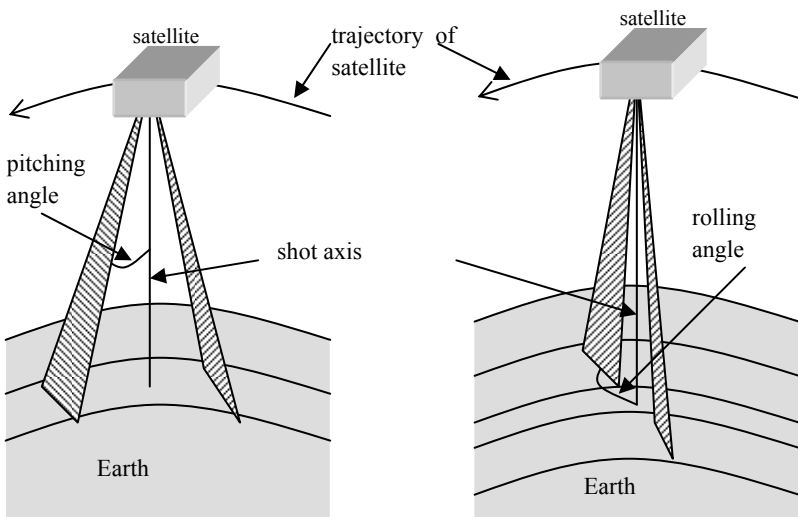


Figure 8.1. Movement of the satellite shot axis: “north-south” for pitching and “east-west” for rolling

The imaging axis can move simultaneously by rolling and pitching. Each of these movements is measured by the angles in degrees (Figure 8.2). Thanks to its rolling motion, the imaging camera can photograph points of the Earth belonging to a 900-km-wide band, centered on the satellite's trace to the ground. Due to its pitching movement, a given point on Earth can be photographed from a particular orbit at different instants belonging to a continuous interval. Thus, the satellite can photograph a point on Earth before or after being exactly above it.

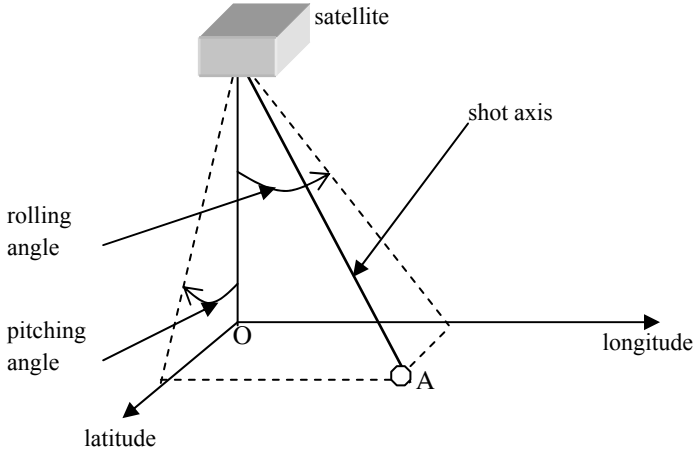


Figure 8.2. Rolling and pitching angles

The satellite users emit imaging requests by specifying the quality desired (resolution, luminosity, cloud cover, etc). Given the production capacities, an image can be made in different ways: from several orbits (the rolling angle of the desired axis is induced for each of the orbits) and, for a given orbit, at different start dates (the pitching angle of the desired axis is induced for each date). The quality of the image depends on the rolling and pitching angles with which the photograph of the corresponding strip was taken. Thus, for each image k are defined its duration of realization, noted g_k , a discrete set $\{\rho_k\}$ with possible rolling angles and, given a rolling angle, a possible interval with pitching angles, noted $[\theta_k^-, \theta_k^+]$, that satisfies the client's demand for quality. Note that θ_k^- is the front maximum pitching angle to photograph the corresponding strip and θ_k^+ is the rear maximum pitching angle. In fact, θ_k^- corresponds exactly to the earliest start date, noted t_{dk}^- , to realize image k from a given orbit, and θ_k^+ at the latest start date, noted t_{dk}^+ . Similarly, we can

define the values of the end dates as follows: the earliest end date t_{fk}^- is given by $t_{fk}^- = t_{dk}^- + g_k$ and the latest end date t_{fk}^+ is given by $t_{fk}^+ = t_{dk}^+ + g_k$.

For an image k , a shot x_i is equivalent to a start date to photograph k from a given orbit. Thus, x_i is characterized by a rolling angle ρ_{x_i} , a pitching angle θ_{x_i} (or similarly, a start date noted t_{dx_i}) and a duration of realization g_{x_i} . Note that the duration of realization is specific for image k , $g_{x_i} = g_k$.

8.2.2. Problems studied

Given a set of requested images and their associated shot, a *sequence of shots* is feasible if:

- at most one shot is realized at each instant,
- the time between two consecutive shots is sufficient to move the desired axis.

Given two shots x_i and x_j , the time $a_{x_i x_j}$ required to pass the rolling and pitching angles of x_i , to the corresponding angles of x_j , is called *transition time* and can be correctly approximated in the following manner:

$$a_{x_i x_j} = \max \{ t_\theta(\theta_{x_i}, \theta_{x_j}), t_\rho(\rho_{x_i}, \rho_{x_j}) \} \text{ with}$$

$$t_\theta(\theta_{x_i}, \theta_{x_j}) = \gamma_\theta + \frac{1}{\beta_\theta} |\theta_{x_i} - \theta_{x_j}| \quad [8.1]$$

$$t_\rho(\rho_{x_i}, \rho_{x_j}) = \gamma_\rho + \frac{1}{\beta_\rho} |\rho_{x_i} - \rho_{x_j}| \quad [8.2]$$

In equations [8.1] and [8.2], γ_θ , β_θ , γ_ρ and β_ρ are positive constants (β_θ and β_ρ are expressed in degrees per second, γ_θ and γ_ρ in seconds). We consider that the rolling and pitching angles are constant during shot. This hypothesis is non-restrictive and lets us simplify the model.

An important characteristic of the satellite is relative to the fact that the same strip cannot be photographed twice consecutively during the same orbit because of pitching limits, transition times and minimum realization times of image acquisition.

In MS the images can be made from different orbits. Thus, each image is associated with a non-enumerable set of shots characterized by different rolling and pitching angles. MS implies a choice in at most one rolling and pitching angle per image to define a feasible sequence of shots that satisfies a maximal number of requests.

In MSO the images can be realized from only one orbit. Thus, each image is associated with a continuous set of shots having the same rolling angle and different pitching angles. MSO implies choosing at most one pitching angle per image so as to define a feasible *orbital shot sequence* including a maximal number of images.

MS and MSO have been studied separately by numerous authors [BEN 96, FAB 92, FUC 90, GAS 89, MOO 75, TES 88, VAS 01] in various contexts.

Concerning MSO, the problem of determining an orbital shot sequence can be considered as a problem of sequencing particular to a machine with time windows and transition times dependent on the start dates of the tasks. This problem is known to be NP-complete and numerous authors have studied it (for the state of the art, see [GOT 93]). Let us nevertheless emphasize the work of [MOO 75] who specifically studied the problem that concerns us. However, the hypotheses of his problem differ from ours since he considers that the transition times depend on the shots, while in our case, the transition times depend on image acquisition (that is, the start dates chosen to realize these images). More recently, the problem of planning the shot at short-term from the Earth observation satellite SPOT5 gave rise to different studies [BEN 96, VAS 01]. It was a matter of finding an orbital shot sequence, but the resulting combinatorial problem is different in the sense where SPOT5 has three imaging cameras, two visible optical cameras and a radar without pitching capacity. Bensanna *et al.* propose two exact methods to solve the problem: the first based on integer linear programming, and the second on constraint programming. Vasquez and Hao use a metaheuristic to solve the problem in an approximate manner.

MS can be seen as an extension of MSO and, in this sense, it presents at least the same difficulties. The principal studies relative to MS were carried out by [GAS 89] and [TES 88]. The authors use, in both cases, artificial intelligence methods to solve in an approximate manner MS for satellites having only rolling capacities: networks of neurons for Gaspin and sequencing methods with “intelligent” backtracking for Tessier-Badie.

In the context of satellites having an imaging camera provided with pitching and rolling capacities, we propose a unique model for MS and MSO. This model is based on a discretization of the continuous set of shots for each picture. That lets us work on an enumerable sub-set of shots. In addition, the remarkable properties of this model let us propose efficient algorithmic solutions. Concerning MS, we

propose algorithms giving a feasible solution whose quality has been evaluated. Concerning MSO, it has been shown that the problem can be solved very easily with this model. However, this model requires a discretization that is not without impact when we try to finely optimize the satellite use (which is the case in MSO). To evaluate this impact, another model has been introduced in which all the possible start dates for each image are taken into account. By comparing the solutions obtained with each of these approaches in different instances, the consequences of the discretization are precisely measured.

8.3. Models and formulations of induced combinatorial problems

Let P ($|P|=n$) be the set of images to be realized. Each image i is associated with a set Ω_i of shots. We will call X the non-enumerable set of shots defined by: $X = \bigcup_{i \in P} \Omega_i$.

We define two binary relations on $X \times X$, called *enchainability* and *conflict*, noted respectively A and C . A pair of images (x_i, x_j) belong to A if and only if x_j can be realized after x_i , while (x_i, x_j) belongs to C if and only if $(x_i, x_j) \notin A$ and $(x_j, x_i) \notin A$.

DEFINITION 8.1. A and B are sets:

$$\begin{aligned} A &= \left\{ (x_i, x_j) \in (X \times X) : t_{dx_j} \geq t_{dx_i} + g_{x_i} + a_{x_i x_j} \right\} \\ C &= \left\{ (x_i, x_j) \in (X \times X) : (x_i, x_j) \notin A \wedge (x_j, x_i) \notin A \right\} \end{aligned} \quad [8.3]$$

PROPOSITION 8.1. *The relation of enchainability A is anti-symmetric and transitive. The relation of conflict C is symmetric and anti-transitive.*

PROOF. For the anti-symmetry, we must show that if $(x_i, x_j) \in A$ then $(x_j, x_i) \notin A$. However,

$$(x_i, x_j) \in A \Rightarrow t_{dx_j} \geq t_{dx_i} + g_{x_i} + a_{x_i x_j}$$

$$(x_j, x_i) \in A \Rightarrow t_{dx_i} \geq t_{dx_j} + g_{x_j} + a_{x_j x_i}$$

which implies, if we add the two preceding inequalities:

$$g_{x_i} + a_{x_i x_j} + g_{x_j} + a_{x_j x_i} \leq 0$$

This inequality cannot be verified since each of the terms is strictly positive. Thus, if $(x_i, x_j) \in A$ then $(x_j, x_i) \notin A$.

To prove the transitivity of A , we must show that if $(x_i, x_j) \in A$ and if $(x_j, x_k) \in A$ then $(x_i, x_k) \in A$. This is easily shown by expressing the relation of enchainability for the first couples according to the expression of times of transition given in section 8.2.2.

The capacities of satellite image acquisition are such that two distinct shots of the same image having the same rolling angle (that is, realized from the same orbit) cannot belong to the enchainability relation. On the other hand, two shots realized from two distinct orbits are always enchainable (because of the alternation of half-orbits of day and night).

Our model consists of choosing a discrete subset V from X so as to have in V the same number of shots for each image from each orbit. We thus define the relations of enchainability and conflict in $V \times V$. For each relation we classically construct the *graph* representing it: each element of V corresponds to a vertex in the graph, and each element (v, w) belonging to the relation is represented by the arc (respectively the edge) (vw) if the relation is anti-symmetric (respectively symmetric). In what follows, we call these graphs by the name of the relation that they represent and we write them $G_i = (V; E_i)$, where $i \in \{A, C\}$. As C is symmetric, the associated graph $G_C = (V; E_C)$ is non-oriented. According to Proposition 8.1, the graph associated with A is oriented and acyclic.

It is clear that every feasible shot sequence is represented by a *path* in G_A . The vertices belonging to a path in G_A correspond to a subset S of V so that $\forall (x_i, x_j) \in S \times S, (x_i, x_j) \notin E_C$. Thus, each set S of vertices constituting a path in G_A equally forms a *stable set* in G_C . Reciprocally, each stable set in G_C represents a feasible shot sequence since the vertices of a stable set represent a set of shots all enchainable two by two.

When the graphs considered represent an instance of MSO, each path in G_A corresponds to a feasible orbital shot sequence (with at most one shot per image). Thus, solving an MSO instance consists of determining a longest path in G_A .

When the graphs considered represent an instance of MS, a path in G_A (or equivalently, a stable set in G_C) corresponds to a feasible shot sequence, but this sequence can contain several shots of the same image. To model the necessity of selecting at most one shot per image, we must introduce a third binary relation in $V \times V$ called the *exclusion* relation, denoted E : a pair of shots belong to the exclusion relation if these shots let us photograph the same image.

DEFINITION 8.2. E is the set:

$$E = \{(x_i, x_j) \in V \times V : \exists k \in P, \{x_i, x_j\} \subseteq \Omega_k\} \quad [8.4]$$

We can easily see that E is symmetric and transitive.

We construct the non-oriented graph $G_E = (V; E_E)$, associated with the binary relation E . A sequence of feasible shots including at most one shot per image corresponds to a stable set at the same time in G_C and in G_E . Consequently, to solve an instance of MS we must determine a stable set of maximal cardinality in the graph $G = (V; E_C \cup E_E)$.

EXAMPLE 8.1. Let us consider 4 image requests d1, d2, d3 and d4 feasible in three distinct orbits: d1 is feasible from orbits 1, 2 and 3; d2, d3 and d4 are feasible from orbits 2 and 3. In addition, d1 cannot be realized from the same orbit as d2, d2 from the same orbit as d3 and d4, and d3 from the same orbit as d4. There exist nine shots: shots x_1, x_2 and x_6 allow us to realize d1 in orbits 1, 2 and 3 respectively, x_3 and x_7 for d2 in 2 and 3 respectively, x_4 and x_8 for d3 in 2 and 3, and x_5 and x_9 for d4 again in orbits 2 and 3. In Figure 8.3 first the orbits and the four strips to photograph are represented, and then the associated graph G . In graph G , a possible stable is constituted of the vertices x_1, x_3 and x_8 , which correspond to realize the request d1 from orbit 1, d2 from orbit 2, and d3 from orbit 3.

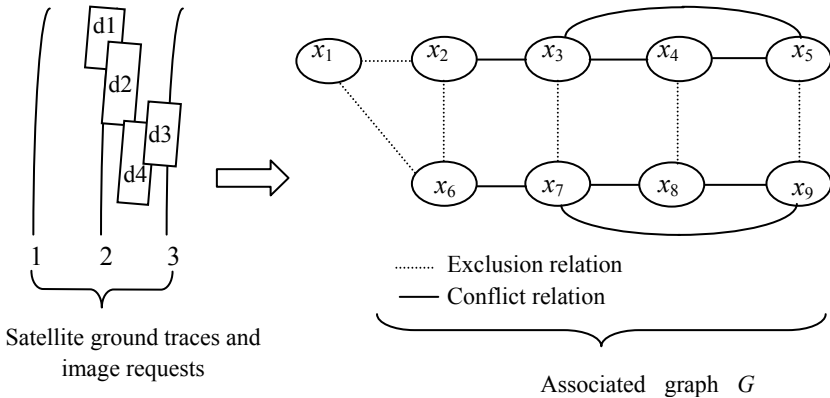


Figure 8.3. Graph G associated with an MS instance

By working on these graphs, the solutions obtained are approximate because of the discretization, and their quality will depend on the level of discretization. To validate the approach used, it is necessary to compare it to another solution taking

into account the continuous shot set. For that, we must introduce certain concepts no longer concerning shots, but images, that is, the continuous intervals of the shots.

It is sufficient to make this comparative study for MSO, insofar as the results are directly transposable to MS. Thus, following this section we will consider a set P of images to realize in the same orbit.

Starting with the relation A , we introduce another relation defined in $P \times P$, called *compatibility*, noted K : a pair of images belong to the relation of compatibility if there exists a pair of shots, one per image, belonging to the enchainability relation.

DEFINITION 8.3. *Let K be the set:*

$$K = \{(i, j) \in P \times P \text{ with } i \neq j : \exists x_u \in \Omega_i, \exists x_v \in \Omega_j, (x_u, x_v) \in A\} \quad [8.5]$$

PROPOSITION 8.2. *K is anti-symmetric, non-transitive and non-reflexive.*

PROOF. The anti-symmetry and the non-reflexivity derive from hypotheses made about the capacities of the satellite. It remains for us to demonstrate the non-transitivity. Given three images h , i and j , so that $(h, i) \in K$ and $(i, j) \in K$, let us suppose, without loss of generality, that there exists only one shot $x_u \in \Omega_h$ and only one shot $x_v \in \Omega_i$ so that $(x_u, x_v) \in A$, and that there exists only one shot $x_{v'} \in \Omega_i$ and only one shot $x_w \in \Omega_j$ so that $(x_{v'}, x_w) \in A$. If $t_{dx_{v'}} < t_{dx_v}$, then the shot sequences $(x_u, x_{v'}, x_w)$ and (x_u, x_v, x_w) are not feasible. In consequence, $(h, j) \notin K$ and there exists no feasible shot sequence that allows us to realize images h , i and j .

In fact K can be decomposed into two relations, one transitive, called *strong compatibility*, noted K_S , and the other non-transitive, called *weak compatibility* and noted K_W .

DEFINITION 8.4. *Let K_S and K_W be the sets:*

$$K_S = \{(i, j) \in P \times P : \forall x_u \in \Omega_i, \forall x_v \in \Omega_j, (x_u, x_v) \in A\}$$

$$K_W = \{(i, j) \in P \times P : (i, j) \in K \wedge (\exists x_u \in \Omega_i, \exists x_v \in \Omega_j, (x_u, x_v) \notin A)\} \quad [8.6]$$

Thus, each set of images (i_1, \dots, i_p) so that $i_j K_S i_{j+1}$, for $j=1, \dots, p-1$, induces the existence of a sequence of feasible shots that allows us to photograph these p images.

Based on these relations, we define the compatibility graph G_K whose vertices represent the images of P , numbered according to their geographical position in the direction “north-south”, plus two vertices noted s and e , that correspond respectively to the start and the end of the orbit. The arcs of G_K represent the relations of strong and weak compatibility. Notice that according to the properties of K , the graph G_K is without circuit. In addition, each path in G_K is not necessarily associated with a feasible sequence of shots. Example 8.2 below illustrates a graph G_K .

EXAMPLE 8.2. We consider the instance of MSO described in Table 8.1, so that $\gamma_0=50s$, $\gamma_p=50s$, $\beta_0=0.017rad/s$ and $\beta_p=0.017rad/s$.

Image number	1	2	3	4
Earliest start date	0	50	170	250
Earliest pitching angle	0.21	0.35	0	0.07
Latest start date	48	146	170	266
Latest pitching angle	-0.21	-0.49	0	-0.07
Rolling angle	0.1	-0.3	0.5	0.2
Duration of realization	40	100	10	30

Table 8.1. *An instance of MSO*

We thus obtain the following graph G_K :

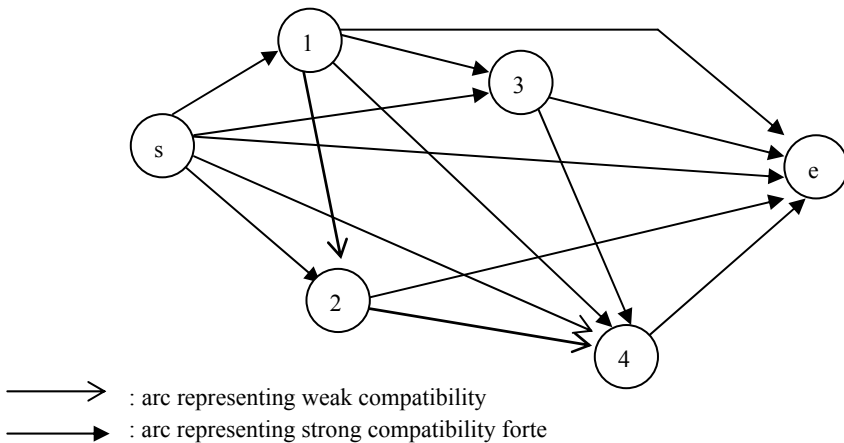


Figure 8.4. *Graph G_K associated with an instance of MSO*

8.4. Algorithms for MS and MSO

To treat combinatorial problems induced by MS and MSO, we propose some approximate and exact algorithms with two objectives: obtain feasible solutions in an acceptable time, and evaluate the quality of the approximate solutions obtained.

8.4.1. Solving MS

It has been shown in section 8.3 that solving MS means determining a stable set of maximal cardinality in $G=(V;E_C\cup E_E)$. The maximal cardinality of a stable set is called the *stability number*. For the graph $G=(V;E_C\cup E_E)$ not presenting any particular remarkable properties, the problem of determining a stable set of maximal cardinality in G is NP-hard. In consequence, the approach followed consists of first developing an approximate algorithm to determine in acceptable time a maximal stable set (in the sense of inclusion), and also to calculate an upper bound of the value of the optimal solution to evaluate the quality of the approximate solution obtained.

Different algorithms can be envisaged to determine a maximal stable set. Those inspired by *greedy heuristics* are the most classical. Others can be conceived on the base of the following property ([BER 73]).

PROPERTY 8.1. *If $\theta(H)$ designates the minimal number of cliques constituting a partition into cliques of the vertices of any graph H , and $\alpha(H)$ the stability number of H , then:*

$$\theta(H) \geq \alpha(H). \quad [8.7]$$

The determination of a minimal partition into cliques of V in G presents a double interest:

- on the base of this partition, a maximal stable set can be constructed by selecting in each clique at most one vertex (by obviously taking care to never obtain two adjacent vertices);
- the cardinality of this partition gives an upper bound of the stability number.

The problem of determining a minimal partition into cliques is a problem as difficult as that of determining a maximum stable set. However, graph G , constructed as the union of G_C and G_E , presents important properties for this problem that we mention below.

PROPOSITION 8.3. *It is possible to determine in polynomial time a minimum partition into cliques of V in G_E .*

PROOF. By construction, the exclusion graph G_E is constituted of a collection of cliques that are two by two disjoint.

PROPOSITION 8.4. *It is possible to determine in polynomial time a minimum partition into cliques of V in G_C .*

PROOF. Let us first remember some definitions and results.

The *complementary graph*, noted $\overline{H} = (N; \overline{E})$, of a graph $H = (N; E)$ is defined as follows:

$$\forall (i, j) \in N \times N \text{ with } i \neq j, (ij) \notin E \Rightarrow (ij) \in \overline{E} \text{ et } (ij) \in E \Rightarrow (ij) \notin \overline{E}$$

The problem of *minimum coloring* of the vertices of a graph H is to attribute a color to each of its vertices so that no two vertices connected by an edge have the same color, and the total number of distinct colors attributed is minimal.

It is evident that, in any graph H , a minimum partition of its vertices into cliques corresponds to a minimum coloring in its complementary graph \overline{H} (it suffices to attribute the same color for a minimum coloring to the vertices belonging to the same clique of the minimum partition).

A graph $H = (N; E)$ is a *comparability graph* if and only if it is possible to orient its edges (that is, to pass from a non-oriented graph to an oriented graph) to obtain a graph representing a transitive relation.

It turns out that the minimum coloring problem is *polynomial* in comparability graphs [EVE 72].

Now, the complementary graph of G_C , is a comparability graph. In effect, if we transform the transitive oriented graph G_A into a non-oriented graph, we obtain exactly the complementary graph G_C .

Consequently, the problem of minimum partition into cliques of V in G_C can be solved in polynomial time by determining a minimum coloring in its complementary graph. \square

PROPOSITION 8.5. *Any clique of G is a clique of G_E or a clique of G_C .*

PROOF. A clique of G that is neither a clique of G_C nor a clique of G_E necessarily contains one of the two subgraphs (1a) or (2a) in Figure 8.5.

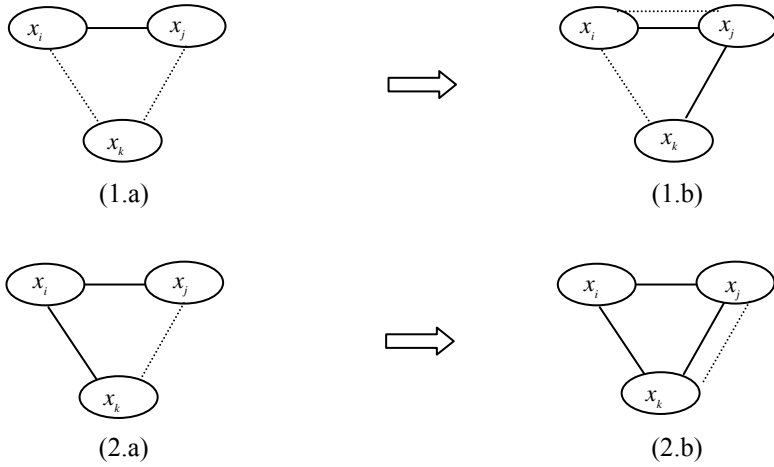


Figure 8.5. Cliques in G

In the subgraph (1a) we have: $(x_i x_j) \in E_C$ and $(x_i x_k), (x_j x_k) \in E_E$. As the exclusion relation is symmetric and transitive, $(x_i x_k)$ and $(x_j x_k) \in E_E$ implies $(x_i x_j) \in E_E$ (we then get the subgraph (1b)).

In the subgraph (2a), we have $(x_i x_j), (x_i x_k) \in E_C$ and $(x_j x_k) \in E_E$. Since $(x_i x_j)$ belongs to E_C , x_i and x_j represent two feasible shots in the same orbit, it is the same for x_i and x_k . Consequently, x_i, x_j and x_k represent three feasible shots in the same orbit, and as x_j and x_k are two shots that allow us to realize the same image, $(x_j x_k)$ is necessarily an edge of the conflict graph (we then end up with subgraph (2b)).

G can thus not have as a subgraph either (1a) or (2a).

PROPOSITION 8.6. *Let L_i be a clique of G_E belonging to a minimum partition into cliques of V on G noted L ; it is always possible to add vertices in L_i so that it becomes maximal in G_E without destroying the minimality of L .*

PROOF. Let L_i^E be a maximal clique of G_E so that $L_i \subset L_i^E$. Suppose without loss of generality that $L_i = L_i^E \setminus \{x_j, x_k\}$. Since L is minimum and because of the very particular structure of G_E , x_j and x_k cannot form the same clique of L ; x_j and x_k thus belong necessarily to two distinct cliques formed by the edges of the conflict graph; let us note these cliques L_j^C and L_k^C . Consequently, it is possible to exclude x_j and x_k from L_j^C and L_k^C respectively and to add them in L_i without changing the cardinality of L .

Propositions 8.5 and 8.6 enable us to affirm that there exists a minimum partition in cliques of V in G that contains either only cliques of G_C , or only maximal cliques of G_E , or both. It is a partition of this type (not necessarily minimum) that we propose to determine in polynomial time.

The algorithm proposed to determine a partition into cliques of V in G is done as follows. At initialization, this algorithm determines a minimum partition in cliques of V in G_E , noted $L^E = \{L_1^E, \dots, L_n^E\}$ (for each i , the vertices of L_i^E represent all the shots to realize the image request i). We note W a set of vertices included in V , $G_C(W)$ (respectively $G_E(W)$) the conflict subgraph (exclusion subgraph, respectively) engendered by W and, $L^C(W)$ (respectively $L^E(W)$) the minimum partition in cliques of W in $G_C(W)$ (respectively in $G_E(W)$). At initialization, W contains all the vertices of V . Then at each iteration we calculate $L^C(W)$ with the help of the procedure *clique* applied to $G_C(W)$. If $L^C(W)$ does not contain a clique of cardinality 1 (called 1-clique), this iteration is the last and the partition of V in G retained, noted L , and is constituted of the cliques $L^C(W)$ and the cliques $L^E(V \setminus W)$. If not, for each of the 1-cliques of $L^C(W)$, if the vertex that constitutes it belongs in L^E to a clique of size greater than 1, say L_j^E is such a clique, we exclude the vertices L_j^E from the set W . When all the 1-cliques have been treated, if W has not been modified, this iteration is the last and L is constituted in the same way as previously; if not we go on to the following iteration. At the end of the last iteration, we retain as partition in cliques of V in G the partition of the smallest cardinality between L and L^E .

The *clique* procedure applied to a graph H , complement of a comparability graph, determines a minimum partition of its vertices in cliques. For that it uses the algorithm presented in [EVE 72] to color the vertices of a comparability graph in a minimum number of colors.

Algorithm 8.1, presented below, has a *complexity* of $O(n|V|^2)$. In effect, at each iteration, the step necessitating the most elementary operations is relative to the determination of a minimum partition in cliques of W in $G_C(W)$ with the help of the *clique* procedure. The time necessary for *clique* to find a minimum partition in cliques of the vertices of a graph of order k is in $O(k^2)$ [EVE 72]. If in the worst case, n iterations are effected, that is, at each iteration a particular clique of G_E is excluded from W , the algorithm proposed exhibits a partition in cliques in a time in $O(n|V|^2)$.

Start

```

 $W \leftarrow V$ 
 $L^E \leftarrow \{L_1^E, \dots, L_n^E\}$ 
While  $W \neq \emptyset$  do
     $L^C(W) \leftarrow \text{clique}(G_C(W))$ 
     $W' \leftarrow \emptyset$ 
    For each  $L_i^C \in L^C(W)$  do
        If  $|L_i^C| = 1$  then
            Let  $L_j^E$  be as  $L_i^C \subset L_j^E$ 
            If  $|L_j^E| > 1$  then
                 $W' \leftarrow W' \cup L_j^E$ 
            End If
        End If
    End For
    If  $W' = \emptyset$  then
         $L \leftarrow L^C(W) \cup L^E(V \setminus W)$ 
         $W \leftarrow \emptyset$ 
    Else
         $W \leftarrow W \setminus W'$ 
    End If
End While
If  $|L| \geq |L^E|$  then
     $L \leftarrow L^E$ 
End If

```

End**Algorithm 8.1.** *Approximate algorithm of partition in cliques*

To determine a maximal stable set in G , we have compared three distinct approximate algorithms. These three algorithms determine three stable sets, S_1 , S_2 and S_3 , in a greedy manner with the help of different heuristics.

The first algorithm, A_1 , is a classical algorithm. We note T the set of the vertices that could belong to a stable set. At initialization of A_1 , T is constituted of all the vertices of V . Then at each iteration, the vertex of the smallest degree in $G(T)$ (the subgraph of G engendered by T) is selected to belong to S_1 , and all the vertices adjacent to the selected vertex are excluded from T . A_1 terminates when T is equal to the empty set.

The second algorithm, A_2 , functions equally on a greedy principal. The heuristic used is related to the particularities of G constructed as the union of two graphs: the exclusion graph and the conflict graph. At initialization, T is constituted of all the vertices of V . And, at each iteration, we augment S_2 by the vertex i that, among the vertices of T having the smallest degree in $G_E(T)$ (the exclusion subgraph engendered by T), has the smallest degree in $G_C(T)$ (the conflict subgraph engendered by T). Then, we eliminate from T all the vertices adjacent to i . A_2 terminates when T is equal to the empty set.

The third approximate algorithm, noted A_3 , constructs S_3 , by selecting at most one vertex in each clique of L (remember that L is the partition in cliques of the vertices of G found with the help of Algorithm 8.1) so as to never retain two adjacent vertices. At initialization, T is constituted of all the vertices of V . As long as T is different from the empty set, at each iteration, we select the vertex i from T that has, within the clique of the smallest cardinality, the smallest degree in $G(T)$, and we exclude from T the vertices adjacent to i . We notice that if S_3 has the same cardinality as L , S_3 is a maximum stable set and L a minimum partition.

These different algorithms enable us to solve MS in an approximate manner by bounding the stability number of G in the following way:

$$\max \{|S_1|; |S_2|; |S_3|\} \leq \alpha(G) \leq |L|$$

8.4.2. Solving MSO in the discrete case

Since the problem of determining a longest path in an oriented graph without cycles is polynomial, each instance of MSO can be solved in polynomial time by determining the longest path from s to e in G_A (the algorithm used is presented in Chapter 2 of this book and in [GON 85]).

8.4.3. Solving MSO in the continuous case

It is clear that the quality of the solutions obtained with the algorithms presented in sections 8.4.1 and 8.4.2 depends on the level of discretization. In order to evaluate this quality we must compare the solutions with others found by considering the complete set of shots.

This comparative work has been conducted uniquely for MSO since the conclusions are transposable to MS. As MSO is NP-complete in the continuous case, two algorithms are proposed: one is exact but exponential, and the other approximate and polynomial. Since these two algorithms are based on the

construction of the compatibility graph G_K , we will begin by showing that its construction is done in polynomial time.

In the following, we define an orbital sequence of shots either as a list of shots or as a list of images that this sequence lets us realize.

Construction of the compatibility graph

The topic of this section is to specify how the construction of G_K can be realized in polynomial time. For that, given any two shots i and j in an orbit, we must characterize, which is done in Property 8.2, in which cases (i, j) belong to the relation of strong compatibility K_S and in which cases (i, j) belong to the relation of weak compatibility K_W .

DEFINITION 8.5. *A state of the satellite is a pair (θ, t) , where θ represents a pitching angle and t a date. For an image i , the set of the start states (end states) of i is the set of the states of the satellite corresponding to the interval of the start dates (end dates) of i . The set of the start states (end states) of i is represented in the plan (θ, t) by a segment.*

We note $Z_{TR}(\theta_i, t_f(\theta_i))$ the set of states in which the satellite can be found after having realized the photograph of image i with the pitching angle θ_i (at the date $t_f(\theta_i)$) and the rolling angle ρ_i . $Z_{TR}(\theta_i, t_f(\theta_i))$ is defined by:

$$Z_{TR}(\theta_i, t_f(\theta_i)) = \{(\theta, t) : t \geq \max\{t_{f_i} + t_{\theta}(\theta_i, \theta), t_{f_i} + t_{\rho}(\rho_i, \rho)\} \mid |\theta| \leq \theta_{\max}\} \quad [8.8]$$

It is then possible to formally redefine the relations of strong and weak enchainabilities as follows. Given two images i and j :

- $(i, j) \in K_S$ if the set of the start states of j is contained in $Z_{TR}(\theta_i^+, t_f(\theta_i^+))$;
- $(i, j) \in K_W$ if there exists at least one start state of j in $Z_{TR}(\theta_i^-, t_f(\theta_i^-))$.

Now, it can be shown that, to determine if the set of start states of j is contained in $Z_{TR}(\theta_i^+, t_f(\theta_i^+))$ we must only test if the earliest start state of j is included in $Z_{TR}(\theta_i^+, t_f(\theta_i^+))$. Similarly, to determine if there exists at least one start state of j in $Z_{TR}(\theta_i^-, t_f(\theta_i^-))$, we must only test if the last start state of j is in $Z_{TR}(\theta_i^-, t_f(\theta_i^-))$. From which we obtain Property 8.2, below.

PROPERTY 8.2.

- $(i,j) \in K_S$ if and only if the earliest start state of j belongs to $Z_{TR}(\theta_i^+, t_f(\theta_i^+))$;
- $(i,j) \in K_W$ if and only if the latest start state of j belongs to $Z_{TR}(\theta_i^-, t_f(\theta_i^-))$.

Property 8.2 is sufficient to show that the construction of G_K can be done in polynomial time.

The exact algorithm

Once G_K is constructed, we propose an exact algorithm based on a *branch and bound* method to determine the longest path from s to e in G_K representing a feasible orbital sequence of shots.

Before presenting it, we will introduce G_K' , a partial graph of G_K obtained by suppressing from G_K the transitive arcs of the relation of strong compatibility. In effect, if there exists $k \in P$ so that $(i,k) \in K_S$ and $(k,j) \in K_S$, then that signifies we can always execute another image between the realization of i and j . In this case, the arc (i,j) of G_K is taken off. Thus, to determine an orbital sequence of shots that maximizes the number of images photographed, it suffices to work on G_K' rather than on G_K . Graph G_K' is illustrated in Example 8.3.

Let $\Gamma(i)$ be the set of successors of the vertex i in G_K' , and let $l(i)$ be a longest path (in the sense of the number of vertices) from i to e in G_K' (the vertex i is not taken into account in $l(i)$). Since G_K' is acyclic, determining a longest path between two vertices in this graph is done in polynomial time. This path then represents an orbital sequence of shots that is not necessarily feasible.

The exact algorithm is based on a progressive exploration of the solution space (the set of the feasible orbital sequences of shots), following the branch and bound principle. This exploration brings us to define an arborescence, whose root represents all the orbital sequences of shots, and each vertex corresponds to a particular subset of orbital sequences of shots. A vertex $S = (s, i_1, \dots, i_k)$ represents all the orbital sequences of shots beginning with the realization of images i_1 to i_k . The evaluation of the vertex S is given by $f(S) = k$; it represents a lower bound of a maximum orbital sequence of shots, of which the k first elements are i_1, \dots, i_k . Let $f^* = f(S^*)$, where S^* is the current solution having the maximum evaluation. At each step, we consider a vertex S without successor. We then sort the elements j of $\Gamma(i_k)$ in decreasing order of the longest path from j to e in G_K' and we obtain

$\Gamma(i_k) = (i_{k_1}, \dots, i_{k_m})$. Thus, S can have m sons $S_{i_{k_1}}, \dots, S_{i_{k_q}}, \dots, S_{i_{k_m}}$ with $S_{i_{k_q}} = (s, i_1, \dots, i_k, i_{k_q})$. The vertex $S_{i_{k_q}}$ is conserved in the arborescence if it is possible to successively realize the images i_1, \dots, i_k and i_{k_q} , that is, if there exists a sequence of feasible image acquisitions that allows us to take the set of photographs $(i_1, \dots, i_k, i_{k_q})$; if not, the vertex $S_{i_{k_q}}$ is taken off.

To avoid the exploration of all feasible solutions, we associate with each new vertex $S = (s, i_1, \dots, i_k)$ created, an upper bound for the number of images that can be taken. This upper bound is defined by $w(S) = k + l(i_k)$. Thus, each vertex S as $w(S) < f^*$ induces a sub-arborescence that will not need to be explored. If $f(S) = w(S)$, S is a leaf of the search tree.

The branch and bound algorithm begins by creating the vertices of the arborescence according to a depth-first strategy to obtain a first f^* . Then it returns to the father of the vertex S^* in order to visit its brothers (in decreasing order of the corresponding values w). If one of the vertices recently visited has a value f such that $f > f^*$, then f^* is updated. All the sub-trees rooted at the vertex S' such that $w(S') < f^*$ are cut.

The critical point of this algorithm is to test whether a set of images $(i_1, \dots, i_k, i_{k_q})$ can be photographed, knowing that the subset (i_1, \dots, i_k) represents a feasible orbital sequence of shots. In fact, the feasible character of this subset induces for each image an eventual reduction of the initial set of start dates. Thus, to determine if the image i_{k_q} is feasible after the sequence (i_1, \dots, i_k) , we have only to determine if i_{k_q} is compatible with the new set of possible start dates of i_k . That is done by using the result of Property 8.2.

The exact algorithm explores in the worst-case 2^{n-1} vertices. The calculation time augments exponentially with the size of the problem. More precisely, the calculation time is a function of the size of the solution: if the maximum orbital sequence of image acquisitions is constituted of k images, then the arborescence covered by the exact algorithm is in $O(2^{k-1})$.

The development of the algorithm by branch and bound is illustrated in Example 8.3 below.

EXAMPLE 8.3. We will again take the instance of MSO presented in Example 8.2. Starting from the graph G_K , we obtain graph G_K' illustrated in Figure 8.6.

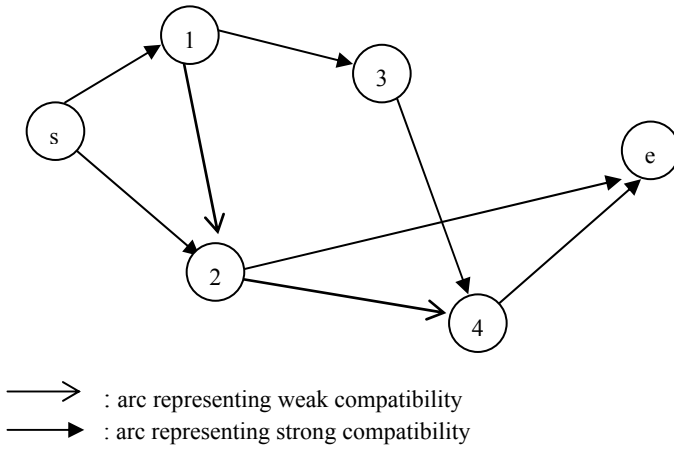


Figure 8.6. Graph G_K' associated with an MSO instance

Following G_K' , we deduce:

$$\Gamma(s) = \{1, 2\}, \Gamma(1) = \{2, 3\}, \Gamma(2) = \{4, e\}, \Gamma(3) = \{4\} \text{ and } \Gamma(4) = \{e\}$$

$$l(s) = 3, l(1) = 2, l(2) = 1, l(3) = 1, l(4) = 0.$$

The application of the branch and bound algorithm will generate the arborescence in Figure 8.7.

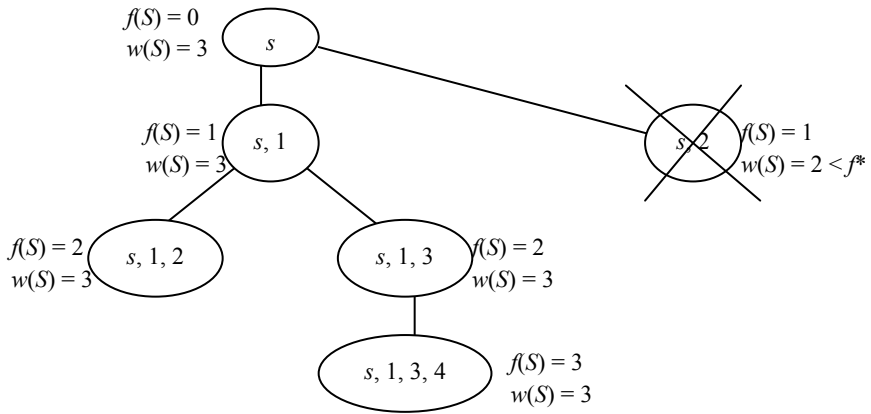


Figure 8.7. Arborescence generated by the exact algorithm

From root s , two sons $(s,1)$ and $(s,2)$ are created whose upper bounds are 3 and 2 respectively. At this stage f^* is equal to 1. The exploration continues to leave from $(s,1)$ because it is the vertex without successor having the best upper bound. Two sons are created: $(s,1,2)$ and $(s,1,3)$ and f^* is equal to 2. Concerning $(s,1,2)$, as no other image acquisition can be realized after 1 and 2, this vertex is a leaf. The exploration starts again in vertex $(s,1,3)$ that only admits one son: $(s,1,3,4)$, and as its upper bound is equal to its evaluation, this vertex is also a leaf of the arborescence. At this stage, f^* is worth 3. There is still vertex $(s,2)$ to explore; its upper bound being less than f^* , it is useless to continue the exploration from this vertex. The optimal solution is thus $\{1, 3, 4\}$ of value 3.

Approximate algorithm

To solve large-size MSO instances, an approximate polynomial algorithm has been developed (see Algorithm 8.2). This algorithm operates on graph G_K , where we associate with each vertex $i \in P \setminus \{s, e\}$ the segment $D(i)$ that represents the initial set of the start dates for image i and $F(i)$ the set of the end states where the satellite can be found after having realized image i . Algorithm 8.2 is based on the following idea: knowing the longest feasible path from s to i , of value $\lambda(i)$, test for all successors j of i if it is more useful to pass by i to obtain a longer feasible path from s to j . Do not forget this approach is only possible because G_K is without circuit and the vertices of G_K are numbered so that at iteration i , the longest path from s to i is known.

Start

For all $i \in P$ **do**

$$D(i) \leftarrow \bigcup_{\theta_i \in [\theta_i^+, \theta_i^-]} (\theta_i, t_d(\theta_i))$$

$$F(i) \leftarrow Z_{TR}(\theta_i^-, t_f(\theta_i^-))$$

End For

For all $i \in P \cup \{e, s\}$ **do** $\lambda(i) \leftarrow 0$

End For

For all $i = s, 1, \dots, |P|$ **do**

For all $j \in \Gamma(i)$ **do**

If j is feasible after i and **If** $\lambda(i) + 1 > \lambda(j)$

Then $\lambda(j) \leftarrow \lambda(i) + 1$

Update $D(j)$ and $F(j)$

End If

End For

End For

End.

Algorithm 8.2. *Approximate polynomial algorithm to solve large-size instances of MSO*

Concerning the complexity of Algorithm 8.2, we notice that the vertices G_K are treated only once, and for each, the algorithm considers all its successors. Consequently, the complexity of Algorithm 8.2 is linear in relation to the number of arcs of G_K (at worst $O(n^2)$).

The behavior of these algorithms and their effectiveness are analyzed in the following section on realistic instances generated randomly.

8.5. Experiments and numerical results

8.5.1. Performances of approximate algorithms proposed to solve MS

A realistic instance of MS is generated by randomly positioning n ($n \in \{50, 100, 200, 300\}$) strips to photograph in a region 2,000 km wide by 3,000 km long, situated on the equator or 45° latitude. The photographs of these strips can be taken during the next 26 days. The parameters adopted are the following:

- during the next 26 days, the satellite describes 27 orbits where it is possible to photograph any point in the region situated at the equator (against 38 in the region situated at 45° latitude); ground traces of the orbits are fixed and are about 100 km distant at the equator against 70 km at 45° latitude;
- the strips to be photographed are 70 km in length;
- the maximal rolling angle ρ_i authorized to photograph the strip i is chosen randomly in the interval $[0, 30^\circ]$. Thus, image i can be realized with any rolling angle included in the interval $[-\rho_i, \rho_i]$;
- only a pitching angle equal to 0° is allowed to photograph the strips
- concerning the displacement speed of the image acquisition axis, we put: $\gamma_0 = \gamma_p = 50$ s, and $\beta_0 = \beta_p = 1^\circ$ /s.

For an instance, knowing the localization of the n images to realize and the interval of the rolling angle authorised to photograph each of them, it is possible to enumerate the set of image acquisitions to realize during the next 26 days. Notice that, everything else being equal, the number of image acquisitions is higher when we localize the strips in the region at 45° latitude rather than at the equator.

700 instances of MS have been randomly generated. Each of these instances is represented by a graph G . The order of these graphs (the number of vertices) varies from 194 to 1,460, and is, on average, 756. The mean density of these graphs (number of edges divided by the number of vertices in a graph) is 8, the weakest density is 3, while the strongest is 15.5.

In each of these graphs S_1 , S_2 , S_3 , and L are determined. The execution times of the algorithm that allow us to determine L are very weak since they vary from zero to five minutes.

Table 8.2 presents, according to the number of images requested, the mean, maximal, and minimal values of the following ratio:

$$\frac{\max_{i=1,2,3} \{|S_i|\}}{|L|} \quad [8.9]$$

These results allow us to conclude that partitioning in cliques determined by Algorithm 8.1 lets us bound in a very satisfactory manner the stability number of the randomly generated graphs. The mean value of the lower bound for $\alpha(G)$ compared to the upper bound of $\alpha(G)$ in all the instances considered is 0.945.

	$n = 50$	$n = 100$	$n = 200$	$n = 300$
Mean	0.997	0.938	0.9	0.945
Maximum	1	1	0.992	0.985
Minimum	0.94	0.837	0.819	0.891

Table 8.2. Mean, maximum and minimum values of [8.9]

Table 8.3 presents the mean, maximum, and minimum values of the following ratios:

$$r_i = \frac{|S_i|}{|L|} \forall i = \{1, 2, 3\} \text{ and } r_4 = \frac{\max_{i=\{1,2,3\}} \{|S_i|\}}{|L|}. \quad [8.10]$$

We notice that the ratio r_3 is on average higher than r_1 and r_2 . This result is interesting because A_3 is the algorithm that most “exploits” the properties of the model to find solutions. However, it remains preferable to use the three algorithms to find three distinct stable sets to retain the stable set of maximum cardinality.

	r_1	r_2	r_3	r_4
Mean	0.937	0.93	0.941	0.945
Maximum	1	1	1	1
Minimum	0.819	0.8	0.797	0.819

Table 8.3. Mean, maximum and minimum values of [8.10]

The results of these numerical experiments are thus very satisfying: they highlight the effectiveness of the proposed algorithms to manage the stability number in a graph, and thus obtain an approximate solution close to the optimum.

8.5.2. Performances of approximate algorithms proposed to solve MSO

Realistic instances of MSO are randomly generated by random positioning of a set of strips to photograph during a quarter of an orbit. The parameters adopted are the following:

- the strips (70 km long) to photograph are localized randomly in a zone 900 km wide by 10,000 km long centered on the satellite's ground trace; the position of a strip in such a zone induces the rolling angle of the desired axis necessary to realize the image;
- with each strip is associated a maximum front pitching angle chosen randomly in the interval $[0, 30^\circ]$ (the rear maximum pitching angle is then its opposite);
- the displacement speed of the image acquisition axis is chosen as in section 8.5.1.

Following these parameters, we randomly generate 100 graphs G_K of order 102 where we test the approximate Algorithm 8.2. The maximum degree in these graphs belongs to the interval $[32, 47]$, with a mean degree 38.53. The density of each graph belongs to the interval $[18.54, 25.82]$ and its mean value is 21.61. The maximum degree of weak compatibility arcs belong to the interval $[16, 27]$ while the maximum degree of strong compatibility arcs is in $[19, 29]$. The mean densities concerning the weak compatibility arcs and the strong compatibility arcs are respectively equal to 9.99 and 11.61. For the set of these data, the solutions obtained have cardinalities varying from 13 to 17 images. For each of these graphs we have applied the exact algorithm and Algorithm 8.2.

The performances of Algorithm 8.2 are satisfactory. The mean approximation ratio defined by the cardinality of the approximate solution divided by the cardinality of the exact solution has a value of 0.939. At the minimum, it is equal to 0.818. In addition, in 38% of the cases the optimal solution is obtained.

We then compared the algorithms in section 8.4.3 with the algorithm described in section 8.4.2 (the discrete case). We randomly generated MSO instances characterized by 50, 100 and 200 images (100 instances per case). For each instance, we consider three discretizations possible:

- a single image acquisition for each image i : this image acquisition is realized with a pitching angle 0° , inducing a start date noted t_{di}^0 ;
- three image acquisitions for each image i : their start dates are t_{di}^- , t_{di}^0 and t_{di}^+ ;
- five image acquisitions for each image i : their start dates are t_{di}^- , $t_{di}^- + (t_{di}^0 - t_{di}^-)/2$, t_{di}^0 , $t_{di}^+ - (t_{di}^+ - t_{di}^0)/2$ and t_{di}^+ .

For each instance, we determine the cardinalities $p^{(i)}$, $i = a, b, c$ of the orbital image acquisition sequences obtained by applying the algorithm in section 8.4.2 with a discretization of type (a), (b), (c) respectively. We determine equally the cardinalities p^* or p' of the orbital image acquisition sequences obtained either with the exact algorithm, or with the approximate algorithm.

For the instances characterized by 50 images to photograph, we calculate the mean, maximum and minimum differences between p^* and $p^{(i)}$ with $i = a, b, c$. The results are presented in Table 8.4.

	$p^* - p^{(a)}$	$p^* - p^{(b)}$	$p^* - p^{(c)}$	p^*
Mean	1.59	0.70	0.48	8.6
Maximum	3.00	2.00	1.00	10
Minimum	1.00	0.00	0.00	7

Table 8.4. Results for instances of 50 images

For the instances characterized by 100 and 200 images to photograph, we make the same calculations as before but in place of p^* we consider p' (because the calculation times to obtain p^* are too great for instances of these sizes). The results are presented in Tables 8.5 and 8.6.

	$p' - p^{(a)}$	$p' - p^{(b)}$	$p' - p^{(c)}$	p'
Mean	1.63	0.67	0.18	15
Maximum	3.00	2.00	1.00	17
Minimum	1.00	0.00	-1.0	13

Table 8.5. Results for instances of 100 images

	$p' - p^{(a)}$	$p' - p^{(b)}$	$p' - p^{(c)}$	p'
Mean	2.64	1.34	0.47	29
Maximum	4.00	3.00	2.00	32
Minimum	1.00	0.00	-1.0	27

Table 8.6. Results for instances of 200 images

We notice that type (c) discretization produces approximately the same results as we obtained by considering the continuous set of image acquisitions. In the worst case, the orbital sequence of image acquisition determined by the discretization (c) included two fewer image acquisitions, and this difference is a mean of 0. The fact of operating with a discretization does not seem to reduce the quality of the solutions obtained.

8.6. Conclusion

The formalism of operations research lets us define a unique model for MS and MSO and thus clarify the induced combinatorial problems. In addition, by calling on certain operations research tools, it was possible to construct efficient solution methods that helped us make decisions in the short- and medium-term planning phase (see [GAB 94]).

8.7. Bibliography

- [BEN 96] E. Bensanna, G. Verfaillie, J.-C.Agnès, N. Bataille and D. Blumstein, “Exact and inexact methods for the daily management of an Earth observation satellite”, *Proceedings of the 4th International Symposium on Space Mission Operations and Ground Data Systems*, Munich, Germany, 1996.
- [BER 73] C. Berge, *Graphs and Hypergraphs*, North Holland, Amsterdam, 1973.
- [EVE 72] S. Even, S. Lempel and A. Pnueli, “Permutation graphs and transitive graphs”, *Journal of the Association for Computing Machinery*, vol. 19, no. 3, 400–410, 1972.
- [FAB 92] F. Fabiani, Planification de missions d’un satellite d’observation, Mémoire de DEA, ENSAE, Toulouse, 1992.
- [FUC 90] J.F. Fuchs, A. Gasquet, B. Olalinty and K. Currie, “An expert planning system for generating spacecraft mission plans”, *Proceedings of the International Conference on Expert Planning Systems*, 1990.
- [GAB 01] V. Gabrel, C. Murat and V. Paschos, La recherche opérationnelle pour aider à la planification de prises de vue par satellite, Document du LAMSADE, no. 121, 2001.

- [GAS 89] C. Gaspin, “Missions scheduling”, *J. Telematics and Informatics*, vol. 6, no. 3/4, 159–169, 1989.
- [GON 85] M. Gondran and M. Minoux, *Graphes et Algorithmes*, Eyrolles, Paris, 1985.
- [GOT 93] GOTHA, “Les problèmes d’ordonnancement”, *RAIRO-OR* 27, 77–150, 1993.
- [MOO 75] J.E. Moore, “An algorithm for a single machine scheduling problem with sequence dependent setup times and scheduling windows”, *AIIE Transactions*, p. 35–41, March 1975.
- [TES 88] C. Tessier-Badie, Contribution à l’étude des problèmes d’affectation de ressources et d’ordonnancement: application au domaine spatial, PhD thesis, ENSAE, Toulouse, 1988.
- [VAS 01] M. Vasquez and J.-K. Hao, “A ‘logic-constrained’ knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite”, *Journal of Computational Optimization and Applications*, vol. 20, p.137-157, 2001.

List of Authors

Wojciech BIENIA
G-SCOP
Ecole nationale supérieure
d'informatique et de mathématiques
appliquées de Grenoble
France

Nadia BRAUNER
G-SCOP
Joseph Fourier University
Grenoble
France

Alexandre CAMINADA
UTBM-SET
University of Technology of
Belfort Montbéliard
Belfort
France

Jacques DESROSIERS
Ecole des hautes études
commerciales
Montreal
Canada

Clarisse DHAENENS
Polytech'Lille
Villeneuve d'Ascq
France

Lionel DUPONT
Ecole des Mines d'Albi Carmaux
Albi
France

Marie-Laure ESPINOUSE
G-SCOP
Joseph Fourier University, IUT 1
Grenoble
France

Gerd FINKE
G-SCOP
Joseph Fourier University
Grenoble
France

Virginie GABREL
LIPN
Paris-Nord University
France

Jin-Kao HAO
LERIA
University of Angers
France

Alain HERTZ
GERAD
Ecole Polytechnique
Montreal
Canada

Jean-Luc LUTTON
CORE/CPN
France Telecom R&D
Issy les Moulineaux
France

Vincent MARTIN
Audience and Online Advertising
Line of Business – Orange
Cesson-Sévigné
France

Michel MITTAZ
CR/ARS
Institut de Recherches
Robert Bosch SA
Lonay
Switzerland

Cécile MURAT
LAMSADE
Paris-Dauphine University
France

Vangelis T. PASCHOS
LAMSADE
Paris-Dauphine University
France

Bernard PENZ
G-SCOP
Ecole nationale supérieure de génie
industriel
Grenoble
France

Index

A

ACCPM, 160
activity, 106
admissible basic solution, 7
admissible region, 4
Air Canada, 158
Air France, 157, 158
air transportation, 157
algorithm
 exponential, 78
 greedy, 39, 82
 list, 82
 of Dijkstra, 42
 of Kruskal, 38
 polynomial, 77
 priority rule, 82
 pseudo-polynomial, 77
allocation of resources, 1
altitude suite, 158
arc, 36
arc routing problem, 165
articulation point, 35
Augment-Merge algorithm, 184

B

backtracking, 92
basic solution, 7
basic variables, 7
basis, 7
Benders' decomposition, 160
binary variable, 22
branch and bound, 90

branch and bound procedure, 91
bombardier flexjet, 159
branch and bound, 253

C

canadian national railway, 159
canonical form, 5
capacity constraints, 170, 183
chain, 33
 alternating, 46
 augmenting, 47
 closed, 33
 elementary, 33
 simple, 33
Cheapest Insertion algorithm, 167
Chinese postman problem, 174
Christofides' algorithm, 168
circuit, 36
Clarke and Wright's algorithm, 171
class NP, 78
class P, 79
clique, 32, 246, 249
coloring
 minimum, 247
column generation, 138, 148, 159
complementary slack, 13
complexity, 27, 249
connected component, 34
constraints, 1
convex hull, 21
complexity, 77
 algorithmic, 77, 98
covering tour, 177

CPM, 108
 crew scheduling, 139
 cumulative constraint, 107
 cycle, 33

D

Dantzig-Wolfe decomposition, 150, 160
 Darsy, 157
 decision tree, 21
 decision variables, 2
 decomposition, 146
 degree, 30
 in, 36
 out, 36
 delay, 26
 determinant, 7
 discretisation, 236, 240, 260
 Disjunctive constraint, 107
 dual problem, 11
 dual variables, 12
 duration, 26

E

Earliest dates, 113
 edges, 30
 efficient labels, 141
 enchainability, 241, 242, 244
 ends, 30
 extension function, 143, 151
 extreme points. *See*

F

facets, 21
 feasible solutions, 3
 flow, 56
 flow variable, 142
 forest, 35
 Frederickson's algorithm, 178
 fundamental problem, 113

G

graph, 72
 bipartite, 32
 comparability, 247
 complementary, 247
 complete, 30
 complete bipartite, 32

connected, 34
 directed, 36
 Eulerian, 45
 Hamiltonian, 45
 induced, 32
 partial, 32
 partitioning, 73
 perfect, 49
 simple, 30
 undirected, 30
 Gantt chart, 118
 GencoL, 138, 157, 158
 generic algorithm, 143
 Giro, 157
 global constraints, 145

H

half-planes, 3
 hastus, 157
 heuristic, 98
 analysis of the average, 82
 construction method, 81
 Genetic algorithm, 89
 Gradient method, 84
 improvement method, 81
 local optimum, 84
 methods of exchange, 85
 neighborhood of a solution, 83
 Simulated annealing, 87
 tabu method, 86
 worst case, 82

I, K

image, 235
 inequations, 3
 integer variables, 19
 integrality, 20
 integrality constraints, 150
 knapsack problem, 74
 kronos canadian system, 158

L

label, 140
 Lagrangian relaxation, 160
 latest dates, 114
 length, 37
 linear programme, 2
 linear programming, 1

linear relaxation, 24, 148
 linear system, 6
 local restrictions, 145, 151
 Local Search techniques, 169, 172, 185
 loop, 30
 lower bound, 82, 91

M

margins, 116
 master problem, 148
 matching, 46
 perfect, 46
 maximal matching, 168, 175
 maximal tree, 168, 179
 maximisation, 5
 metaheuristic (see improvement method),
 240
 method
 exact, 98
 of decomposition, 82
 minimisation, 5
 minimum cost flow, 177
 mixed graph, 174
 monthly schedules, 139
 MPM, 108

N

nearest Neighbor algorithm, 166
 neighbors, 30
 network, 54
 non-basic variables, 7
 normalised form, 5
 NP-Complete, 78
 in the ordinary sense, 79
 in the strong sense, 79
 NP-Difficult, 79
 NP-hard, 166, 174, 177, 178
 node routing problem, 165

O

objective function, 1
 optimal basis, 7
 optimal solution, 3
 operating constraints, 139
 orbit, 235

P

path, 36
 elementary, 36
 simple, 36
 paths, 139
 penalty, 26
 PERT, 108
 pitching, 237
 pivoting, 7
 Polynomial, 78
 Polynomial reduction, 79
 precedence constraint, 107
 primal problem, 11
 probabilistic PERT, 119
 problem
 assignment, 73
 bin-packing, 74
 Chinese postman, 75
 decidable, 78
 decision, 78
 generalised assignment, 73
 indecidable, 78
 location, 75
 p-center, 75
 p-median, 76
 transportation, 75
 traveling salesman, 74, 81, 91, 93, 98
 vehicle routing, 74
 programming
 dynamic, 96, 99
 project scheduling, 105

R

rail transportation, 158
 relaxation, 81
 representation PERT/CPM, 117
 required arcs, 177
 resource, 106
 resource variables, 150
 restricted master problem, 148
 rolling, 237
 rural postman problem, 177

S

- saving, 170
- scheduling, 76
 - flowshop, 76
 - jobshop, 76
 - openshop, 76
 - team planning, 77
- scheduling methods, 108
- sensitivity intervals, 14
- sequencing with cumulative constraints, 126
- sequencing with disjunctive constraints, 123
- shortest path problem with time windows, 140
- simplex method, 4
- sink, 36
- slack variable, 6
- solution
 - approximate, 236
 - optimal, 96
- source, 36
- space-time network, 140, 149, 154, 155, 157, 159
- stable set, 32, 242
 - maximal, 250
- stability number, 246
- standard form, 6
- sub-problems, 148

T

- task, 106, 139
- temporal constraint, 107
- time-cost trade-off problem, 131
- time variable, 142
- time windows, 137, 139, 141, 144, 145, 151, 158, 159
- totally unimodular, 17
- transportation problem, 15
- travelling salesman problem, 166
- tree, 34
 - spanning, 37
- triangular inequality, 168, 178
- TSP (see travelling salesman problem)
- two-phase methods, 171

U, V

- upper bound, 92, 93
- urban transportation, 156
- vehicle routing, 139
- vertex, 21
- vertices, 30